Руководство по цифровой обработке сигналов

для научных работников и инженеров

второе издание

Не забудьте посетить вебсайт этой книги в интернете:

www.DSPguide.com

The Scientist and Engineer's Guide to

Digital Signal Processing

Second Edition

by
Steven W. Smith

California Technical Publishing
San Diego, California

Руководство по цифровой обработке сигналов

для научных работников и инженеров

второе издание

Стивен В. Смит

ПЕРЕВОД С АНГЛИЙСКОГО
Н.М. Котельниковой

Санкт-Петербург

2006

Руководство по цифровой обработке сигналов

для научных работников и инженеров

второе издание

Стивен В. Смит

ISBN 0-9660176-7-6 твердый переплет
ISBN 0-9660176-4-1 мягкий переплет
ISBN 0-9660176-6-8 электронная версия
LCCN 97-80293

California Technical Publishing
P.O. Box 502407
San Diego, CA 92150-2407

Для контактов с авторами через интернет:
вебсайт: www.DSPguide.com
e-mail: Smith@DSPguide.com

Отпечатано в Соединенных Штатах Америки
Первое издание 1997
Второе издание 1999

Беглое оглавление

БАЗИС

ОСНОВЫ

ЦИФРОВЫЕ ФИЛЬТРЫ

ПРИЛОЖЕНИЯ

КОМПЛЕКСНЫЕ МЕТОДЫ

Оглавление

КОМПЛЕКСНЫЕ МЕТОДЫ

Предисловие к русскому изданию

Уважаемый читатель!

Книга, которую Вы держите в своих руках, является единственным подобным изданием, которая на фоне многообразия литературы по цифровой обработке сигналов (ЦОС) выгодно отличается глубиной методической проработки, тщательным отбором и композицией материала, рассчитанными на самый широкий круг читателей. При изучении новой области знаний возникает вопрос – с чего начать? Если речь идет о цифровой обработке сигналов и первых практических шагах, особенно в области программирования сигнальных микропроцессоров, предлагаемая Вашему вниманию книга станет хорошим началом.

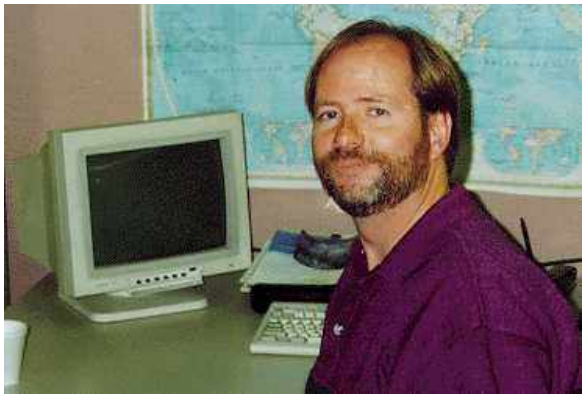
Настоящая книга, прежде всего, написана для специалистов занятых в различных областях науки и техники, не являющихся профессионалами в области ЦОС, но которые должны сталкиваться с ней по роду своей работы. Естественно, что таким специалистам важно понимать принципы работы систем с ЦОС, иметь представление об их возможностях и особенностях, о преимуществах и недостатках, а в некоторых случаях и самим создавать простейшие устройства ЦОС. Все это Вы найдете в этой книге.

Освещение самых различных вопросов, связанных с применением практических методов ЦОС, простота изложения, избегающая абстрактной теории, наличие большого числа примеров и иллюстраций делают это руководство полезным широкому кругу ученых, инженеров, преподавателей, аспирантов и студентов вузов. В то же время книга будет небесполезна и профессионалам практикам, которые найдут в ней описание большого количества практических приложений: цифровые фильтры, нейронные сети, сжатие данных, обработку звука и изображения и т.п.

Цифровая обработка сигналов одна из наиболее мощных технологий, которые формируют, и будут формировать науку и технику в XXI веке, и нет сомнений, что развитие ее методов в будущем приведет к еще более значительным результатам.

Пусть эта книга позволит Вам получить полезные знания и испытать чувство восторга за высоты, достигнутые человеческой мыслью.

Об авторе этой книги



Доктор философии ***Стивен Смит***, автор предлагаемой Вашему вниманию книги, является президентом и техническим директором Spectrum San Diego, Inc. Основная сфера деятельности Стивена Смита развитие новых систем получения рентгеновских изображений для медицины, средств обеспечения безопасности и промышленных приложений. Его самый большой проект был изобретение и продвижение на рынок системы *БЕЗОПАСНОСТЬ 1000* – системы безопасности для обнаружения оружия, взрывчатых веществ, и наркотиков скрытых под одеждой преступников и террористов. Чтобы выйти на рынок *БЕЗОПАСНОСТЬ 1000* потребовала пяти лет работы и более миллиона долларов вложений. До этого, по крайней мере, четыре компании предпринимали попытку развивать и продвигать на рынок системы такого типа. Критики технологии утверждали, что этого невозможно сделать! Как они заявляли, что даже если технические проблемы будут разрешены, система будет очень сложной в эксплуатации, и регламентирующие агентства никогда ее не одобрят, а публика будет в панике лишь от одной мысли быть подвергнутой рентгеновскому облучению. Ничто не говорит об “успехе” лучше, чем гора заказов на приобретение; в настоящее время в Соединенных Штатах и других странах используется приблизительно две дюжины систем *БЕЗОПАСНОСТЬ 1000*.

В области медицины доктор Смит разработал рентгеновские системы для обнаружения закупоренных сердечных артерий (атеросклероз) и обнаружения потери костями кальция (остеопороз).

Результаты его работ по неразрушающему контролю используются при проверке уровня заполнения и наличия утечек банок и бутылок, а также при проверке дефектных паяных соединений на печатных платах.

Доктор Смит автор еще одной книги “The Inner Light Theory of Consciousness”, изданной в 2001 году.

Предисловие

Цели и стратегии этой книги

Технический мир изменяется очень быстро. Только за последние 15 лет производительность персональных компьютеров увеличилась почти в *тысячу раз*. По всем подсчетам она увеличится еще в тысячу раз в следующие 15 лет. Такая огромная мощность изменила пути, которыми шли наука и инженерная практика, и этому нет лучшего примера, чем Цифровая Обработка Сигналов. В начале 1980-х ЦОС преподавалась как курс уровня дипломированного специалиста в области электроинженерии. Десятилетием позже ЦОС стала стандартной частью учебного плана подготовки студентов. Сегодня ЦОС – *основной навык* необходимый ученым и инженерам во многих областях. К сожалению, образование в области ЦОС происходит медленно, чтобы адаптироваться к этим изменениям. Почти все учебники по ЦОС написаны в традиционном электроинженерном стиле детальной и строгой математики. ЦОС невероятно мощна, но если Вы не можете понять ее, Вы не можете пользоваться ею.

Эта книга была написана для ученых и инженеров, работающих в широко варьирующихся областях: физики, биоинженерии, геологии, океанографии, механики и электричества, и это всего несколько перечисленных сфер. Цель книги состоит в том, чтобы представить практические методы, почти избежав барьеров детальной математики и абстрактной теории. Для достижения этой цели, при написании этой книги, были использованы три стратегии:

Первое, справедливость *объясняемых* методов, не просто доказывается через математические выкладки. Хотя математики содержится достаточно много, она не используется как первичное средство передачи информации. Ничто не может сравниться с несколькими хорошо написанными параграфами, поддержанными хорошими иллюстрациями.

Второе, к изучению *комплексных чисел* обращаются как к дополнительной теме после того, как усвоены фундаментальные принципы. Главы 1-29 объясняют все базовые методы, используя только алгебру, и в редких случаях небольшое число элементарных вычислений. Главы 30-33 показывают, как комплексная математика расширяет возможности ЦОС, представляя методы, которые не могут быть использованы только с действительными числами. Многим такой подход покажется ересью! Традиционные учебники по ЦОС изобилуют комплексной математикой, часто начиная прямо с первой главы.

Третье, используются очень простые компьютерные программы. Большинство компьютерных программ по ЦОС написаны на C, FORTRAN или подобных языках программирования. Однако изучение ЦОС и использование ЦОС имеют разные требования. Студенту нужно сконцентрироваться на алгоритмах и методах, без того чтобы отвлекаться на причуды конкретного языка. Производительность и гибкость здесь не важна, простота же является критической. Программы в этой книге написаны для изучения ЦОС наиболее целенаправленным способом, при этом все другие факторы рассматриваются как второстепенные. Там, где требуется сделать программную логику более ясной, хороший стиль программирования игнорируется. Например:

- используется упрощенная версия языка BASIC
- приведены номера строк
- в цикле FOR-NEXT используются только управляемые структуры
- не приводятся выражения ввода/вывода

Это наиболее простой стиль программирования, который я смог найти. Некоторые могут подумать, что эта книга была бы лучше, если бы программы были написаны на C. Ну что же, больше я не могу не соглашаться.

Аудитория для которой предназначена эта книга

Эта книга, прежде всего, предназначена для годичного курса по практической ЦОС со студентами, привлеченными из широко варьирующихся областей науки и инженерной практики. Предполагаемыми предпосылками являются:

- Курс практической электроники: (операционные усилители, R-C цепи и т.д.)
- Курс программирования (Фортран или что-то подобное)
- Один год численных методов

Эта книга была также написана в расчете на практиков-профессионалов. В ней обсуждается большое количество повседневных приложений: цифровые фильтры, нейронные сети, сжатие данных, обработка звука и изображения и т.д. Эти главы являются самостоятельными в максимально возможной степени и не требуют от читателя просмотра всей книги для решения конкретной проблемы.

Поддержка комплектующими Analog Devices

Второе издание этой книги включает две новые главы по *Цифровым Сигнальным Процессорам*, микропроцессорам, специально сконструированным для решения задач ЦОС. Большинство информации для этих глав было великодушно предоставлено Analog Devices, Inc. (объединение Аналоговые Приборы - прим. ред. перев.), мировым лидером в разработке и производстве электронных компонентов для обработки сигналов. Поощрение и поддержка ADI значительно расширила возможности этой книги, показывая, что алгоритмы ЦОС можно использовать только совместно с соответствующими аппаратными средствами.

Признательности

Особая благодарность большому числу рецензентов, высказавших свои замечания и предложения по этой книге. Щедрый дар их времени и мастерства сделал эту работу лучше: **Магнус Аронсон** (факультет электроинженеров, университет Юты); **Брюс Б. Азими** (военно-морской флот США); **Вернон Л. Чи** (факультет компьютерных наук, университет Северной Каролины); **Манохар Дас**, **доктор философии** (факультет электрического и системного инженерного дела, Оклендский университет); **Кэрол А. Дин**, (объединение Analog Devices); **Фред де Пиеро**, **доктор философии**, (электротехнический факультет, государственный университет Кэлполи); **Хосе Фридман**, **доктор философии**, (объединение Analog Devices); **Фредерик К. Дюеннебиер**, **доктор философии**, (факультет геологии и геофизики, Гавайский университет, Маноа); **Д. Ли Фугал**, (Технологии космоса и сигналов); **Фильсон Х. Глэнз**, **доктор философии**, (факультет электрического и компьютерного инженерного дела, университет Нью Гемпшира); **Кеннет Х. Джеккер**, (факультет компьютерных наук, Аппалачинский государственный университет); **Раджив Кападиа**, **доктор философии**, (электротехнический факультет, государственный университет в Манкато); **Дан Кинг**, (объединение Analog Devices); **Кевин Лири**, (объединение Analog Devices); **А. Дэйл Магоун**, **доктор философии**, (факультет компьютерных наук, университет Северной Луизианы); **Бен Мбугуа**, (объединение Analog Devices); **Бернард Дж. Максум**, **доктор философии**,

(электротехнический факультет, университет в Ламаре); **Пол Морган**, **доктор философии**, (факультет геологии, университет Северной Аризоны); **Дэйл Х. Маглер**, **доктор философии**, (факультет математических наук, Экронский университет); **Кристофер Л. Маллен**, **доктор философии**, (факультет гражданского инженерного дела, университет Миссисипи); **Синтия Л. Нельсон**, **доктор философии**, (национальные лаборатории в Сандии); **Бронислава Перуничик-Драженович**, **доктор философии**, (электротехнический факультет, университет в Ламаре); **Джон Шмелк**, **доктор философии**, (факультет математических наук, университет общей медицины Виржинии); **Ричард Р. Шульц**, **доктор философии**, (электротехнический факультет, университет Северной Дакоты); **Давид Школьник**, (объединение Analog Devices); **Джей Л. Смит**, **доктор философии**, (центр аэрокосмических технологий, государственный университет Вебера); **Джефри Смит**, **доктор философии**, (факультет компьютерных наук, университет Джорджии); **Оскар Янес Суарес**, **доктор философии**, (электротехнический факультет, Столичный университет, университетский городок Истапалапа, деловая часть Мехико) и другие рецензенты, пожелавшие остаться не названными.

Сейчас эта книга в руках конечного рецензента, которым являетесь Вы. Пожалуйста, найдите время дать мне свои комментарии и предложения. Это позволит будущим переизданиям и новым изданиям служить Вашим целям значительно лучше. Все, что для этого потребуется, это двух минутное сообщение по электронной почте на адрес: Smith@DSPguide.com. Спасибо. Я надеюсь, книга доставит Вам удовольствие.

Стив Смит
Январь 1999

ГЛАВА

1

ШИРОТА И ГЛУБИНА ЦОС

Цифровая обработка сигналов одна из наиболее мощных технологий, которые будут формировать науку и технику в двадцать первом веке. Уже произошли революционные изменения в *широком* диапазоне областей: связь, изображения в медицине, радиолокация и гидролокация, высококачественное воспроизведение музыки, разведка нефти, и это только несколько перечисленных приложений. Каждая из этих областей развивалась *глубокой* ЦОС технологией со своими собственными алгоритмами, математикой и специализированной техникой. Такая комбинация широты и глубины делает невозможным для любого индивидуума овладеть всеми разработанными технологиями ЦОС. Изучение ЦОС включает две задачи: изучение основных концепций, которые применимы к области в целом, и изучение специализированной техники для специфических интересующих Вас областей. Эта глава начинает наше путешествие в мир Цифровой Обработки Сигналов с описания впечатляющих эффектов, которые ЦОС произвела в различных областях. Революция началась.

Корни ЦОС

Цифровая Обработка Сигналов отличается от других областей в компьютерной науке уникальным типом используемых данных – *сигналами*. В большинстве случаев эти сигналы являются измеренными данными реального мира: сейсмические колебания, визуальные изображения, звуковые волны и т.д. ЦОС это математика, алгоритмы и техника манипулирования этими сигналами после того, как они были преобразованы в цифровую форму. Она включает широкое разнообразие целей таких как: улучшение видео изображений, распознавание и генерирование речи, сжатие данных для хранения и передачи и т.д. Предположим, что мы подключили к компьютеру аналого-цифровой преобразователь и использовали его для получения порции данных из реального мира. ЦОС отвечает на вопрос: "*А что дальше*?".

ЦОС уходит корнями в 1960-е и 1970-е годы, когда впервые стали доступными цифровые вычислительные машины. В эту эру компьютеры были очень дорогими, и ЦОС была ограничена только несколькими критическими применениями. Пионерские усилия были сделаны в четырех ключевых областях: *радио-* и *гидролокация*, где подвергалась риску национальная безопасность, *поиск нефти*, где могло быть получено большое количество денег, *исследование космоса*, где данные являются незаменимыми, *изображения в медицине*, где могли быть спасены жизни. Революция персональных компьютеров 1980-х и 1990-х стала причиной взрыва ЦОС новыми применениями. Внезапно коммерческий рынок повел ЦОС быстрее, чем ЦОС мотивировалась военными и правительственными нуждами. Все, кто думал, что сможет сделать деньги в быстро расширяющейся области, вдруг стал распространителем ЦОС. ЦОС обогатила общество такими продуктами как: мобильные телефоны, проигрыватели компакт дисков и электронная голосовая почта. Рис. 1.1 иллюстрирует некоторые из этих разнообразных применений.

Эта технологическая революция происходила сверху вниз. В ранние 1980-е ЦОС преподавалась как дисциплина для студентов *старших* курсов при подготовке электроинженеров. Десять лет спустя ЦОС стала стандартной частью учебного плана для студентов младших курсов. Сегодня ЦОС является *основой мастерства*, требуемого ученому и инженеру в различных областях. Если проводить аналогию, ЦОС может сравниться с первой технологической революцией – *электроникой*. Оставаясь сферой электроинженеров, почти каждый ученый и инженер имеет некоторый базис по основам проектирования электронных схем. Без него они бы потерялись в технологическом мире. ЦОС ожидает такое же будущее.

Рис. 1.1 Некоторые из применений ЦОС

Экскурс в недавнюю историю ЦОС - более чем любопытство, история имеет огромное влияние на *Вашу* способность изучать и использовать ЦОС. Предположим, что Вы сталкиваетесь с проблемой ЦОС, и для того чтобы найти решение, открываете учебник или другую публикацию. То, что Вы, как правило, там найдете, это будут уравнения страница за страницей, непонятные математические символы и незнакомая терминология. Ночной кошмар! Большинство литературы по ЦОС сбивает с толку даже тех, у кого есть практический опыт в этой области. И не то чтобы этот материал был неправилен, просто он предназначен для очень специализированной аудитории. Искусным исследователям для понимания теоретического вклада работы нужна детальная математика.

Основное предназначение этой книги в том, чтобы большинство практических методов ЦОС можно было бы изучить и использовать без традиционных барьеров подробной математики и теории. *Руководство по цифровой обработке сигналов для научных работников и инженеров* написано для тех, кто хочет использовать ЦОС не как новое *занятие*, а как *инструмент*.

Остаток этой главы иллюстрирует области, где ЦОС произвела революционные изменения. По мере знакомства с каждым приложением обратите внимание, что ЦОС *междисциплинарна* и опирается на большое число смежных областей. Как показано на рис. 1.2, границы между ЦОС и другими дисциплинами не являются четкими и строго определенными, а скорее размыты и перекрываются. Если Вы хотите специализироваться в ЦОС вот смежные области, которые Вам тоже понадобится изучить.

Рис. 1.2 Границы между ЦОС и другими дисциплинами

Связь

Связь это передача информации из одного места в другое. Она включает в себя большое количество форм информации: телефонные разговоры, телевизионные сигналы, компьютерные файлы и другие типы данных. Для передачи информации требуется *канал* между связываемыми местами. Это может быть пара проводов, радиосигнал, оптическое волокно и т.д. Телекоммуникационные компании получают *оплату* за передачу информации их клиентов, а те *платят* за создание и обслуживание канала связи. Итоговая финансовая строка очень проста: чем больше информации они могут передать по одному каналу, тем больше они сделают денег. ЦОС совершила революцию во многих областях телекоммуникационной индустрии: генерирование и детектирование тональных сигналов, сдвиг полосы частот, фильтрация для удаления фона линий электропередачи и т.д. Здесь будут обсуждены три специфических примера взятых из телефонных сетей: мультиплексирование данных, сжатие данных и подавление эха.

Мультиплексирование данных

В мире приблизительно *один миллиард* телефонов. Нажатием нескольких кнопок телефонные сети позволяют одному абоненту соединится с любым другим всего за нескольких секунд. Необъятность этой задачи вызывает головную боль. До 1960-х годов соединение между двумя телефонами требовало прохождения голосового аналогового сигнала через механические переключатели и усилители. Одно соединение требовало одну пару проводов. Для сравнения, ЦОС преобразует звуковой сигнал в поток

последовательных цифровых данных. Поскольку биты могут быть легко перемешены, а позже разделены, по одному каналу может быть передано большое количество телефонных разговоров. Например, телефонный стандарт известный как *система Т-несущей* может одновременно передавать двадцать четыре голосовых сигнала. Используя компандированное (с логарифмическим сжатием) аналого-цифровое преобразование, каждый голосовой сигнал подвергается дискретизации 8000 раз в секунду. В результате в каждом голосовом сигнале представлено 64000 бит/сек, а все двадцать четыре канала содержатся в 1544 мегабит/сек. Этот сигнал, при использовании обычного телефонного кабеля 22 типоразмера с медными проводами, может быть передан на расстояние приблизительно около 6000 футов, что является типичной дистанцией для соединения. Финансовое преимущество цифровой передачи огромно. Провода и аналоговые ключи дороги, цифровые логические элементы дешевы.

Сжатие данных

Когда голосовой сигнал оцифровывается на 8000 отсчетов/сек, большая часть цифровой информации является *избыточной*. То есть информация, которую несет какой-либо отсчет, в значительной степени дублируется соседними выборками. Для преобразования оцифрованных голосовых сигналов в потоки данных, которые требуют меньшего количества бит/сек, было разработано множество алгоритмов ЦОС. Они называются алгоритмами **сжатия данных**. Соответственно **восстанавливающие** алгоритмы используются для воссоздания сигнала в его первоначальной форме. Эти алгоритмы варьируются по величине достигаемого сжатия и качеству звука. Вообще говоря, снижение соотношения данных с 64 килобит/сек до 32 килобит/сек не приводит к существенной потере качества звука. Когда же происходит сжатие до соотношения данных 8 килобит/сек, звук заметно искажается, но его еще можно использовать в протяженных телефонных сетях. Максимально достижимое сжатие - около 2 килобит/сек, в результате звук становится сильно искаженным, но может использоваться в некоторых приложениях, таких как военная и подводная связь.

Подавление эха

Эхо представляет серьезную проблему при осуществлении телефонных соединений большой длины. Когда Вы говорите по телефону, сигнал представляющий Ваш голос движется к приемнику, с которым установлена связь, там часть его отражается и возвращается в виде эха. Если связь - в пределах нескольких сотен миль, то конечное время до получения эха составляет всего несколько миллисекунд. Уху человека привычно слышать эхо с такими маленькими задержками, и звуки при связи воспринимаются вполне нормально. По мере увеличения расстояния эхо становится все более заметным и раздражающим. При межконтинентальной связи задержка может составлять несколько сотен миллисекунд и особенно нежелательна. Цифровая Обработка Сигналов атакует такой тип проблем, измеряя отраженный сигнал и генерируя для подавления преступного эха соответствующий *антисигнал*. Эта же техника позволяет пользователям громкоговорящих телефонных аппаратов слышать и говорить в одно и тоже время без того, чтобы бороться со звуковой обратной связью (визжанием). Она может также использоваться для снижения шумов окружающей среды, подавляя их сгенерированными в цифровой форме *антишумами*.

Обработка звука

Зрение и слух - два основных чувства человека. Соответственно, многое в ЦОС связано с обработкой изображения и звука. Люди слушают и *музыку*, и *речь*. ЦОС произвела революционные изменения в обеих этих областях.

Музыка

Путь, пролегающий от микрофона музыканта до динамика аудиофила, исключительно долог. Цифровое представление данных существенно для предотвращения ухудшения характеристик, связываемого обычно с аналоговыми способами хранения и манипуляции. Любому, кто имел возможность сравнивать качество музыки на кассетах и компакт дисках, это очень знакомо. В типичных сценариях музыкальный фрагмент записывается в студии звукозаписи на множество каналов или дорожек. В некоторых случаях это даже позволяет раздельно записывать индивидуальные инструменты и певцов. Это делается для того, чтобы обеспечить более высокую гибкость инженеру звукозаписи в создании конечного продукта. Сложный процесс объединения индивидуальных дорожек в конечный продукт называется *микшированием*. ЦОС может обеспечить несколько важных функций в процессе микширования, включая: фильтрацию, сложение и вычитание сигналов, редактирование сигнала, и т.п.

Одно из наиболее интересных приложений ЦОС в подготовке музыки – *искусственная реверберация*. Если индивидуальные каналы просто складываются вместе, результирующий звуковой фрагмент - хилый и растворенный, такой, как если бы музыканты играли за дверью. Это происходит потому, что на слушателя сильно влияет содержащееся в музыке эхо или реверберация, которая в музыкальной студии обычно минимизируется. ЦОС позволяет добавлять во время микширования искусственное эхо и реверберацию, моделируя различное идеальное слуховое окружение. Эхо с задержкой несколько сотен миллисекунд дает впечатление нахождения как будто в соборе. Добавление эха с задержкой 10-20 миллисекунд обеспечивает восприятие комнаты для прослушивания более скромного размера.

Генерирование речи

Генерирование и распознавание речи используется для взаимодействия человека с машиной. Вместо того чтобы использовать Ваши руки и глаза, Вы пользуетесь Вашим ртом и ушами. Очень удобно, когда ваши руки и глаза могут делать что-нибудь еще, похожее на: вождение автомобиля, хирургическую операцию или (к сожалению) стрельбу из Вашего оружия по врагу. Для генерирования компьютерной речи используется два подхода: *цифровая запись* и *симулирование голосового тракта*. При цифровой записи звук речи человека оцифровывается и хранится, обычно в сжатом виде. При воспроизведении хранимые данные восстанавливаются и обратно преобразуются в аналоговый сигнал. Один час записанной речи требует около трех мегабайт памяти, это неплохо даже для систем с характеристиками небольших компьютеров. Это наиболее обычный метод цифрового генерирования речи, встречающийся сегодня.

Симуляторы голосового тракта более сложны и пытаются имитировать физические механизмы, с помощью которых люди создают речь. Голосовой тракт человека это акустическая полость с резонансными частотами, определенными размером и формой камеры. Звук возникает в голосовом тракте одним из двух основных способов, названных *звонкими* и *фрикативными* звуками. При звонких звуках вибрация гортани создает близкие к периодическим импульсы воздуха в голосовых полостях. Для сравнения, фрикативные звуки возникают из турбулентности шумящего воздуха на сжатых конструкциях, таких как зубы и губы. Симуляторы звукового тракта работают, генерируя цифровые сигналы, которые похожи на эти два типа возбуждения. Характеристики резонансной камеры имитируются прохождением возбужденного сигнала через цифровые фильтры с подобными резонансами. В одних из самых ранних историй с успешным использованием ЦОС этот подход был использован в широко продаваемой обучающей электронике для детей, *Произноси и пиши*.

Распознавание речи

Автоматическое распознавание речи человека чрезвычайно труднее, чем генерирование речи. Распознавание речи является классическим примером вещей, которые мозг человека делает хорошо, а цифровые компьютеры делают плохо. Цифровые компьютеры могут хранить и перезаписывать огромное количество данных, со скоростью молнии производить математические вычисления и без устали выполнять повторяющиеся задачи, не снижая производительности. К сожалению, сегодняшние компьютеры действуют очень плохо, когда сталкиваются с необработанными измеряемыми данными. Проще обучить компьютер посылать Вам ежемесячный счет за электричество. Обучить такой компьютер понимать Ваш голос – большая проблема.

Цифровая Обработка Сигнала в основном подходит к проблеме распознавания голоса в два шага: *выделение характеристик*, за которым следует *сравнение характеристик*. Каждое слово входного звукового сигнала изолируется, а затем анализируется с целью определения типа возбуждения и резонансных частот. Затем эти параметры сравниваются с предварительно взятыми примерами произнесенных слов для определения ближайшего совпадения. Часто такие системы ограничиваются использованием только нескольких сотен слов, могут принимать речь только с отчетливыми паузами между словами и для каждого говорящего должны быть индивидуально повторно обучены. Хотя все это и адекватно для многих коммерческих приложений, такие ограничения делают все это очень скромным при сравнении с возможностями слуха человека. Работы, которую нужно проделать в этой области еще очень много, с огромным финансовым вознаграждением тем, кто произведет успешный коммерческий продукт.

Эхолокация

Самый обычный способ получения информации об удаленном объекте это отразить от него фронт волны. Например, работа радара заключается в передаче импульса радиоволн и проверке принятого сигнала на наличие эха от самолета. В сонарах для обнаружения подводных лодок и других подводных объектов сквозь воду передается звуковая волна. Геофизики давно исследуют землю, производя взрывы и прослушивая эхо от глубоко залегающих слоев пород. Хотя эти приложения имеют общую нить, каждое обладает своими собственными специфическими проблемами и нуждами. Цифровая Обработка Сигналов произвела революционные изменения во всех трех областях.

Радар

Радар это акроним от *RAdio Detection And Ranging* (радио обнаружение и определение дальности - прим. перев.). В простейших радарных системах радиопередатчик вырабатывает импульс радиочастотной энергии длительностью несколько микросекунд. Этот импульс подается на высоконаправленную антенну, откуда со скоростью света распространяется прочь образовавшаяся радио волна. Самолет, находящийся на пути этой волны отразит небольшую часть энергии обратно по направлению к приемной антенне, находящейся рядом с местом передачи. Расстояние до объекта вычисляется из времени прошедшего между переданным импульсом и полученным эхом. Направление на объект находится более просто: Вы знаете, *куда* Вы навели направленную антенну в момент, когда было получено эхо.

Рабочая дальность радарных систем определяется двумя параметрами: величиной энергии первоначального импульса и уровнем шума радиоприемника. К сожалению, увеличение энергии импульса требует увеличения его *длительности*. В свою очередь, продолжительность импульса снижает достоверность и точность измерения прошедшего времени. В результате возникает конфликт между двумя важными параметрами:

способностью обнаружить объект на большом расстоянии и способностью точно определить расстояние до объекта.

ЦОС произвела революцию радаров в трех областях, каждая из которых относится к этой основной проблеме. Первое, ЦОС может *сжимать* импульс, после того как он принят, обеспечивая более хорошее определение расстояния без снижения рабочей дальности. Второе, ЦОС может выполнять фильтрацию принятого сигнала для снижения шума. Это увеличивает дальность без ухудшения способности определения расстояния. Третье, ЦОС обеспечивает быструю селекцию и генерирование импульсов различной формы и длительности. Среди прочего, это позволяет оптимизировать импульсы для специфических задач обнаружения. А теперь самая впечатляющая часть: почти все это проделывается при частотах выборки сравнимых с используемыми радиочастотами, столь же высокими, порядка нескольких сотен мегагерц! Что касается радаров, ЦОС - это и масса всего о разработке высокоскоростной аппаратной части, и об алгоритмах.

Сонар

Сонар это акроним от *SOund NAvigation and Ranging* (звуковая навигация и определение дальности - прим. перев.). Сонары делятся на две категории, *активные* и *пассивные*. В активных сонарах в воде передаются звуковые импульсы между 2 кГц и 40 кГц, и обнаруживается и анализируется полученное эхо. Использование активных сонаров включает: обнаружение и локализацию подводных тел, навигацию, связь и картографирование морского дна. Типичная максимальная рабочая дальность от 10 до 100 километров. Для сравнения, пассивные сонары просто *прослушивают* подводные звуки, которые включают: природные турбулентности, морскую жизнь и механические шумы от подводных лодок и судов на поверхности. Поскольку пассивный сонар не излучает энергии, он является идеальным для тайных операций. Вы хотите обнаружить *другого парня*, при этом чтобы он не обнаружил *Вас*. Наиболее важное применение пассивных сонаров в военных системах наблюдения, обнаруживающих и следящих за подводными лодками. Пассивные сонары обычно используют более низкие частоты, чем активные сонары, поскольку они распространяются в воде с меньшим затуханием. Расстояния обнаружения могут составлять тысячи километров.

ЦОС революционизировала сонары во многих из тех областей, что и радары: генерирование импульсов, сжатие импульсов и фильтрация обнаруженных сигналов. На первый взгляд из-за используемых более низких частот сонары *проще*, чем радары. С другой точки зрения, сонары более *сложны*, чем радары, поскольку окружающая среда гораздо менее однообразна и стабильна. Гидролокационные системы обычно используют пространственные множества передающих и принимающих элементов, в отличие от просто использования единственного канала. Соответствующим образом, управляя и смешивая сигналы этих многих элементов, гидролокационная система может направить испускаемый импульс к желаемому месту и определить направление, откуда пришло эхо. Чтобы справляться с такой многоканальностью, гидролокационная система требует такой же огромной вычислительной мощности ЦОС, как для радара.

Эхо сейсмология

На заре 1920-х геофизики открыли, что структура земной коры может быть исследована с помощью звука. Геологоразведчики могли осуществлять взрыв и записывать эхо от пограничных слоев из глубины более десяти километров от поверхности. Чтобы составить карту подпочвенной структуры эта эхо сейсмограмма интерпретировалась невооруженным глазом. Метод сейсмических отражений быстро стал первостепенным методом для определения месторождений нефти, минеральных отложений и остается таким и сегодня.

В идеальном случае звуковой импульс, посланный в землю, вызывает отдельное эхо от каждого пограничного слоя, через который проходит импульс. К сожалению

ситуация не так проста как обычно. Каждое эхо, возвращающееся к поверхности, должно пройти через все другие пограничные слои, лежащие выше места, где возникло эхо. Это может привести к возникновению эха, циркулирующего межу слоями туда-сюда, увеличивая возможность обнаружения на поверхности *эха от эха*. Эти вторичные эхо могут сделать обнаруженный сигнал очень сложным и трудно интерпретируемым. С 1960-х, для отделения в эхо сейсмограммах первичного эха от вторичных, широко использовалась Цифровая Обработка Сигнала. Как же первые геофизики обходились без ЦОС? Ответ прост: они искали в *легких* местах там, где многократные отражения были минимальны. ЦОС позволяет находить нефть в *трудных* местах, таких как дно океана.

Обработка изображений

Изображения это сигналы со специальными характеристиками. Прежде всего, они являются величиной изменения параметра в *пространстве* (в зависимости от местоположения), в то время как большинство сигналов являются величиной изменения параметра во *времени*. Во вторых, они содержат огромное количество информации. Например, для хранения одной секунды телевизионного видео может понадобиться более 10 мегабайт. Это более чем в тысячу раз больше, чем для звукового сигнала такой же длины. В третьих, конечная оценка качества часто скорее является субъективной оценкой человека, чем объективным критерием. Такие специфические характеристики сделали обработку изображения обособленной подгруппой в пределах ЦОС.

Медицина

В 1895 Вильгельм Конрад Рентген открыл, что х-лучи могут проходить через существенное количество вещества. Способность заглянуть внутрь тела живого человека произвела революцию в медицине. В течение всего нескольких лет медицинские системы х-лучей распространились по всему миру. Несмотря на их очевидный успех, до тех пор пока в 1970-х не пришли ЦОС и связанные с ней методы, использование медицинского изображения х-лучей (рентгеновских снимков - прим. перев.) ограничивалось четырьмя проблемами. Первое, перекрывающиеся структуры в теле могут загораживать друг друга. Например, части сердца не могут просматриваться позади ребер. Второе, не всегда удается провести различие между подобными тканями. Например, можно отличить кость от мягких тканей, но опухоль с печенью неразличимы. Третье, изображения, поученные с помощью х-лучей, показывают *анатомию* - строение тела, а не физиологию – функционирование тела. Изображение живого человека, поученное с помощью х-лучей, выглядит точно так же, как полученное с помощью х-лучей изображение мертвого человека. Четвертое, облучение х-лучами может стать причиной возникновения рака, поэтому необходимо, что бы их использование было щадящим и оправданным.

Проблема перекрывающихся структур была решена в 1971 с введением первого сканера для компьютерной томографии (формально называемого томограф с вычислением по осям, или *КАТ* сканнер (компьютерный аксиальный томограф – прим. перев.)). Компьютерная томография - это классический пример Цифровой обработки сигналов. Х-лучи пропускаются через исследуемый участок тела пациента с нескольких направлений. Вместо того чтобы просто сформировать изображение с помощью обнаруженных х-лучей, сигнал преобразовывается в цифровую форму и заносится в компьютер. Затем информация используется для *вычисления* изображений, которые являются *срезами исследуемого участка*. Такие изображения показывают значительно больше деталей, чем традиционная техника, позволяя значительно лучше диагностировать и лечить. Влияние компьютерной томографии было почти такого же масштаба, как первое появление самих изображений, полученных с помощью х-лучей. В течение всего нескольких лет доступ к сканеру компьютерной томографии получила каждая солидная больница в мире. В 1979 Годфри Н. Хоунсфилд и Алан М. Кормак, внесшие основной вклад в развитие

компьютерной томографии, разделили Нобелевскую премию в области медицины. *Вот это – классная ЦОС!*

Последние три проблемы х-лучей были решены за счет использования другой, чем у х-лучей проникающей энергии, такой как у радио и звуковых волн. ЦОС играет ключевую роль во всех этих методиках. Например, магниторезонансная интроскопия (МРИ) для исследования внутренностей человеческого тела использует магнитные поля в сочетании с радиоволнами. Поля, с должным образом подобранными силой и частотой, становятся причиной резонанса атомных ядер между энергетическими квантовыми уровнями в локализованной области тела. Результатом этого резонанса становится эмиссия вторичных радиоволн, обнаруживаемых антенной, расположенной рядом с телом. Сила и другие характеристики этого обнаруженного сигнала обеспечивают информацию о локализованной резонирующей области. Управление магнитным полем позволяет резонирующей области сканировать тело, картографируя его внутреннюю структуру. Такая информация обычно представляется в виде изображений, точно так же, как и в компьютерной томографии. Помимо обеспечения превосходной дифференциации между различными типами мягких тканей МРИ может давать информацию относительно физиологии такой, как поток крови через артерии. МРИ полностью базируется на технике Цифровой Обработки Сигнала и без нее не может быть реализован.

Космос

Иногда Вам удается сделать самую плохую из всех существующих фотографий. Это частый случай для изображений, полученных от беспилотных спутников и космических исследовательских планетоходов. Никто не собирается из-за этого посылать ремонтника к Марсу только для того, чтобы потом пощелкать кнопками на камере! ЦОС может улучшить качество изображений, сделанных при чрезвычайно неблагоприятных условиях, несколькими способами: регулированием яркости и контрастности, выделением контуров изображения, снижением шума, регулированием фокуса, уменьшением смазанности вызванной движением и т.д. Изображения, которые имеют пространственное искажение подобное тому, с которым сталкиваются, когда снимается плоское изображение сферической планеты, могут также быть *деформированы* до правильного их представления. Большое число индивидуальных изображений могут быть объединены в единую базу данных, позволяющую воспроизводить информацию уникальными способами. Например, видеопоследовательностью, моделирующей воздушный полет над поверхностью удаленной планеты.

Коммерческие видео продукты

Большой объем информации, содержащийся в изображениях - проблема для систем, продаваемых широкой публике в массовых количествах. Коммерческие системы должны быть *дешевы*, и не должны попасть в западню большой памяти и высокой скорости передачи данных. Единственным ответом на эту дилемму является *сжатие изображения*. Так же, как и звуковые сигналы, изображения содержат большое количество избыточной информации и могут быть пропущены через алгоритмы уменьшающие число бит, необходимых для представления этих изображений. Телевизионные и другие кино картины особенно подходят для сжатия, поскольку большая часть изображения от кадра к кадру не изменяется. Коммерческие видео продукты, которые воспользовались преимуществами этой технологии, включают в себя: видеотелефоны, компьютерные программы, воспроизводящие движущиеся картины, и цифровое телевидение.

ГЛАВА

2

СТАТИСТИКА, ВЕРОЯТНОСТЬ И ШУМЫ

Для того чтобы дать характеристику сигналам и процессам, в результате которых эти сигналы генерируются, в Цифровой Обработке Сигналов, используется статистика и теория вероятности. Первостепенным применением ЦОС является, например, снижение в полученных данных помех, шумов и других нежелательных компонентов. Эти компоненты могут быть частью свойственной измеряемому сигналу, являющейся результатом дефектов в системе получения данных, или могут быть внесены как неизбежный побочный продукт некоторых операций ЦОС. Статистика и теория вероятности позволяют измерить и классифицировать эти разрушительные характеристики, что является первым шагом в разработке стратегии по удалению компонент правонарушителей. Эта глава знакомит с наиболее важными понятиями в статистике и теории вероятности, подчеркивая то, как они применяются к полученным сигналам.

Терминология сигналов и временных диаграмм

Сигнал - это описание того, как один параметр соотносится с другим. Например, наиболее обычным типом сигнала в аналоговой электронике является *напряжение,* которое изменяется *во времени*. Поскольку оба параметра здесь могут принимать непрерывный ряд значений, мы будем называть это **непрерывным сигналом**. Для сравнения, прохождение этого сигнала через аналого-цифровой преобразователь делает каждый из этих двух параметров *квантованным*. Представим, например, что было выполнено 12 битовое преобразование со скоростью выборки 1000 отсчетов в секунду. Теперь, число возможных двоичных уровней напряжения сокращается до 4096 (2^{12}), а время определено только через интервалы в одну миллисекунду. Говорят, что сигналы, сформированные из параметров, которые квантованы таким образом, являются **дискретными сигналами** или **оцифрованными сигналами**. Непрерывные сигналы главным образом существуют в природе, в то время как дискретные сигналы существуют внутри компьютера (хотя, Вы можете найти исключение в обоих случаях). Возможно также иметь сигналы, где один параметр непрерывный, а другой дискретный. Поскольку такие смешанные сигналы весьма необычны, у них нет присвоенного им специального названия, и природа двух параметров должна быть заявлена ясно.

На рис. 2.1 показаны два дискретных сигнала такие же, какие могут быть получены с помощью цифровой системы сбора данных. Вертикальная ось может представлять напряжение, интенсивность света, давление звука или бесконечное число других параметров. Поскольку мы не знаем, что она представляет в данном частном случае, мы дадим ей обобщенное обозначение: **амплитуда**. Этот параметр также называют несколькими другими именами: **ось *y***, **зависимая переменная**, **диапазон** и **ордината**.

Горизонтальная ось представляет другой параметр сигнала и сопровождается такими именами как: **ось *x***, **независимая переменная**, **область** и **абсцисса**. Наиболее обычный параметр, появляющийся на горизонтальной оси полученных сигналов - в*ремя*, однако, в специфических приложениях применяются другие параметры. Например, геофизики могли бы собрать измерения плотности скалы на равноудаленных *расстояниях*

друг от друга, вдоль поверхности земли. В общем случае, подразумевая параметры, *мы* маркируем горизонтальную ось просто: **номер отсчета**. Для непрерывного сигнала, должно быть использовано другое обозначение, такое как: *время*, *расстояние*, *x* и т.д.

В общем, два параметра, которые формируют сигнал, не взаимозаменяемы. Говорят, что параметр по оси *y* (зависимая переменная) является **функцией** параметра по оси *x* (независимая переменная). Другими словами, независимая переменная описывает, как или когда снимается каждый отсчет, в то время как зависимая переменная является фактическим измерением. Задаваясь определенным значением по оси *x*, мы всегда можем найти соответствующее значение по оси *y*, но, как правило, не наоборот.

Обратите особое внимание на слово *область*, очень широко используемый термин в ЦОС. Например, сигнал, который использует время как независимую переменную (т.е. параметр по горизонтальной оси), как считают, находится во **временной области**. Другие обычные сигналы в ЦОС в качестве независимой переменной используют частоту, что выражается термином **частотная область**. Аналогично, сигналы, которые используют расстояние как независимый параметр, как считают, находятся в **пространственной области**. Это очень просто, тип параметра на горизонтальной оси и *есть* область сигнала. А что если ось *x* отмечена чем-то очень обобщенным, вроде *номера отсчета*? Авторы обычно обращаются с такими сигналами так, как если бы они находились во *временной* области. Это потому, что наиболее обычный способ получения сигналов - осуществление выборки через равные промежутки времени и здесь нет ничего очень специфического, чтобы давать ему свое название.

Хотя сигналы на рис. 2.1 дискретные, они показаны на этом рисунке как непрерывные линии. Это в связи с тем, что различимых отсчетов слишком много, чтобы быть показанными в виде индивидуальных отметок. Индивидуальные отметки обычно показываются на графиках, которые изображают более короткие сигналы, скажем менее чем 100 отсчетов. В зависимости от того, как хочет автор, чтобы Вы увидели данные, непрерывные линии могут изображаться или не изображаться. Например, непрерывная линия может предполагать то, что происходит *между* отсчетами, или просто преследует цель помочь глазу читателя следовать за тенденцией в зашумленных данных. Для определения работаете ли Вы с дискретным или с непрерывным сигналом, главное - это проверять маркировку горизонтальной оси. Не полагайтесь на способности иллюстратора рисовать точки.

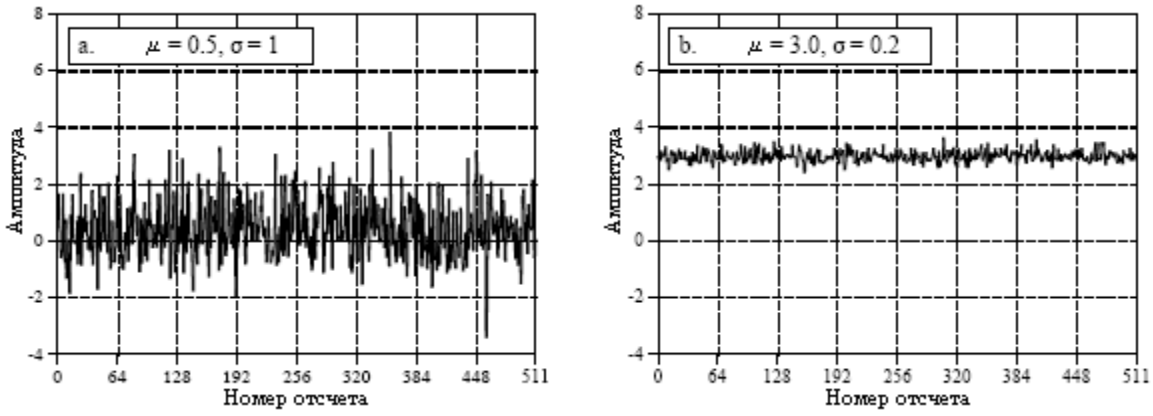


Рис.2.1 Примеры двух оцифрованных сигналов с различным средним значением и стандартным отклонением

Для представления общего количества отсчетов в сигнале в ЦОС широко используется переменная ***N***. Например, для сигнала на рис 2.1 *N* =512. Для поддержания порядка в данных каждому отсчету присваивается **номер отсчета** или **индекс**. Ими являются числа находящиеся вдоль горизонтальной оси. Для указания номера отсчета

обычно используются два вида обозначений. При первом виде обозначений индекс отсчета изменяется от 1 до N (т.е. от 1 до 512). При втором виде обозначений индекс отсчета изменяется от 0 до N-1 (т.е. от 0 до 511). Математики часто используют первый способ (от 1 до N), в то время как те, кто работает в ЦОС, обычно используют второй (от 0 до N-1). В этой книге мы будем использовать второй вид обозначения. Не относитесь к этому как к незначительной проблеме. Время от времени это *будет* смущать Вас в течение всей вашей карьеры. Следите за этим!

Среднее значение и стандартное отклонение

Среднее значение обозначается как μ (греческая прописная буква *мю*), это жаргон статистики для обозначения средней величины сигнала. Оно находится так же, как Вы бы и ожидали: сложить все отсчеты вместе и разделить на N. В математической форме это выглядит следующим образом:

$$\mu = \frac{1}{N}\sum_{i=0}^{i=N-1} x_i \ . \qquad (2.1)$$

Выражаясь словами, суммируем значения сигнала x_i, позволяя индексу i изменяться от 0 до N-1. Затем заканчиваем вычисление, деля сумму на N. Это идентично уравнению: $\mu = (x_0 + x_1 + x_2 + ... + x_{N-1})/N$. Если Вы еще не осведомлены о том, как используется Σ (греческая заглавная буква *сигма*) для обозначения *суммирования*, тщательно изучите это уравнение и сравните его с компьютерной программой приведенной в таблице 2.1. ЦОС изобилует суммированиями такого типа, и Вам необходимо полностью понять это обозначение.

В электронике среднее значение обычно называют постоянной составляющей. Аналогично переменная составляющая относится к тому, как сигнал колеблется вокруг величины среднего значения. Если сигнал представляет собой простую повторяющуюся волну, такую как синусоидальная или прямоугольная волна, его отклонения могут быть описаны изменениями его амплитуды от максимума до максимума. К сожалению, большинство получаемых сигналов имеют произвольный характер такой, как сигналы на рис.2.1, и не демонстрируют значения, хорошо определяемые от максимума к максимуму. В таких случаях должен использоваться более обобщенный метод, тут требуется **стандартное отклонение**, обозначаемое как σ (греческая прописная буква *сигма*).

Здесь следует начать с выражения $|x_i - \mu|$, которое показывает *отклонение* (величину отличия) i-го отсчета от среднего значения. Среднее отклонение сигнала находится суммированием отклонений всех индивидуальных отсчетов, а затем делением результата на количество отсчетов N. Заметьте, что до суммирования мы берем абсолютную величину каждого отклонения, в противном случае положительные и отрицательные составляющие будут усредняться к нулю. Среднее отклонение дает число, показывающее типичное расстояние отсчетов от среднего значения. Несмотря на удобство и простоту вычисления, среднее отклонение почти никогда не используется в статистике. Это связано с тем, что оно плохо отражает физику работы сигнала. В большинстве случаев важный параметр это не *отклонение* от среднего значения, а *мощность*, представляемая отклонением от среднего значения. Когда в электронной схеме происходит объединение сигналов случайных шумов, результирующий шум равен суммарной мощности индивидуальных сигналов, а не их суммарной амплитуде.

Стандартное отклонение подобно среднему отклонению за исключением того, что вместо усреднения амплитуды, проводится усреднение мощности. Это достигается возведением каждого отклонения в квадрат перед тем, как определить среднее

(вспомните, *мощность* пропорциональна *напряжению*^{2}). Для компенсации первоначального возведения в квадрат, в конце извлекается квадратный корень. В форме уравнения стандартное отклонение вычисляется как:

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{i=N-1}(x_i - \mu)^2 . \qquad (2.2)$$

В альтернативном обозначении:

$$\sigma = \sqrt{(x_0-\mu)^2 + (x_1-\mu)^2 + \ldots + (x_{N-1}-\mu)^2 /(N-1)} .$$

Обратите внимание, что среднее получается делением на *N*-1 вместо *N*. Это тонкая особенность уравнения, которая будет обсуждена в следующем разделе. Термин σ^2 часто встречается в статистике, и ему присвоено название **выборочная дисперсия**. Стандартное отклонение - это мера того, насколько далеко сигнал флюктуирует от среднего значения. Выборочная дисперсия представляет мощность этих флюктуаций. Другой, часто используемый в электронике термин, с которым Вы должны легко обращаться, это **среднеквадратичная** величина. По определению стандартное отклонение измеряет только переменную составляющую сигнала, в то время как среднеквадратичная величина измеряет совместно как переменную, так и постоянную составляющую сигнала. Если постоянная составляющая в сигнале отсутствует, то его среднеквадратичная величина идентична его стандартному отклонению. Рис. 2.2 показывает соотношения между стандартным отклонением и значением от максимума до минимума для нескольких типичных форм волн.

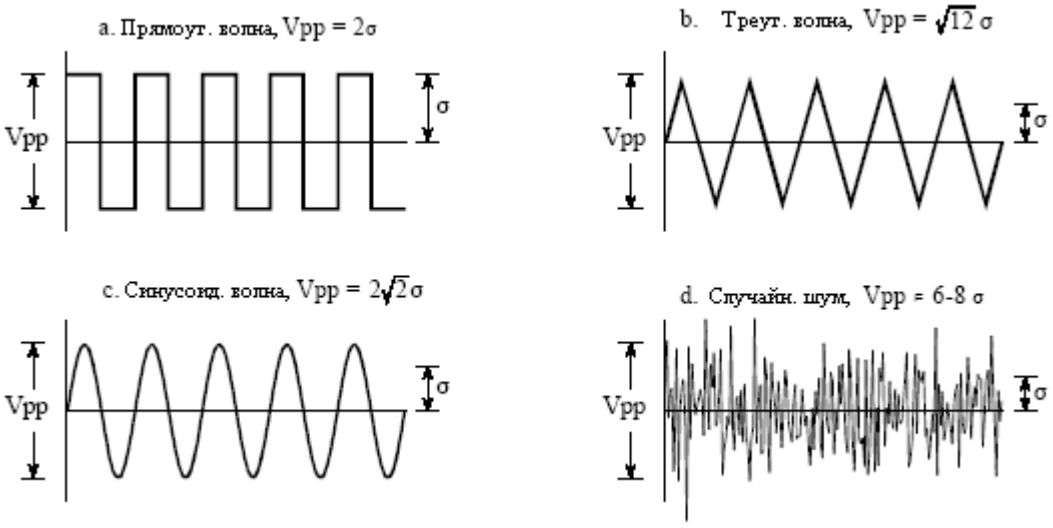


Рис. 2.2 Соотношение между стандартным отклонением и значением от максимума до минимума для нескольких типичных форм волн

В таблице 2.1 приведен листинг компьютерной программы для вычисления среднего значения и стандартного отклонения в соответствии с уравнениями (2.1) и (2.2). Программы в этой книге предназначены для выражения *алгоритмов* наиболее простым способом, все другие факторы трактуются как второстепенные. Хорошая техника программирования пренебрегается в том случае, если это делает программную логику более ясной. Например: используется упрощенная версия алгоритмического языка BASIC,

включены номера строк, для цикла FOR-NEXT допускаются только управляемые структуры, отсутствуют операции ввода/вывода и т.д. Воспринимайте эти программы как альтернативный способ понимания используемых в ЦОС, уравнений. Если Вы не сможете уловить одно, то может быть Вам поможет другое. В BASIC, символ % в конце имени переменной показывает, что это целое. Все другие переменные с плавающей запятой. Эти типы переменных детально обсуждаются в Главе 4.

Таблица 2.1

```
100 CALCULATION OF THE MEAN AND STANDARD DEVIATION
110 '
120 DIM X[511]                    'The signal is held in X[0] to X[511]
130 N% = 512                      'N% is the number of points in the signal
140 '
150 GOSUB XXXX                    'Mythical subroutine that loads the signal into X[ ]
160 '
170 MEAN = 0                      'Find the mean via Eq. 2.1
180 FOR I% = 0 TO N%-1
190 MEAN = MEAN + X[I%]
200 NEXT I%
210 MEAN = MEAN/N%
220 '
230 VARIANCE = 0                  'Find the standard deviation via Eq. 2.2
240 FOR I% = 0 TO N%-1
250 VARIANCE = VARIANCE + ( X[I%] - MEAN )^2
260 NEXT I%
270 VARIANCE = VARIANCE/(N%-1)
280 SD = SQR(VARIANCE)
290 '
300 PRINT MEAN SD                 'Print the calculated mean and standard deviation
310 '
320 END
```

Представленный метод вычисления среднего значения и стандартного отклонения адекватен для многих приложений, однако имеет два ограничения. Первое, если среднее значение намного больше, чем стандартное отклонение, то в уравнении (2.2) будет возводиться в степень разница двух очень близких по величине чисел. Это может привести к чрезмерной ошибке округления в вычислениях, более детально эта тема обсуждена в Главе 4. Второе, когда новые отсчеты получены и добавлены к сигналу, очень часто желательно пересчитать среднее значение и стандартное отклонение. Мы будем называть такой тип вычислений **бегущей статистикой**. При использовании в бегущей статистике метода уравнений (2.1) и (2.2) требуется, чтобы *все* отсчеты были включены в каждое новое вычисление. Это очень неэффективное использование вычислительной мощности и памяти.

Решение этой проблемы может быть найдено манипулированием уравнениями (2.1) и (2.2) таким образом, чтобы получить иное уравнение для вычисления стандартного отклонения:

$$\sigma^2 = \frac{1}{N-1}\left[\sum_{i=0}^{N-1} x_i^2 - \frac{1}{N}\left(\sum_{i=0}^{N-1} x_i\right)^2\right], \qquad (2.3)$$

или используя более простые обозначения,

$$\sigma^2 = \frac{1}{N-1}\left[сумма \quad квадратов - \frac{сумма^2}{N} \right].$$

По мере продвижения по сигналу сохраняются бегущие дубликаты трех параметров: (1) количество уже обработанных отсчетов, (2) сумма этих отсчетов и (3) сумма квадратов отсчетов (то есть, возведите в квадрат значение каждого отсчета и сложите результат с аккумулированным значением). Используя только текущие значения трех параметров, среднее значение и стандартное отклонение могут быть эффективно вычислены после любого обработанного отсчета. Таблица 2.2 представляет программу, которая по мере того как во внимание берется каждый новый отсчет, сообщает в такой манере, о среднем значении и стандартном отклонении. Такой метод используется для нахождения статистики последовательности чисел на ручных калькуляторах. Каждый раз, как только Вы вводите число и нажимаете клавишу ∑ (суммирования), обновляются три параметра. Тогда среднее значение и стандартное отклонение могут быть найдены в любой желаемый момент без необходимости пересчета всей последовательности.

Таблица 2.2

```
100 'MEAN AND STANDARD DEVIATION USING RUNNING STATISTICS
110 '
120 DIM X[511]                    'The signal is held in X[0] to X[511]
130 '
140 GOSUB XXXX                    'Mythical subroutine that loads the signal into X[ ]
150 '
160 N% = 0                        'Zero the three running parameters
170 SUM = 0
180 SUMSQUARES = 0
190 '
200 FOR I% = 0 TO 511             'Loop through each sample in the signal
210 '
220 N% = N%+1                     'Update the three parameters
230 SUM = SUM + X[I%]
240 SUMSQUARES = SUMSQUARES + X[I%]^2
250 '
260 MEAN = SUM/N%                 'Calculate mean and standard deviation via Eq. 2.3
270 IF N% = 1 THEN SD = 0: GOTO 300
280 SD = SQR( (SUMSQUARES - SUM^2/N%) / (N%-1) )
290 '
300 PRINT MEAN SD                 'Print the running mean and standard deviation
310 '
320 NEXT I%
330 '
340 END
```

Прежде чем закончить дискуссию о среднем значении и стандартном отклонении, необходимо упомянуть о двух других терминах. В некоторых случаях *среднее значение* описывает то, что было измерено, в то время как *стандартное отклонение* представляет шум и другие помехи. В этих случаях стандартное отклонение важно не само по себе, а

только в *сравнении* со средним значением. Это придает значимость термину **отношение сигнал-шум**, который равен среднему значению, деленному на стандартное отклонение. Другой также используемый термин – **коэффициент вариации**. Он определяется как стандартное отклонение, деленное на среднее значение, умноженное на 100 процентов. Например, сигнал (или другая группа измеренных значений) с коэффициентом вариации 2% имеет отношение сигнал-шум 50. Более хорошие данные означают более *высокое* значение отношения сигнал-шум и более *низкое* значение коэффициента вариации.

Сигнал против процесса лежащего в его основе

Статистика это наука об интерпретации *численных данных,* таких как полученные сигналы. Для сравнения, **теория вероятности** используется в ЦОС для понимания *процессов*, генерирующих сигналы. Различие между **полученным сигналом** и **процессом**, **лежащим в его основе**, хотя они и тесно связаны, является ключом к большому количеству методов ЦОС.

Например, представьте себе создание сигнала, содержащего 1000 точек, тысячекратным подбрасыванием монеты. Если монета падает орлом, соответствующий отсчет принимает значение единицы. Решка и соответствующий отсчет устанавливается в ноль. *Процесс*, создающий этот сигнал, имеет среднее значение *0,5*, заданное относительной вероятностью каждого возможного исхода: 50% орел, 50% решка. Однако вряд ли фактический 1000 точечный сигнал будет иметь среднее значение точно *0,5*. Каждый раз, когда генерируется сигнал, случайные события будут делать число единиц и нулей слегка разнящимися. *Вероятности* основного процесса постоянны, но *статистика* полученного сигнала при повторении эксперимента всякий раз меняется. Эта произвольная нерегулярность, обнаруживаемая в фактических данных, имеет такие названия как: **статистическая вариация**, **статистическая флюктуация** и **статистический шум**.

Обсудим здесь небольшую дилемму. Как Вы определяете, обращается ли автор к статистике фактического сигнала или к вероятности основного процесса, создающего сигнал, когда Вы видите термины “среднее значение” и “стандартное отклонение”? К сожалению, контекст является единственным способом, которым Вы можете сообщить об этом (для случайной величины вместо среднего значения используют термин математическое ожидание, а при описании стандартного отклонения часто пользуются таким приемом: детерминированная величина – среднеквадратичная величина, а стохастическая – среднеквадратическая – прим. перев.). Дело не обстоит так для всех терминов статистики и теории вероятности. Например, *гистограмма* и *функция массы вероятности* (обсуждаемые в следующем разделе) соответствуют концепции разных названий.

Вернемся к уравнению 2.2, вычисляющему стандартное отклонение. Как отмечалось ранее, при вычислении среднего значения квадрата отклонения это уравнение делится на N-1, а не просто на N. Для того чтобы понять, почему это так, представьте себе, что Вы хотите найти среднее значение и стандартное отклонение некоторого процесса, генерирующего сигнал. Двигаясь к концу, Вы получаете от процесса сигнал из N отсчетов и вычисляете среднее значение сигнала в соответствии с уравнением 2.1. Затем Вы можете использовать результат, как оценку среднего значения *основного процесса*, однако, Вы знаете, что из-за статистического шума там будет содержаться ошибка. В частности, для случайных сигналов типичная ошибка между средним значением N точек и основным процессом определяется как:

$$Типичная \quad ошибка = \frac{\sigma}{N^{1/2}} \qquad (2.4)$$

Если величина *N* мала, статистический шум в вычисленном среднем значении будет очень большим. Другими словами, у Вас нет доступа к достаточно большому количеству данных, чтобы характеризовать процесс должным образом. Чем больше значение *N*, тем меньше становится ожидаемая ошибка. Верстовой столб теории вероятностей, могучий *закон больших чисел*, гарантирует, что ошибка будет стремиться к нулю по мере приближения *N* к бесконечности.

Следующим шагом попытаемся вычислить стандартное отклонение полученного сигнала и использовать его в качестве оценки стандартного отклонения основного процесса. Здесь кроется проблема. Прежде чем вычислять стандартное отклонение в соответствии с уравнением (2.2), Вы уже должны знать среднее значение, μ. Однако, Вы не знаете среднего значения основного процесса, а знаете только среднее значение *N* точечного сигнала, содержащего *ошибку* из-за присутствия *статистического шума*. Эта ошибка приводит к уменьшению вычисленного значения стандартного отклонения. И для компенсации, *N* заменяется *N*-1. Если *N* велико, отличие не имеет значения. Если *N* мало, эта замена обеспечивает более точную оценку стандартного отклонения основного процесса. Другими словами уравнение (2.2) - это *оценка стандартного отклонения* основного процесса. Если мы в уравнении будем делить на *N*, то это даст стандартное отклонение полученного сигнала.

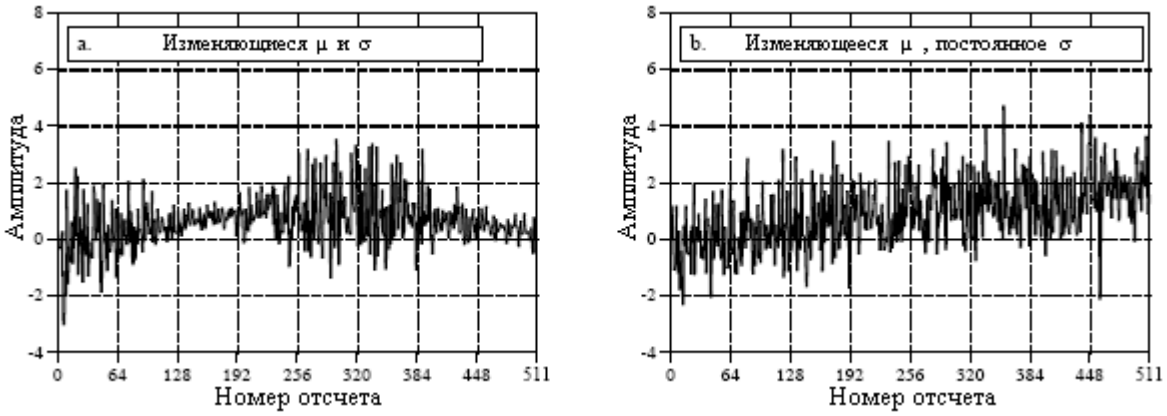


Рис. 2.3 Примеры сигналов созданных нестационарными процессами

В качестве иллюстрации этих идей рассмотрим сигналы на рис. 2.3 и зададим себе вопрос: являются ли изменения в этих сигналах результатом статистического шума или это результат изменений основного процесса? Вероятно, не трудно убедиться, что эти изменения являются слишком большими для случайного события и должны относиться к основному процессу. Процессы, которые изменяют свои характеристики в такой манере, называются **нестационарными**. Для сравнения, представленные ранее сигналы на рис. 2.1 были сгенерированы стационарным процессом, и изменения полностью являются результатом статистического шума. Рис. 2.3b. иллюстрирует общую проблему нестационарных сигналов: медленные изменения *среднего значения* мешают вычислению *стандартного отклонения*. Стандартное отклонение сигнала в этом примере перекрывает небольшой интервал величиной *единица*. Однако, стандартное отклонение полного сигнала равно *1,16*. Эта ошибка может быть почти устранена путем разбиения сигнала на короткие участки и индивидуального вычисления статистики для каждого участка. Для получения одного значения, при необходимости, стандартные отклонения для каждого из участков могут быть усреднены.

Гистограмма, функция массы вероятности и функция плотности вероятности

Предположим, что мы подключили 8 битовый аналого-цифровой преобразователь к компьютеру и получили 256000 отсчетов некоторого сигнала. В качестве примера, на рис. 2.4a показано 128 отсчетов, которые могут быть частью этого набора данных. Каждый отсчет может принимать одно из 256 значений от 0 до 255. **Гистограмма** показывает *число отсчетов* сигнала, принимающих каждое из этих *возможных значений*. Рисунок 2.4b показывает гистограмму для 128 отсчетов, взятых с рис. 2.4a. Для примера, здесь есть два отсчета, значение которых 110, семь отсчетов со значением 131, отсутствуют отсчеты со значением 170 и т.д. Мы будем представлять гистограмму символом H_i, где i - индекс, изменяющийся от 0 до M-1, а M - число возможных значений, которое может принимать каждый отсчет. Например, H_{50} – число отсчетов имеющих значение 50. Рисунок 2.4c показывает гистограмму сигнала для полного набора данных, всех 256K точек. Как можно увидеть, большее количество отсчетов приводит к более гладкому внешнему виду. Так же, как и в случае среднего значения, статистический шум (шероховатость) гистограммы обратно пропорционален квадратному корню числа используемых отсчетов.

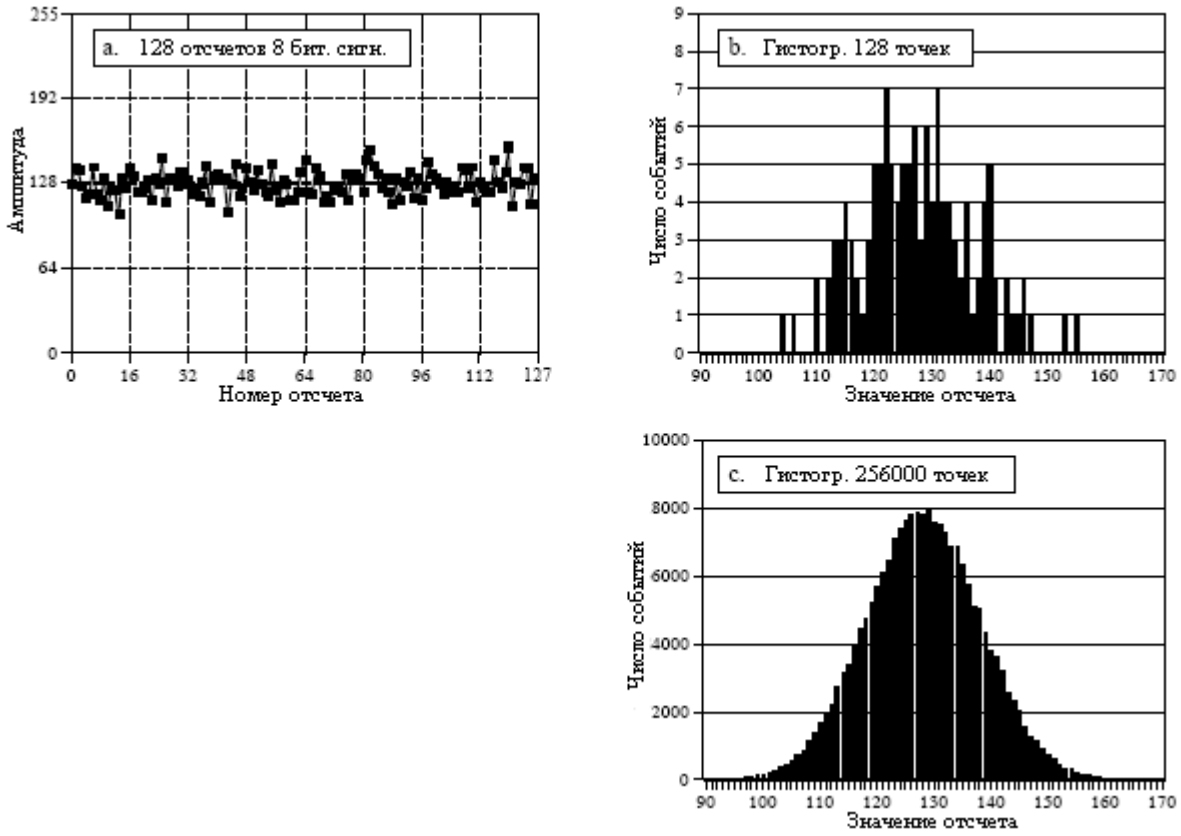


Рис. 2.4 Примеры гистограмм

В соответствии со способом задания определения гистограммы, сумма всех значений в гистограмме должна быть равна числу точек в сигнале:

$$N = \sum_{i=0}^{M-1} H_i \,. \qquad (2.5)$$

Гистограмма может использоваться для эффективного вычисления среднего значения и стандартного отклонения очень больших наборов данных. Это особенно важно для *изображений*, которые могут содержать миллионы отсчетов. Гистограмма группирует отсчеты имеющие одинаковые значения вместе. Это позволяет вычислять статистику, обрабатывая несколько групп вместо огромного числа индивидуальных отсчетов. Используя такой подход, среднее значение и стандартное отклонение вычисляются из гистограммы по уравнениям:

$$\mu = \frac{1}{N} \sum_{i=0}^{M-1} iH_i \tag{2.6}$$

$$\sigma^2 = \frac{1}{N-1} \sum_{i=0}^{M-1} (i-\mu^2)H_i \tag{2.7}$$

Таблица 2.3 содержит программу вычисления гистограммы, среднего значения и стандартного отклонения в соответствии с этими уравнениями. Вычисление гистограммы происходит очень быстро, поскольку оно требует только индексирования и инкрементирования. Для сравнения, вычисление среднего значения и стандартного отклонения требует более длительных операций сложения и умножения. Стратегия этого

Таблица 2.3

```
100 'CALCULATION OF THE HISTOGRAM, MEAN, AND STANDARD DEVIATION
110 '
120 DIM X%[25000]                 'X%[0] to X%[25000] holds the signal being processed
130 DIM H%[255]                   'H%[0] to H%[255] holds the histogram
140 N% = 25001                    'Set the number of points in the signal
150 '
160 FOR I% = 0 TO 255             'Zero the histogram, so it can be used as an accumulator
170 H%[I%] = 0
180 NEXT I%
190 '
200 GOSUB XXXX                    'Mythical subroutine that loads the signal into X%[ ]
210 '
220 FOR I% = 0 TO 25000           'Calculate the histogram for 25001 points
230 H%[ X%[I%] ] = H%[ X%[I%] ] + 1
240 NEXT I%
250 '
260 MEAN = 0                      'Calculate the mean via Eq. 2-6
270 FOR I% = 0 TO 255
280 MEAN = MEAN + I% * H%[I%]
290 NEXT I%
300 MEAN = MEAN / N%
310 '
320 VARIANCE = 0                  'Calculate the standard deviation via Eq. 2-7
330 FOR I% = 0 TO 255
340 VARIANCE = VARIANCE + H%[I%] * (I%-MEAN)^2
350 NEXT I%
360 VARIANCE = VARIANCE / (N%-1)
```

```
370 SD = SQR(VARIANCE)
380 '
390 PRINT MEAN SD                'Print the calculated mean and standard deviation.
400 '
410 END
```

алгоритма применять такие медленные операции не ко многим отсчетам в сигнале, а только к нескольким числам в гистограмме. Это делает алгоритм значительно быстрее, чем ранее описанные методы. Для очень длительных сигналов думается раз в десять, при условии выполнения вычислений на компьютерах общего назначения.

Понимание того, что полученный сигнал является зашумленной версией основного процесса, очень важно; настолько важно, что некоторым из понятий даются отличные друг от друга названия. *Гистограмма* это то, что формируется из полученного сигнала. Соответствующая кривая для основного процесса называется **функцией массы вероятности**. Гистограмма всегда вычисляется при использовании конечного числа отсчетов, в то время как функция массы вероятности это то, что *может* быть получено при использовании бесконечного числа отсчетов. Функция массы вероятности может быть оценена (выведена) из гистограммы или может быть получена с помощью нескольких математических методов таких, как в примере с подбрасыванием монеты.

На рис. 2.5 показан пример функции массы вероятности и одна из возможных гистограмм, которая может ассоциироваться с ней. Ключ к пониманию данного понятия лежит в единицах измерения вертикальной оси. Как объяснялось ранее, вертикальная ось гистограммы это *число раз*, которое конкретное значение встречается в сигнале. Вертикальная ось функции массы вероятности содержит такую же информацию, кроме выраженного через *дробь основания*. Другими словами, чтобы аппроксимировать функцию массы вероятности, каждое значение в гистограмме делится на общее число отсчетов. Это означает, что каждое значение в функции массы вероятности должно лежать между нулем и единицей, и что сумма всех значений в функции массы вероятности будет равна единице.

Функция массы вероятности важна, поскольку она описывает вероятность, с которой будет генерироваться конкретное значение. Например, представьте сигнал, такой как ранее показанный на рис. 2.4a, с функцией массы вероятности, как на рис. 2.5b. Какова вероятность того, что отсчет, взятый из этого сигнала, будет иметь значение равное 120? Рисунок 2.5b дает ответ 0,03 или около одного шанса из 34. Какова вероятность того, что случайно выбранный отсчет будет иметь значение большее 150? Суммируя значения в функции массы вероятности для: 151, 152, 153, …, 255, получим ответ 0,0122 или около 1 шанса из 82. Таким образом, сигнал будет иметь значение, превышающее *150*, как ожидалось бы, в среднем через каждые 82 точки. Какова вероятность того, что какой-либо отсчет попадет в промежуток между 0 и 255? Суммируя все значения в функции массы вероятности, получим вероятность *1,00*, т.е. это произойдет обязательно.

Гистограмма и функция массы вероятности могут быть использованы только с дискретными данными, такими как оцифрованный сигнал, проживающий в компьютере. Подобные понятия применимы и к непрерывным сигналам, таким как напряжение, возникающее в аналоговой электронике. **Функция плотности вероятности** (фпв) для непрерывных сигналов то же, что и функция массы вероятности для дискретных сигналов. Например, представьте аналоговый сигнал, проходящий через аналого-цифровой преобразователь и преобразующийся в оцифрованный сигнал, такой как на рис. 2.4a. Для простоты предположим, что напряжение между 0 и 255 милливольтами при оцифровке принимает значения чисел от 0 до 255. Функция массы вероятности этого цифрового сигнала на рис.2.5b отмечена *маркерами*. Аналогично на рис. 2.4c *непрерывной линией* показана функция плотности вероятности, отмечающая непрерывный диапазон значений,

которые может принимать сигнал, таких же значений, какие может принимать напряжение в электронной цепи.

Единицей измерения вдоль вертикальной оси функции плотности вероятности является плотность вероятности, а не просто вероятность. Например, функция плотности вероятности в 0,03 при 120,5 *не* означает, что напряжение в 120,5 милливольта будет встречаться 3% времени. В действительности, вероятность непрерывного сигнала величиной точно 120,5 милливольт элементарно мала. Это из-за того, что имеется бесконечное число возможных значений, между которыми сигнал должен разделить свое время: 120,49997, 120,49998, 120,49999 и т.д. Шанс того, что сигнал будет точно 120,50000… на самом деле, конечно, очень мал!

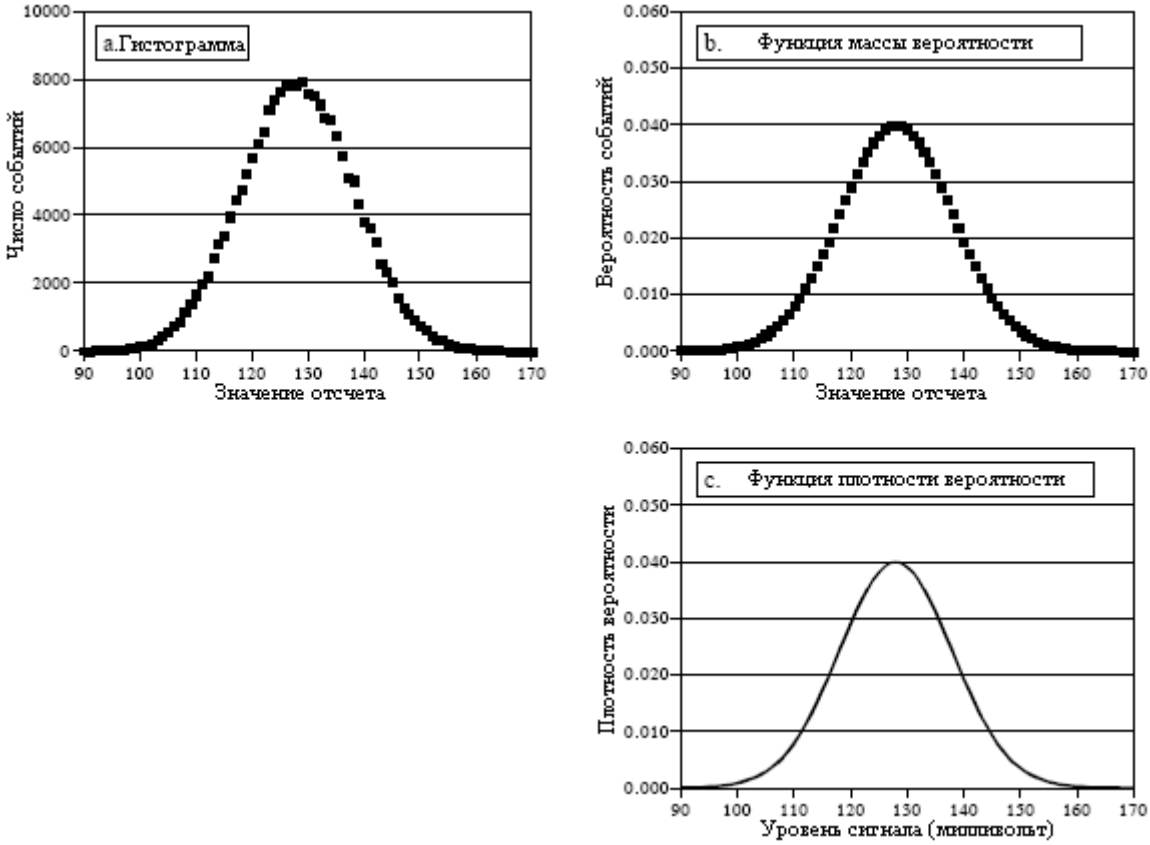


Рис. 2.5 Соотношения между гистограммой, функцией массы вероятности и функцией плотности вероятности

Для вычисления *вероятности*, *плотность вероятности* умножается на *диапазон* значений. Например, вероятность того, что сигнал в любой заданный момент будет находиться между значениями 120 и 121 будет: (121 - 120) × 0,03 = 0,03. Вероятность того, что сигнал будет между 120,4 и 120,5 будет: (120,5 – 120,4) × 0,03 = 0,003 и т.д. Если функция плотности вероятности непостоянна в пределах интересующего нас диапазона, умножение заменяется интегрированием в пределах этого диапазона. Другими словами, площадью под функцией плотности вероятности ограниченной указанными значениями. Поскольку значение сигнала всегда должно принимать *какое-то значение*, вся площадь под кривой функции плотности вероятности, интеграл от -∞ до +∞, всегда будет равна единице. Это так же, как сумма всех значений функции массы вероятности равна единице, и так же, как сумма всех значений гистограммы равна *N*.

Гистограмма, функция массы вероятности и функция плотности вероятности являются очень близкими понятиями. Математики всегда используют их строго и непосредственно, однако частенько, Вы будете обнаруживать, что многие ученые и

инженеры используют их друг вместо друга (и, следовательно, некорректно). На рис. 2.6 показаны три *непрерывных* формы волны и их *функции плотности вероятности*. Если бы здесь были *дискретные сигналы*, отмеченные заменой обозначения горизонтальной оси на “номер отсчета”, использовались бы *функции массы вероятности*.

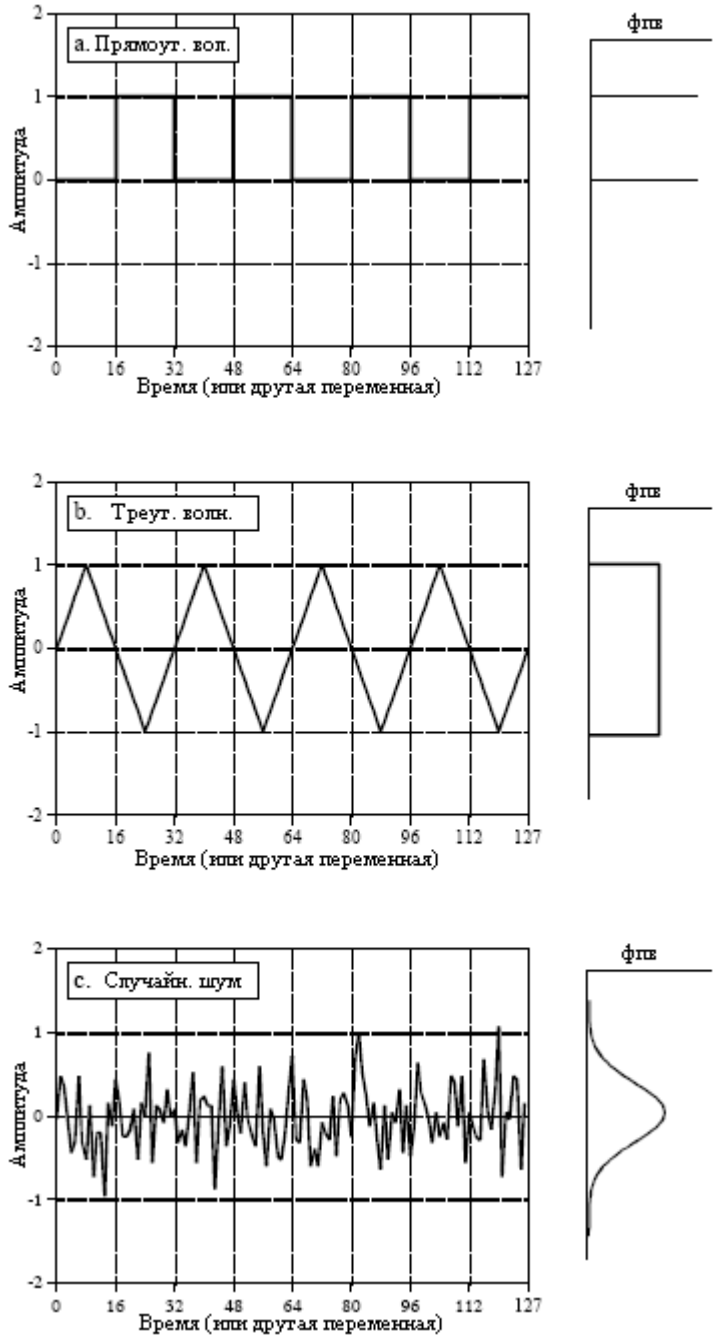


Рис. 2.6 Три типичных формы волны и их функции плотности вероятности

При вычислении гистограмм, когда число уровней, которые может принимать каждый отсчет, значительно превышает число отсчетов в сигнале, можно столкнуться с проблемой. Это также относится к сигналам, представленным в виде чисел с *плавающей запятой*, где каждый отсчет хранится как дробная величина. Например, представление в виде целого числа требует, чтобы значение отсчета было либо 3, либо 4, в то время как плавающая запятая дает миллионы возможных дробных значений *между* 3 и 4. Описанный ранее подход к вычислению гистограммы включает подсчет числа отсчетов, принимающих каждый из возможных уровней квантования. Для данных с плавающей

запятой это невозможно, поскольку существуют *миллиарды* возможных уровней, которые должны быть приняты во внимание. Даже хуже, почти все из этих возможных уровней не будут иметь ни одного отсчета соответствующего им. Например, представим сигнал, содержащий 10000 отсчетов, с одним миллиардом возможных значений для каждого отсчета. Обычная гистограмма состояла бы из одного миллиарда точек данных, все из которых, кроме 10000, имели бы нулевое значение.

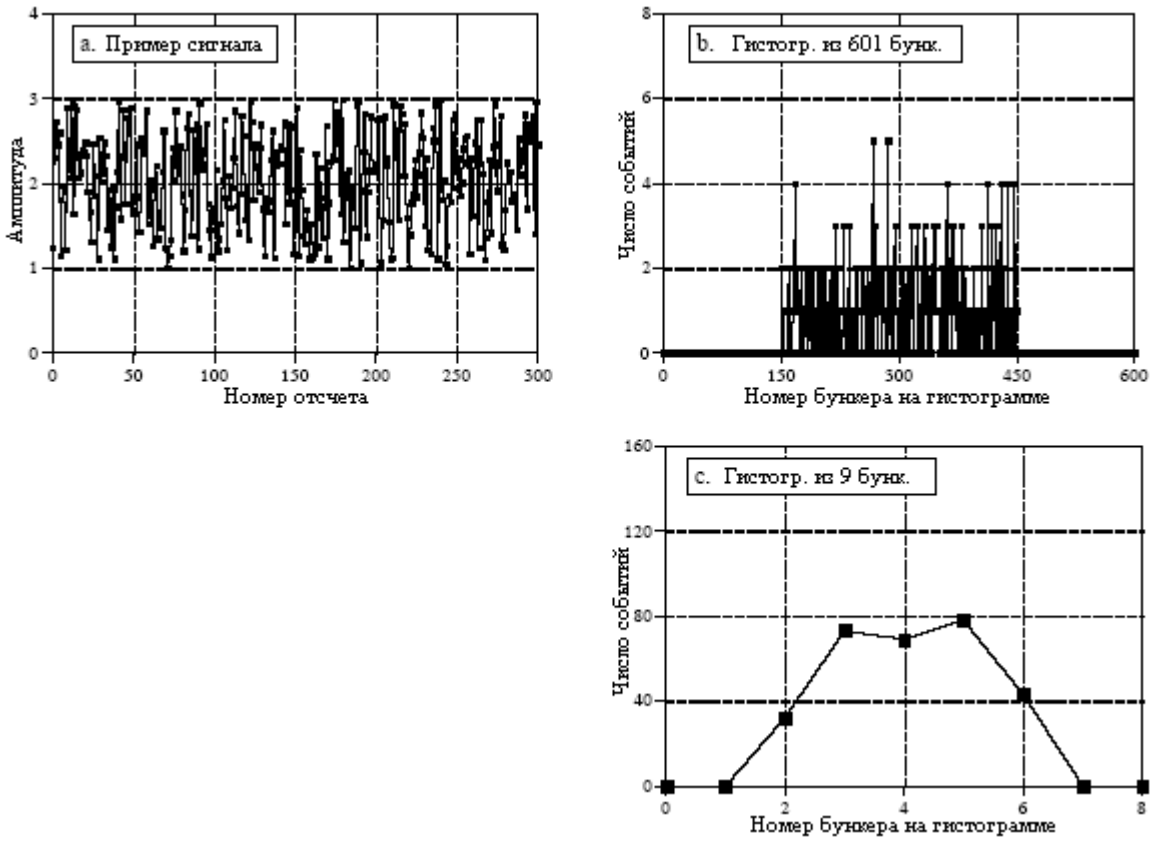


Рис. 2.7 Пример бункерованных гистограмм

Решением этой проблемы является метод носящий название **бункерование**. Оно выполняется следующим образом, произвольно выбирается длина гистограммы, так чтобы она была некоторым удобным числом, скажем 1000 точек, так называемый **бункер**. Величина каждого бункера представляет общее количество отсчетов в сигнале, которые имеют значение в пределах *некоторого диапазона*. Представьте, например, сигнал с плавающей запятой, содержащий значения между 0,0 и 10,0 и гистограмму с 1000 бункеров. Бункер 0 на гистограмме - это число отсчетов в сигнале со значением между 0 и *0,01*, бункер 1 - это число отсчетов со значением между 0,01 и 0,02 и т.д. до бункера 999, содержащего число отсчетов со значением между 9,99 и 10,0. В таблице 2.4 представлена программа для вычисления бункерованной гистограммы таким способом.

Таблица 2.4

```
100 'CALCULATION OF BINNED HISTOGRAM
110 '
120 DIM X[25000]                  'X[0] to X[25000] holds the floating point signal,
130                               'with each sample having a value between 0.0 and 10.0.
140 DIM H%[999]                   'H%[0] to H%[999] holds the binned histogram
150 '
```

```
160 FOR I% = 0 TO 999                'Zero the binned histogram for use as an accumulator
170 H%[I%] = 0
180 NEXT I%
190 '
200 GOSUB XXXX                       'Mythical subroutine that loads the signal into X%[ ]
210 '
220 FOR I% = 0 TO 25000              'Calculate the binned histogram for 25001 points
230 BINNUM% = INT( X[I%] * 100 )
240 H%[ BINNUM%] = H%[ BINNUM%] + 1
250 NEXT I%
260 '
270 END
```

Какое должно быть использовано количество бункеров? Это компромисс между двумя проблемами. Как показано на рис. 2.7, слишком большое количество бункеров затрудняет оценку *амплитуды* основной функции массы вероятности. Это происходит из-за того, что в каждый бункер попадает всего несколько отсчетов, создавая высокий уровень статистического шума. Другая крайность, очень небольшое количество бункеров, затрудняет оценку основной функции массы вероятности в *горизонтальном* направлении. Другими словами, число бункеров управляет компромиссом между разрешением вдоль оси *y* и разрешением вдоль оси *x*.

Нормальное распределение

Сигналы, формируемые случайными процессами, обычно имеют колоколообразную форму функции плотности вероятности. Это называется **нормальным распределением**, **Гауссовым распределением** или **Гауссианом**, в честь великого немецкого математика Карла Фридриха Гаусса (1777-1855). Причина, почему эта кривая встречается в природе так часто, будет кратко обсуждена в связи с *генерированием цифрового шума*. Основная форма кривой образуется из *экспоненты с отрицательным квадратом* степени:

$$y(x) = e^{-x^2} .$$

Эта сырая кривая может быть преобразована в полный Гауссиан, добавлением выверенного среднего значения μ и стандартного отклонения σ. Кроме того, уравнение должно быть нормализовано так, чтобы вся площадь, ограниченная кривой, была равна *единице*, это требование всех функций распределения вероятности. Это приводит к общей форме нормального распределения, одного из наиболее важных соотношений в статистике и теории вероятности:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \qquad (2.8)$$

Рис.2.8 показывает несколько примеров Гауссовых кривых с разными средними значениями и стандартными отклонениями. Среднее значение центрует кривую вокруг конкретного значения, в то время как стандартное отклонение регулирует ширину колоколообразной формы.

Интересной характеристикой Гауссиана является то, что его *хвосты* очень быстро приближаются к нулю, намного быстрее, чем у других обычных функций, таких как

экспоненциального радиоактивного распада или *1/x*. Например, при стандартных отклонениях от среднего значения два, четыре и шесть величина Гауссовой кривой уменьшилась приблизительно на 1/19, 1/7563, и 1/166666666 соответственно. Вот почему нормально распределенные сигналы, такие как показано на рис. 2.6c, *кажется*, имеют почти одинаковую величину от максимума до минимума. В принципе, сигналы этого типа могут испытывать отклонения неограниченной амплитуды. На практике, крутое снижение Гауссовой функции распределения вероятности диктует то, что такие крайности почти никогда не случаются. Это приводит к появлению формы волны, имеющей относительные ограничения, с полной амплитудой от максимума до минимума около 6-8 σ.

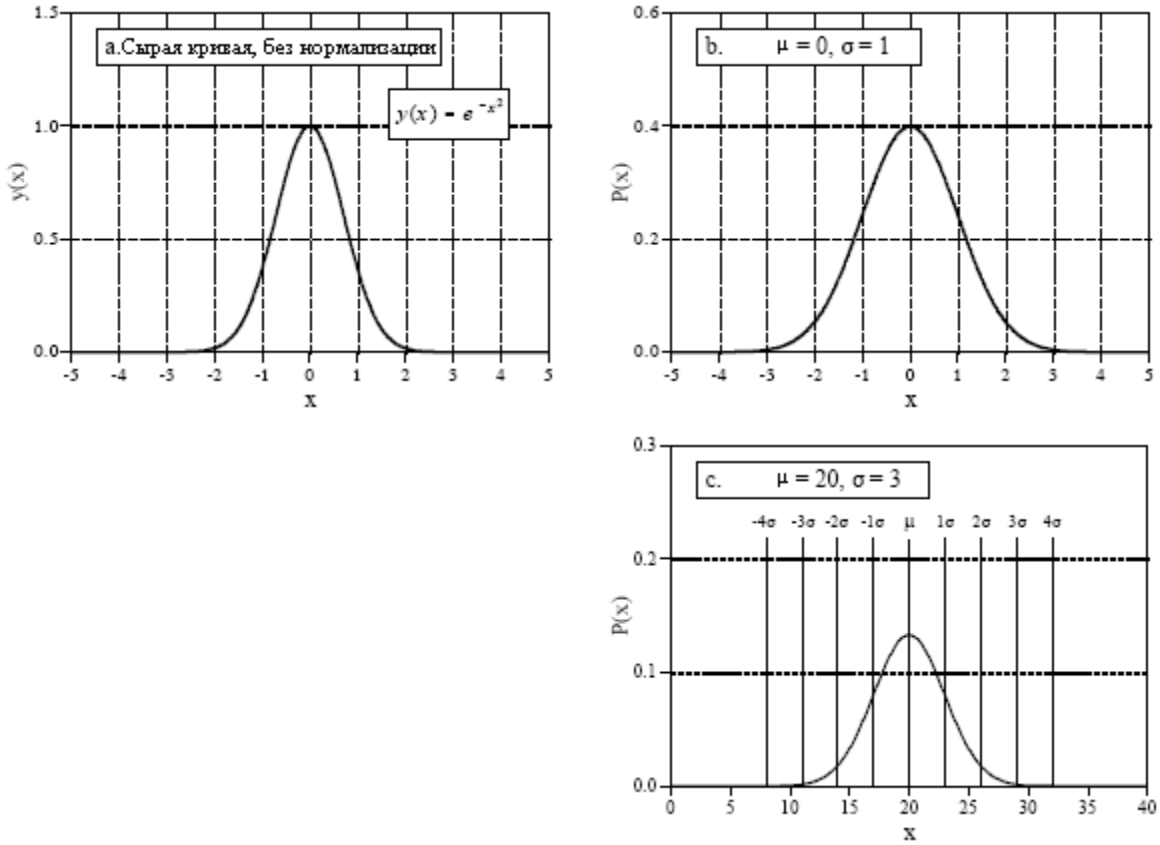
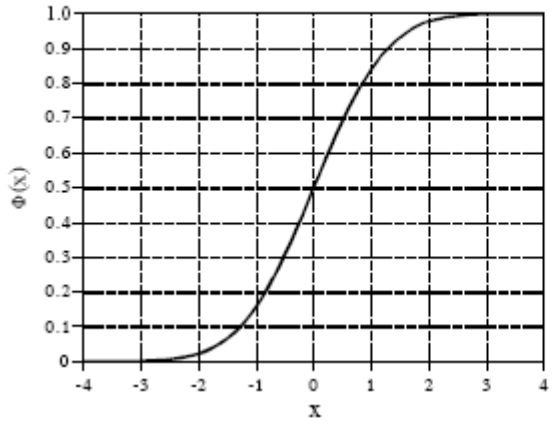


Рис. 2.8 Примеры Гауссовых кривых

Как было показано ранее, интеграл функции плотности вероятности используется для нахождения вероятности того, что сигнал будет в пределах некоторого диапазона значений. Это делает интеграл функции плотности вероятности достаточно важным, чтобы дать ему свое собственное название: **интегральная функция распределения**. Особенно неприятной проблемой с Гауссианом является то, что он не может быть проинтегрирован при помощи элементарных функций. Обойти это, можно подсчитав интеграл Гауссиана *численным интегрированием*. Это потребует очень частой дискретизации непрерывной Гауссовой кривой, скажем несколько миллионов точек между -10σ и +10σ. Затем, для моделирования *интегрирования* отсчеты в этом дискретном сигнале *суммируются*. Дискретная кривая, являющаяся результатом этого смоделированного интегрирования, сохраняется затем в таблице для использования при вычислении вероятностей.

Для нормального распределения на рис. 2.9 показана интегральная функция распределения с ее численными значениями, приведенными в таблице 2.5. Поскольку эта кривая так часто используется в теории вероятности, ей присвоен свой собственный символ: *Φ(x)* (греческая заглавная буква *фи*). Например, *Φ(-2)* имеет значение 0,0228. Это

показывает, что в любой произвольно выбранный момент времени, вероятность того, что значение сигнала будет между -∞ и двумя стандартными отклонениями ниже среднего значения, равна 2,28%. Аналогично, значение *Ф(1)* = 0,8413 означает, что есть 84,13% шансов того, что значение сигнала, в произвольно выбранное мгновение, будет между -∞



x	Φ(x)	x	Φ(x)
-3.4	.0003	0.0	.5000
-3.3	.0005	0.1	.5398
-3.2	.0007	0.2	.5793
-3.1	.0010	0.3	.6179
-3.0	.0013	0.4	.6554
-2.9	.0019	0.5	.6915
-2.8	.0026	0.6	.7257
-2.7	.0035	0.7	.7580
-2.6	.0047	0.8	.7881
-2.5	.0062	0.9	.8159
-2.4	.0082	1.0	.8413
-2.3	.0107	1.1	.8643
-2.2	.0139	1.2	.8849
-2.1	.0179	1.3	.9032
-2.0	.0228	1.4	.9192
-1.9	.0287	1.5	.9332
-1.8	.0359	1.6	.9452
-1.7	.0446	1.7	.9554
-1.6	.0548	1.8	.9641
-1.5	.0668	1.9	.9713
-1.4	.0808	2.0	.9772
-1.3	.0968	2.1	.9821
-1.2	.1151	2.2	.9861
-1.1	.1357	2.3	.9893
-1.0	.1587	2.4	.9918
-0.9	.1841	2.5	.9938
-0.8	.2119	2.6	.9953
-0.7	.2420	2.7	.9965
-0.6	.2743	2.8	.9974
-0.5	.3085	2.9	.9981
-0.4	.3446	3.0	.9987
-0.3	.3821	3.1	.9990
-0.2	.4207	3.2	.9993
-0.1	.4602	3.3	.9995
0.0	.5000	3.4	.9997

Рис. 2.9 и Таблица 2.5 Интегральная функция распределения с ее численными значениями

и одним стандартным отклонением выше среднего значения. Для вычисления вероятности того, что сигнал будет *между* двумя значениями, необходимо вычесть соответствующие числа, найденные в таблице *Ф(x)*. Например, вероятность того, что значение сигнала в некоторое произвольно выбранное время будет между двумя стандартными отклонениями ниже среднего значения и одним стандартным отклонением выше среднего значения, определяется как: *Ф(1) - Ф(-2)* = 0,8185 или 81,85%.

При использовании этого метода, отсчеты, взятые из сигнала с нормальным распределением, будут в пределах $\pm 1\sigma$ от среднего значения, приблизительно 68 % времени. Приблизительно 95 % времени они будут в пределах $\pm 2\sigma$ и в пределах $\pm 3\sigma$ приблизительно 99,75 % времени. Вероятность нахождения сигнала за пределами 10 стандартных отклонений от среднего значения настолько ничтожна, что ожидается, что это могло бы произойти в течение всего нескольких *микросекунд* за время существования вселенной, около 10 миллиардов лет!

Уравнение (2.8) может также использоваться для выражения функции массы вероятности *дискретного* сигнала с нормальным распределением. В этом случае *x* ограничивается одним из уровней квантования, которые может принимать сигнал, наподобие одного из 4096 двоичных значений, выходящих из 12 битового аналого-цифрового преобразователя. Игнорируйте член $1/\sigma\sqrt{2\pi}$, он используется только для того, чтобы сделать всю площадь под кривой функции плотности вероятности равной

единице. Вместо него Вы можете включить любой необходимый член, делающий сумму всех значений в функции массы вероятности равной *единице*. В большинстве случаев это делается генерированием кривой, не беспокоясь о нормализации, суммированием всех ненормализованных значений и затем делением всех значений на сумму.

Генерирование цифрового шума

Случайные шумы являются важной темой, как в электронике, так и в ЦОС. Например, они создают ограничение того, насколько малый сигнал может быть измерен прибором, на какое расстояние могут связываться радиосистемы и какой уровень радиации требуется для того, чтобы создать рентгеновское изображение. Обычная потребность в ЦОС – генерирование сигналов имеющих сходство с различными типами случайных шумов. Это требуется для проверки функционирования алгоритмов, которые должны *работать* в присутствии шумов.

Сердцем цифрового генерирования шума является **генератор случайных чисел**. В большинство языков программирования он входит как стандартная функция. Оператор BASIC-а: X = RND загружает переменную X новым случайным числом, каждый раз, когда исполняется эта команда. Каждое случайное число имеет значение между нулем и единицей с равной вероятностью везде между этими двумя крайними значениями. Рис. 2.10a показывает сигнал, сформированный 128 отсчетами, взятыми от генератора случайных чисел такого типа. Среднее значение основного процесса, который генерирует эти сигналы, равно *0,5*, стандартное отклонение составляет $1/\sqrt{12} = 0{,}29$ при равномерном распределении между нулем и единицей.

Алгоритмы должны проверяться с использованием того же самого вида данных, с которыми они столкнутся при действительном исполнении. Это создает необходимость генерировать цифровой шум с *Гауссовой* функцией распределения вероятности. Существует два метода для генерирования таких сигналов с использованием генератора случайных чисел. Рис. 2.10 иллюстрирует первый метод. Рисунок 2.10b показывает сигнал, полученный сложением двух случайных чисел для формирования каждого отсчета, т.е. X = RND+RND. Поскольку каждое из случайных чисел может изменяться от нуля до единицы, сумма может изменяться от нуля до двух. В этом случае среднее значение равно *единице*, а стандартное отклонение $1/\sqrt{6}$ (помните, когда независимые случайные сигналы складываются, дисперсии складываются тоже). Как показано, функция распределения вероятности изменилась от *равномерного* распределения до *треугольного* распределения. То есть сигнал находится большее количество времени вокруг значения *единицы* и меньшее время около *нуля* или *двойки*.

Рис.2.10c продвигает эту идею на шаг вперед, складывая двенадцать случайных чисел для получения каждого отсчета. Теперь среднее значение равно *шести*, а стандартное отклонение *единице*. Самое важное, что функция распределения вероятности виртуально становится *Гауссовой*. Эта процедура может использоваться для создания сигнала шума с нормальным распределением, с произвольным средним значением и стандартным отклонением. Для каждого отсчета в сигнале: (1) сложить двенадцать случайных чисел, (2) вычесть шесть, чтобы сделать среднее значение равным нулю, (3) умножить на величину желаемого стандартного отклонения и (4) прибавить желаемое среднее значение.

Математический базис для этого алгоритма заложен в **центральной предельной теореме**, одной из наиболее важных концепций в теории вероятности. В ее наиболее простой форме центральная предельная теорема утверждает, что *сумма* случайных чисел подчиняется нормальному распределению тем больше, чем больше случайных чисел складывается вместе. Центральная предельная теорема *не требует*, чтобы

индивидуальные случайные числа формировались каким-либо конкретным распределением или даже чтобы случайные числа формировались *одним и тем же* распределением. Центральная предельная теорема объясняет причины, почему сигналы с нормальным распределением так широко распространены в природе. Всякий раз, когда взаимодействует большое количество различных случайных сил, результирующая функция распределения вероятности становится *Гауссовой*.

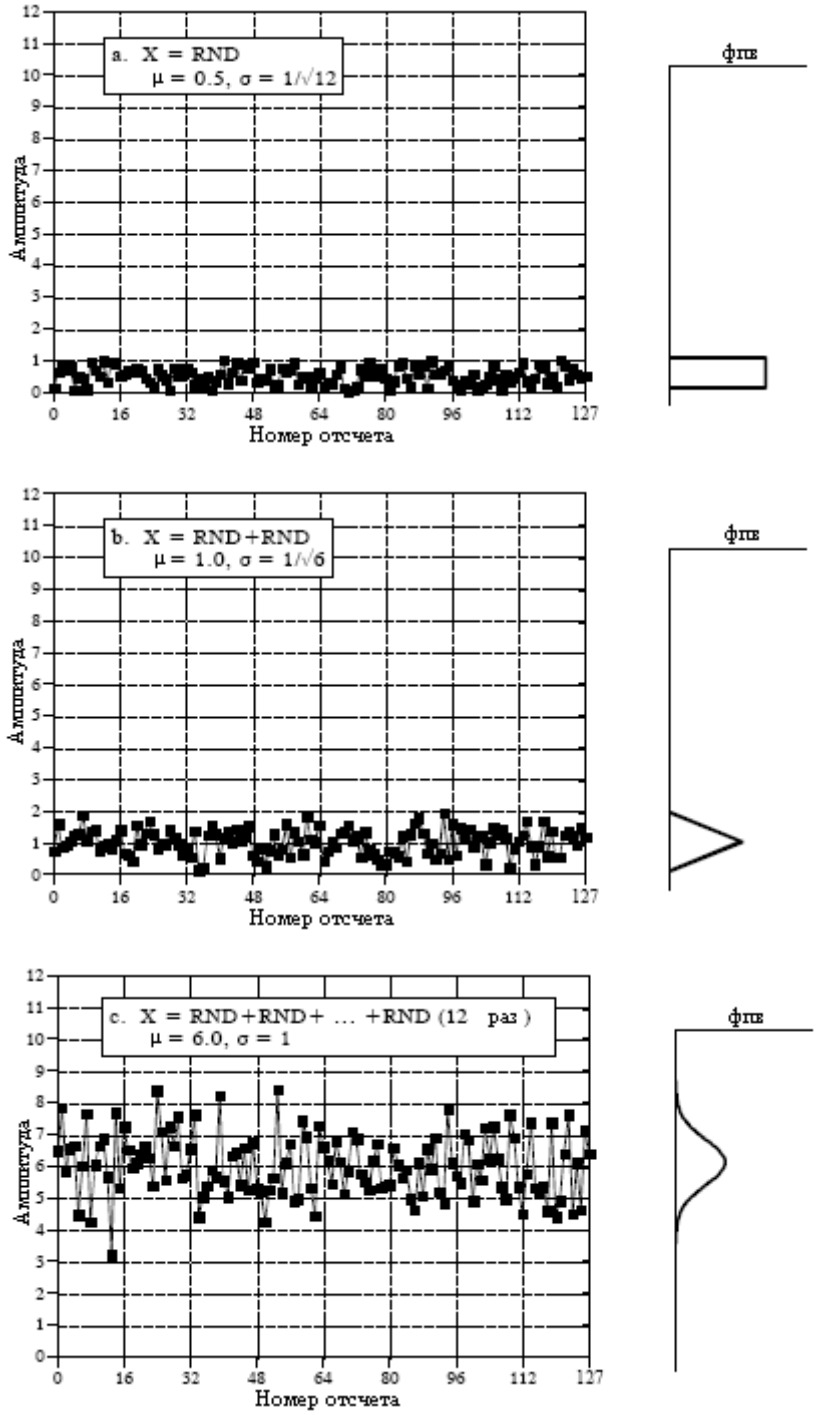


Рис. 2.10 Преобразование равномерного распределения в Гауссово

Во втором методе для генерирования случайных чисел с нормальным распределением, генератор случайных чисел вызывается дважды для получения R_1 и R_2. Случайное число X с нормальным распределением может быть найдено тогда, как:

$$X = (-2\lg R_1)^{1/2} \cos(2\pi R_2) \tag{2.9}$$

Также как и прежде, этот подход может генерировать случайные сигналы с нормальным распределением при произвольном среднем значении и стандартном отклонении. Возьмите каждое число, сгенерированное этим уравнением, умножьте его на желаемое стандартное отклонение и прибавьте желаемое среднее значение.

Работа генераторов случайных чисел начинается с **затравки**, числа находящегося между нулем и единицей. Когда происходит вызов генератора случайных чисел, затравка проходит через фиксированный алгоритм, результатом работы которого является новое число между нулем и единицей. Это новое число рассматривается как *случайное число* и хранится внутри для использования в качестве затравки в следующий раз, когда будет вызван генератор случайных чисел. Алгоритм, который преобразует затравку в новое случайное число, часто имеет следующую форму:

$$R = (aS + b) \mod c \tag{2.10}$$

Таким способом может быть вычислена непрерывная последовательность случайных чисел, началом которым, служит все та же затравка. Это позволяет многократно запускать программу и пользоваться той же самой последовательностью случайных чисел. Если Вы хотите изменить последовательность случайных чисел, большинство языков обладают средствами для **перенастраивания** генератора случайных чисел, позволяя Вам выбрать число, первоначально используемое в качестве затравки. Обычным приемом является использование в качестве затравки *времени* (показываемого системными часами компьютера), это позволяет получать новую последовательность каждый раз, когда запускается программа.

С чисто математической точки зрения числа, сгенерированные таким способом, не могут быть абсолютно случайными, поскольку каждое число полностью определяется *предыдущим* числом. Чтобы подчеркнуть эту ситуацию, часто используют термин **псевдослучайное** число. Однако здесь нет ничего, что должно было бы Вас беспокоить. Последовательности, генерируемые генераторами случайных чисел, статистически являются случайными в чрезвычайно высокой степени. Очень маловероятно, что Вы столкнетесь с ситуацией, при которой они будут неадекватны.

Точность и погрешность

Точность и погрешность это термины, которые используются для описания систем и методов, которые *измеряют*, *оценивают* или *предсказывают*. Во всех этих случаях имеется некоторый параметр, значение которого Вы желаете знать. Так называемое **истинное значение** или просто **истину**. Вы хотите, чтобы методы выдавали **измеряемую величину** по возможности как можно ближе к истинному значению. *Точность* и *погрешность* это пути описания ошибок, которые могут существовать между двумя этими величинами.

К сожалению, по нетехническому назначению, точность и погрешность используются как равнозначные. Фактически, словари определяют их с помощью взаимных ссылок друг на друга! Несмотря на это, наука и инженерное дело имеют очень конкретные определения для каждого из них. Вы должны считать обязательным для себя, правильное использование терминов и спокойно терпеть других, когда они используют их неправильно.

В качестве примера рассмотрим океанографа, измеряющего глубину воды с помощью *гидролокационной* системы. Короткие взрывы звука передаются от судна,

отражаются от океанского дна и принимаются на поверхности воды в виде эха. Звуковые волны перемещаются в воде с относительно постоянной скоростью (около 1500 метров в секунду – прим ред. перев.), позволяя находить глубину по времени, прошедшему между переданными и полученными импульсами. Как и со всеми эмпирическими измерениями, существует некоторая величина ошибки между измеренными и истинными значениями. На это конкретное измерение могло оказывать воздействие множество факторов: случайные шумы в электронной аппаратуре, волны на поверхности океана, рост растений на океанском дне, изменения температуры воды, вызывающие изменение скорости звука, и т.д.

Для изучения этих эффектов океанограф проделал множество последовательных считываний в месте, глубина которого *точно* известна и равна 1000 метров в глубину (истинное значение). Затем по результатам этих измерений была построена гистограмма, показанная на рис. 2.11. Как следует из центральной предельной теоремы, полученные данные подчиняются нормальному распределению. *Среднее значение* проходит через центр распределения и представляет лучшую оценку глубины, основанную на всех измеренных данных. Стандартное отклонение определяет ширину распределения, описывая, какова величина отклонений происходящих между последовательными измерениями.

Эта ситуация выражается двумя ошибками общего типа, которые может испытывать система. Первое, среднее значение может быть сдвинуто относительно истинного значения. Величина этого сдвига называется **погрешностью** измерения. Второе, индивидуальные измерения могут не согласовываться хорошо друг с другом, что показано шириной распределения. Это называется **точностью** измерения и выражается через стандартное отклонение, соотношение сигнал/шум или коэффициент вариации.

Рассмотрим измерение, имеющее хорошую погрешность, но плохую точность; центр гистограммы находится на среднем значении, но она очень широка. Хотя *групповые* измерения точны, каждое индивидуальное считывание является плохой мерой истинного значения. Говорят, что эта ситуация обладает плохой *повторяемостью*; измерения, сделанные последовательно, не согласуются хорошо. Плохая точность является результатом **случайных ошибок**. Это название, даваемое ошибкам, которые изменяются каждый раз, когда измерение повторяется. Усреднение нескольких измерений *всегда* будет улучшать точность. Короче говоря, *точность* - это *мера случайного шума*.

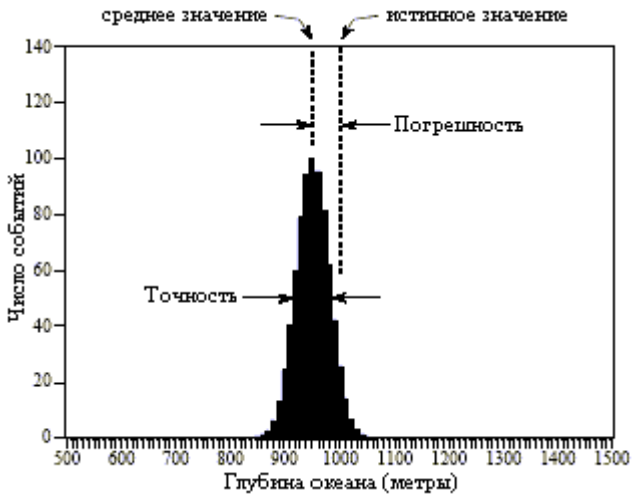


Рис. 2.11 Определение точности и погрешности

А теперь, представьте измерения, которые очень точны, но обладают плохой погрешностью. Это делает гистограмму, очень стройной, но не сосредоточенной вокруг

истинного значения. Последовательные считывания очень близки по значению, однако, *все* они имеют большую ошибку. Плохая точность следует из систематических ошибок. Это те ошибки, которые повторяются точно одним и тем же образом каждый раз, когда проводится измерение. Погрешность обычно зависит от того, как Вы *калибруете* систему. Например, при измерении океанских глубин, непосредственно измеряемый параметр время прохождения сигнала. Оно преобразуется в глубину с помощью процедуры калибровки, которая ставит в соответствие *миллисекундам метры*. Это может быть так же просто, как умножение на фиксированную скорость, или так же сложно, как дюжины исправлений второго порядка. Усреднение индивидуальных измерений ничего не дает для улучшения погрешности. Короче говоря, *погрешность* - это *мера калибровки*.

В реальной практике имеется много путей, где точность и погрешность могут переплетаться. Например, вообразите построенный электронный усилитель из 1% резисторов. Этот допуск указывает, что значение каждого резистора будет в пределах 1% от номинального значения в широком диапазоне изменения условий, таких как температура, влажность, срок службы и т.д. Эта ошибка в сопротивлении даст соответствующую ошибку усиления в усилителе. Эта ошибка - проблема погрешности или точности?

Ответ зависит от того, как Вы проводите измерения. Например, представьте, что Вы построили *один* усилитель и тестируете его несколько раз в течение нескольких минут. Ошибка в усилении остается постоянной при каждом испытании, и Вы заключаете, что проблемой является *погрешность*. Для сравнения, представьте, что Вы построили *тысячу* усилителей. От прибора к прибору усиление будет меняться случайным образом, и появившаяся проблема будет одной из проблем *точности*. Аналогично, любой из этих усилителей покажет изменение усиления в ответ на изменения температуры и других изменений окружающей среды. И снова проблема будет названа *точностью*.

Когда Вы решаете, каким именем назвать проблему, задайте себе два вопроса. Первый: Будет ли усреднение последовательных считываний обеспечивать улучшение измерений? Если да, зовите ошибку “точность”, если нет, зовите ее “погрешность”. Второй: Будет ли калибровка корректировать ошибку? Если да, зовите ее “погрешность”, если нет, зовите ее “точность”. Это может потребовать некоторых размышлений особенно относительно того, как будет выполняться калибровка прибора и, как часто это будет происходить.

ГЛАВА

3

АЦП И ЦАП

Большинство сигналов, с которыми непосредственно сталкиваются в науке и инженерной практике, относятся к *непрерывным*: изменяющаяся с расстоянием интенсивность света; изменяющееся во времени напряжение; зависящая от температуры скорость химической реакции и т.д. Аналого-цифровое и цифро-аналоговое преобразования это процессы, которые позволяют компьютерам обрабатывать эти повседневные сигналы. Цифровая информация отличается от своей непрерывной коллеги в двух важных аспектах: она *дискретна* и *квантована*. И то и другое определяет количество информации, которое может содержать цифровой сигнал. Эта глава посвящена *управлению информацией*: пониманию того, какую информацию Вы должны сохранить, и какую информацию Вы можете позволить себе потерять. В свою очередь это диктует выбор частоты дискретизации, числа разрядов и типа аналогового фильтра, необходимого для преобразования аналогового царства в цифровое.

Квантование

Сначала немного пустяков. Как Вы знаете, компьютер является *цифровым*, а не компьютером *цифр*. Обрабатываемая информация называется *цифровыми* данными, а не *цифирными* данными. Почему тогда аналого-цифровое преобразование вообще называют (в английском языке - прим. перев.) *дигитайз* (оцифровывать - прим. перев.) и *дигитизация* (оцифровка - прим. перев.), а не *дигиталайз* и *дигитализация*. Ответ не имеет ничего общего с тем, что бы Вы ожидали. Когда электроника добралась до изобретения цифровых методов, привилегированные названия почти столетие назад, уже были заняты медицинским сообществом. *Дигиталайз* и *дигитализация* означают принимать сердечное возбуждающее средство *дигиталис*. (Дигиталис – наперстянка – род травянистых растений, содержащих *гликозиды*, которые применяют в медицине. прим. перев.)

Рисунок 3.1 показывает формы волн электронных сигналов типичного аналого-цифрового преобразования. На рис. 3.1a показан аналоговый сигнал, который необходимо оцифровать. Как показано обозначениями на рисунке, этот сигнал является изменяющимся во *времени напряжением*. Чтобы сделать восприятие чисел более легкими, мы будем предполагать, что напряжение может измениться от 0 до 4,095 вольт, соответствуя цифровым уровням между 0 и 4095, получаемым 12 битовым аналого-цифровым преобразователем. Обратите внимание, что блок-схема разделена на две части, устройство выборки и хранения (УВХ) и аналого-цифровой преобразователь (АЦП). Как Вы, вероятно, узнали на занятиях по электронике, выборка и хранение требуются для того, чтобы на время пока происходит преобразование, держать напряжение, поступающее на вход АЦП постоянным. Однако это *не та* причина, по которой это обсуждается здесь, разбиение оцифровки на эти две стадии является важной теоретической моделью для ее понимания. Тот факт, что все это напоминает то, что происходит в обычной электронике, является только счастливым подарком.

Как видно из разницы между 3.1a и 3.1b, сигнал на выходе устройства выборки и

хранения может изменяться только через периодические интервалы времени, на протяжении которых он идентичен мгновенному значению входного сигнала. Изменения во входном сигнале, происходящие за время между двумя соседними отсчетами полностью игнорируются. То есть, **снятие отсчета** преобразует *независимую переменную* (в этом примере время) из непрерывной величины в дискретную.

Как видно из различий между 3.1b и 3.1c, для каждой из плоских областей в 3.1b, АЦП выдает значение целого числа между 0 и 4095. Это вносит ошибку, поскольку плато может принимать *любые* значения напряжения между 0 и 4,095 вольт. Например, и 2,56000 вольт, и 2,56001 вольт будут преобразованы в цифровой код 2560. Другими словами, **квантование** преобразует *зависимую переменную* (в этом примере напряжение) из непрерывной в дискретную.

Обратите внимание, что мы тщательно избегаем сравнения 3.1a и 3.1c, поскольку и дискретизация, и квантование вместе были бы неподъемной ношей. Очень важно, что мы анализируем их раздельно, поскольку они упрощают сигнал различными способами, также как и управляются в электронике различными параметрами. Встречаются также случаи, где они используются друг без друга. Например, дискретизация без квантования используется в фильтрах на переключаемых конденсаторах.

Посмотрим сперва на эффект квантования. Каждый одиночный отсчет в оцифрованном сигнале может иметь максимальную ошибку **±1/2 МЗР** (**младший значащий разряд**, жаргон для расстояния между соседними уровнями квантования). Рисунок 3.1d показывает ошибку квантования для этого конкретного примера, найденную вычитанием 3.1b из 3.1c, с соответствующими математическими преобразованиями. Другими словами, цифровой сигнал на выходе 3.1c является эквивалентным непрерывному входному сигналу 3.1b, *плюс* ошибка квантования 3.1d. Важной чертой этого анализа является то, что ошибка квантования возникает в большой степени подобно *случайному шуму*.

Это создает базу для важной модели ошибки квантования. В большинстве случаев *квантование выражается ни в чем ином как в добавлении определенного количества случайного шума к сигналу*. Аддитивный шум однородно распределен между ±1/2 МЗР, имеет нулевое среднее значение и стандартное отклонение равное $1/\sqrt{12}$ МЗР (-0,29 МЗР). Например, прохождение аналогового сигнала через 8 битовое оцифровывающее устройство добавляет среднеквадратический шум: 0,29/256, или около 1/900 полной шкалы значений. 12 битовое преобразование добавляет шум: 0,29/4096 = 1/14000, в то время как 16 битовое преобразование добавляет: 0,29/65536 = 1/227000. Поскольку ошибка квантования это случайный шум, *количество битов* определяет *точность* данных. Например, Вы можете делать следующие утверждения: “Мы увеличили точность с 8 до 12 битов”.

Эта модель чрезвычайно мощна, поскольку случайный шум, произведенный квантованием, просто добавится к любому уже существующему шуму в аналоговом сигнале. Например, представьте аналоговый сигнал с максимальной амплитудой 1,0 вольт и случайным шумом со среднеквадратическим значением 1,0 милливольт. Оцифровка этого сигнала до 8 битов приведет к тому, что 1,0 вольт, станет цифровым кодом 255, а 1,0 милливольт станет 0,255 МЗР. Как обсуждалось в предыдущей главе, при смешивании сигналов случайных шумов их дисперсии *складываются*. То есть, складываются квадраты сигналов: $\sqrt{A^2 + B^2} = C$. Следовательно, полный шум оцифрованного сигнала будет: $\sqrt{0{,}255^2 + 0{,}29^2} = 0{,}386$ МЗР. Это приблизительно 50%-е превышение шума, уже существующего в аналоговом сигнале. Оцифровка того же самого сигнала до 12 битов, фактически не приведет ни к какому увеличению шума, и из-за квантования *ничего* не будет потеряно. Когда требуется принять решение, сколько битов необходимо иметь в

системе, задайте себе два вопроса: (1) Каков уровень шума, *уже* присутствующего в аналоговом сигнале? (2) Какой уровень шума может быть *допустим* в цифровом сигнале?

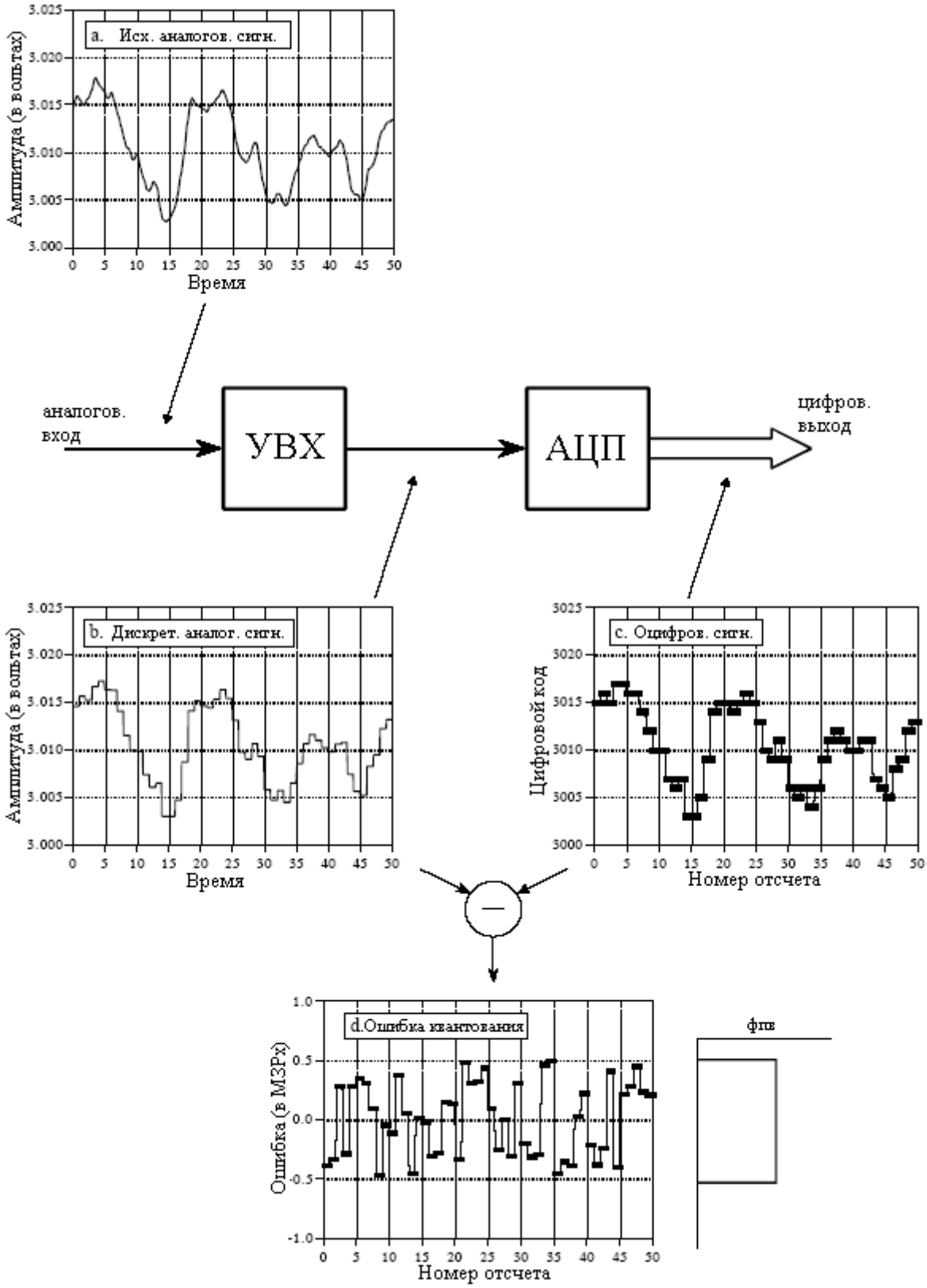


Рис. 3.1 Иллюстрация процесса оцифровки сигнала

Когда же модель квантования не является справедливой? Только тогда, когда с ошибкой квантования нельзя обращаться как со случайной величиной. Наиболее обычный

случай возникновения этого происходит тогда, когда для большого количества последовательных отсчетов аналоговый сигнал остается приблизительно на том же самом значении, как это проиллюстрировано на рис. 3.2a. На выходе для большого количества идущих подряд отсчетов остается *один и тот же* цифровой код даже тогда, когда аналоговый сигнал может изменяться в пределах ±1/2 МЗР. Вместо того чтобы быть аддитивным шумом, ошибка квантования теперь напоминает пороговый эффект или сверхъестественные искажения.

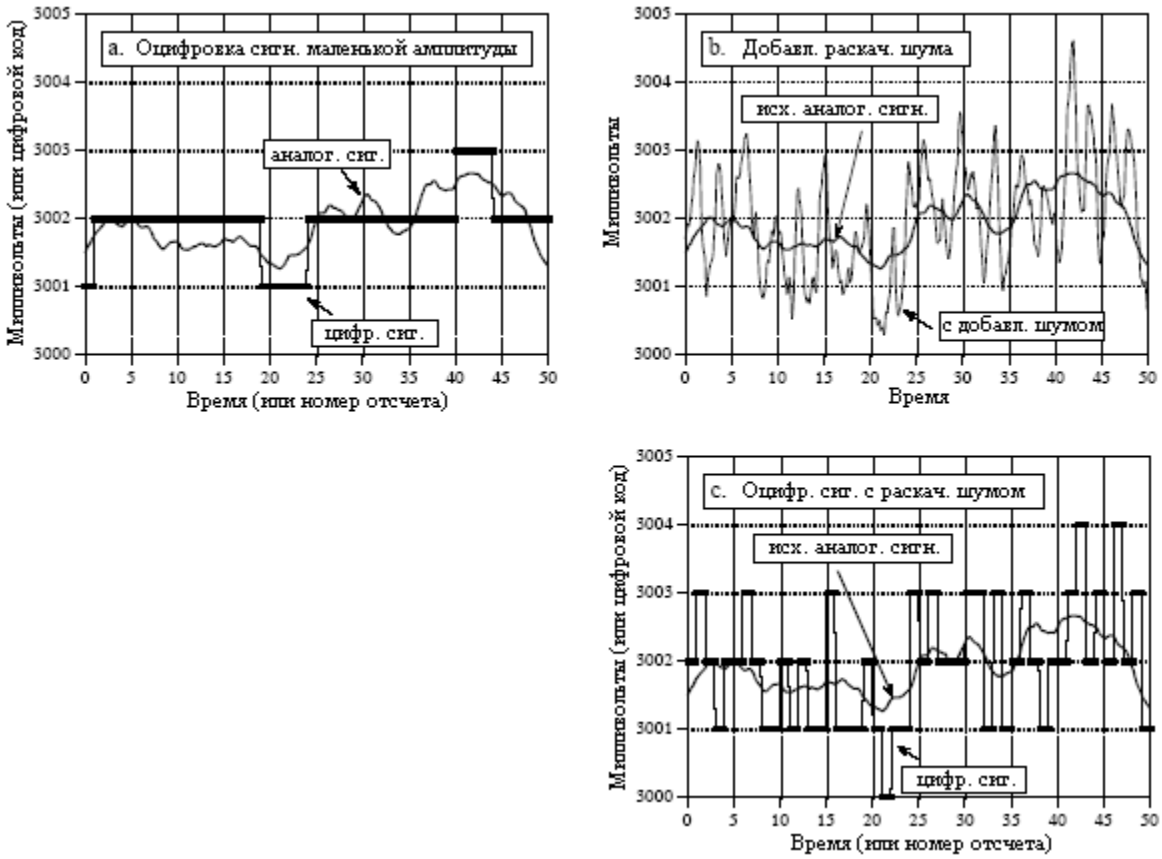


Рис. 3.2 Иллюстрация использования раскачивающего шума

Обычный метод улучшения оцифровки медленно меняющихся сигналов - это использование **раскачивающего шума**. Как показано на рис. 3.2b, к аналоговому сигналу добавляется случайный шум небольшой величины. В этом примере добавленный шум является шумом с нормальным распределением и стандартным отклонением равным 2/3 МЗР, выражающимся в изменении значения шума от максимума до минимума порядка 3 МЗР. Рисунок 3.2c показывает, как добавление этого раскачивающего шума воздействовало на оцифровываемый сигнал. Даже, когда исходный аналоговый сигнал изменяется меньше чем на ±1/2 МЗР, добавленный шум заставляет цифровой выходной сигнал случайным образом перескакивать между соседними уровнями.

Для того чтобы понять, каким образом это улучшает ситуацию, представьте, что входным сигналом является постоянное аналоговое напряжение порядка 3,0001 вольта, составляющее одну десятую часть расстояния между цифровыми уровнями 3000 и 3001. При отсутствии раскачивающего шума снятие 10000 отсчетов этого сигнала даст 10000 идентичных чисел, каждое из которых будет иметь значение 3000. Затем, повторим мысленный эксперимент с добавлением раскачивающего шума небольшой величины. Теперь 10000 значений будут осциллировать между двумя (или более) уровнями, в 90% случаев принимая значение 3000 и в 10% имея значение 3001. Вычисление среднего всех 10000 значений даст что-то близкое к 3000,1. Даже притом, что одиночное измерение

имеет присущее ему ограничение в ±1/2 МЗР, статистика огромного количества отсчетов может значительно улучшить ситуацию. Это весьма странная ситуация: *добавление шума обеспечивает большее количество информации*.

Схемы для получения раскачивающего шума могут быть весьма сложными, в частности использующими компьютер, для генерирования случайных чисел, подаваемых затем на ЦАП с целью получения добавляемого шума. После оцифровки компьютер может *вычесть* случайные числа из цифрового сигнала, используя арифметику с плавающей запятой. Эта изящная техника называется **вычитание раскачивающего шума**, она используется только в системах, проработанных детальнейшим образом. Самый простой метод, хотя и не всегда возможный, состоит в том, чтобы для получения раскачивающего шума использовать шум, уже представленный в аналоговом сигнале.

Теорема о дискретизации

Определение *правильного выполнения дискретизации* весьма просто. Предположим, что Вы некоторым образом снимаете отсчеты непрерывного сигнала. Если по снятым отсчетам Вы можете точно *восстановить* аналоговый сигнал, Вы выполнили дискретизацию *правильно*. Если Вы можете обратить процесс даже тогда, когда полученные в результате дискретизации данные кажутся запутанными и неполными, ключевая информация была поймана.

Рис. 3.3 показывает несколько синусоид до и после оцифровки. Непрерывной линией представлен сигнал, поступающий на вход АЦП, а квадратные маркеры показывают цифровой сигнал, покидающий АЦП. На рис.3.3a аналоговый сигнал - это постоянная составляющая неменяющейся величины, т.е. косинусоида с *нулевой* частотой. Поскольку аналоговый сигнал - это ряд прямых линий между двумя соседними отсчетами, вся информация, необходимая для восстановления аналогового сигнала, содержится в цифровых данных. Согласно нашему определению, это *надлежащее осуществление дискретизации*.

Синусоидальная волна, показанная на рис. 3.3b, имеет частоту 0,09 частоты дискретизации. Это может, например, представлять дискретизацию синусоиды частотой 90 периодов/сек., со скоростью 1000 отсчетов/сек. Выражаясь другим способом, имеются 11,1 отсчетов, взятых в течение каждого полного периода синусоиды. Эта ситуация значительно сложнее чем в предыдущем случае, поскольку аналоговый сигнал не может быть восстановлен простым изображением прямых линий между точками данных. Представляют ли эти отсчеты аналоговый сигнал должным образом? Ответом будет - да, поскольку никаких других синусоид или комбинаций синусоид этот образец отсчетов не даст (в пределах разумных ограничений, приведенных ниже). Эти отсчеты соответствуют только одному аналоговому сигналу и, следовательно, аналоговый сигнал может быть точно восстановлен. Снова случай *надлежащего осуществления дискретизации*.

На рис. 3.3c из-за увеличения частоты синусоиды до 0,31 частоты дискретизации созданная ситуация более трудная. Это выражается в снятии только 3,2 отсчетов за весь период синусоиды. Здесь отсчеты настолько редки, что они кажется, даже не следуют за общей тенденцией аналогового сигнала. Должным ли образом представляют эти отсчеты аналоговую форму волны? И снова ответом будет - да и точно по той же причине. Отсчеты являются единственным представлением аналогового сигнала. В цифровых данных содержится вся информация, необходимая для восстановления непрерывной формы волны. Позже, в этой главе будет обсуждено, как Вам подойти к выполнению восстановления. Очевидно, что это должно быть более сложно, чем просто изображение прямых линий между точками данных. Как это ни покажется странным, согласно нашему определению это *надлежащее осуществление дискретизации*.

На рис. 3.3d аналоговая частота сдвинута еще выше до 0,95 частоты дискретизации с явными 1,05 отсчетов за период синусоиды. Представляют ли эти отсчеты, данные

правильно? *Нет, не представляют*! Отсчеты представляют синусоидальную волну *отличную* от той, что содержится в аналоговом сигнале. В частности, исходная синусоидальная волна с частотой 0,95 от частоты дискретизации искажается в цифровом сигнале до синусоидальной волны с частотой 0,05 от частоты дискретизации. Это явление, когда синусоида меняет свою частоту во время дискретизации, называется **совмещением имен** (это явление еще называют стробоскопическим эффектом – прим. перев.). Так же, как преступник может взять и присвоить себе чужое имя или взять псевдоним (*прозвище*), синусоида присваивает себе другую частоту, которая не является ее собственной. Поскольку цифровые данные больше не связаны с конкретным аналоговым сигналом единственным образом, однозначное восстановление невозможно. В данных прошедших через дискретизацию нет ничего, позволяющего предположить, что исходный аналоговый сигнал скорее имел частоту *0,95*, а не 0,05. Синусоидальная волна полностью скрыла свое истинное лицо; идеальное преступление совершилось! В соответствии с нашим определением, это пример *ненадлежащего осуществления дискретизации*.

Эта строка рассуждений ведет к краеугольному камню в ЦОС - **теореме о дискретизации**. Часто, после авторских публикаций на эту тему в 1940-х, ее называют теоремой о дискретизации *Шеннона* или теоремой о дискретизации *Найквиста* (в отечественной литературе эту теорему, впервые доказанную В.А. Котельниковым еще в 1933 году, называют **теоремой Котельникова** – прим. перев.). Теорема о дискретизации указывает, что непрерывный сигнал, может быть подвергнут дискретизации надлежащим образом, *только если сигнал не содержит частотных составляющих выше половины частоты дискретизации*. Например, частота дискретизации 2000 отсчетов/сек. требует, чтобы аналоговый сигнал был составлен из частот ниже 1000 периодов/сек. Если в сигнале *присутствуют* частоты выше этого ограничения, то они будут перемещены к частотам между 0 и 1000 периодов/сек., комбинируясь там с любой законно находящейся информацией.

При обсуждении теоремы о дискретизации широко используются два термина: **частота Найквиста** и **скорость Найквиста**. К сожалению, их значения не стандартизированы. Для того чтобы понять их, рассмотрим аналоговый сигнал, состоящий из частот между постоянной составляющей и 3 кГц. Для правильной дискретизации этого сигнала снятие отсчетов должно производиться со скоростью 6000 отсчетов/сек. (6 кГц) или выше. Предположим, что мы выбрали частоту дискретизации 8000 отсчетов/сек. (8 кГц), позволяя частотам между постоянной составляющей и 4 кГц быть представленными надлежащим образом. В этой ситуации есть четыре важных частоты: (1) верхняя частота в сигнале - 3 кГц; (2) удвоенная эта частота - 6 кГц; (3) частота дискретизации - 8 кГц; (4) половина частоты дискретизации - 4 кГц. Какая из этих четырех - *частота Найквиста*, а какая - *скорость Найквиста*? Все зависит от того, кого Вы спрашиваете! Используются все возможные комбинации. К счастью большинство авторов тщательно определяют то, как они используют термины. В этой книге оба термина используются для обозначения *половины частоты дискретизации*.

Рис. 3.4 показывает, как меняются частоты во время совмещения имен. Ключевым моментом, о котором необходимо помнить, является то, что цифровой сигнал *не может* содержать частоты выше половины частоты дискретизации (т.е., частоты/скорости Найквиста). Когда частота непрерывной волны ниже частоты Найквиста, частота данных после дискретизации соответствует ей. Однако, когда частота непрерывного сигнала выше частоты Найквиста, совмещение имен *заменяет* его собственную частоту на что-то, что *должно* быть представлено в данных прошедших дискретизацию. Как показано зигзагообразной линией на рис. 3.4, все частоты выше частоты Найквиста в непрерывных сигналах имеют соответствующие частоты между нулем и частотой дискретизации в цифровых сигналах. Если так случится, что на этих более низких частотах уже имеется синусоида, сигнал с совмещенным именем сложится с ней, в результате произойдет потеря информации. Совмещение имен это двойное проклятие; информация может быть

потеряна и в районе более высоких *и* в районе более низких частот. Предположим, Вам дан цифровой сигнал, содержащий частоту, составляющую 0,2 от частоты дискретизации. Если этот сигнал был получен надлежащим осуществлением дискретизации, исходный аналоговый сигнал *должен* иметь частоту 0,2. Если во время осуществления дискретизации имело место совмещение имен, цифровая частота 0,2 могла получиться от любого бесконечного числа частот в аналоговом сигнале: *0,2*, *0,8*, *1,2*, *1,8*, *2,2*, ….

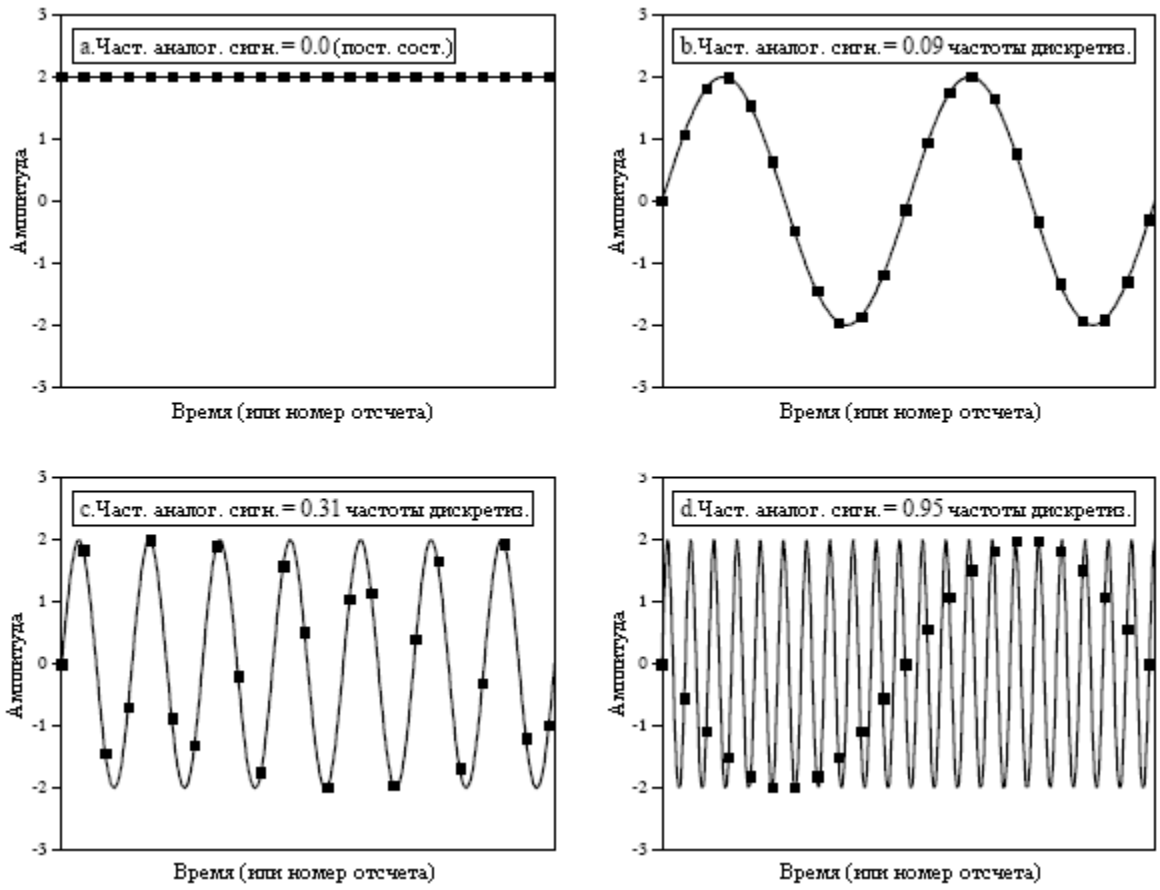


Рис. 3.3 Иллюстрация надлежащего и ненадлежащего осуществления дискретизации

Так же, как совмещение имен может изменять частоту во время дискретизации, оно может изменять и *фазу*. Для примера вернемся к совмещению имен для сигнала на рис. 3.3d. Цифровой сигнал с совмещенным именем *инвертирован* по отношению к исходному аналоговому сигналу; один является синусоидальной волной, в то время как другой - отрицательной синусоидальной волной. Другими словами, совмещение имен изменило частоту *и* ввело 180^{O} фазовый сдвиг. Возможны только два фазовых сдвига: 0^{O} (фазового сдвига нет) и 180^{O} (инверсия). Нулевой сдвиг фазы происходит для аналоговых частот от *0* до *0,5*, от *1,0* до *1,5*, от *2,0* до *2,5* и т.д. Инвертирование фазы происходит для аналоговых частот от *0,5* до *1,0*, от *1,5* до *2,0*, от *2,5* до *3,0* и т.д.

Сейчас мы погрузимся в более детальный анализ процесса дискретизации и того, как происходит совмещение имен. Наша конечная цель состоит в том, чтобы понять, что происходит с информацией, когда сигнал преобразуется из непрерывной формы в дискретную. Проблема состоит в том, что здесь присутствуют очень разные вещи; одна является *непрерывной формой волны*, в то время как другая является *множеством чисел*. Это сравнение, “яблок с апельсинами”, делает анализ очень трудным. Решение находится во введении теоретической концепции называемой **импульсной последовательностью**.

На рисунке 3.5a показан пример аналогового сигнала. Рисунок 3.5c показывает сигнал, дискретизация которого выполнена с использованием *импульсной*

последовательности. Импульсная последовательность - это непрерывный сигнал, состоящий из ряда узких пиков, соответствующих исходному сигналу в моменты снятия отсчета. Ширина каждого импульса бесконечно мала, это понятие будет обсуждено в Главе 13. Во время между отсчетами форма волны имеет нулевое значение. Имейте в виду, что импульсная последовательность это *теоретическая* концепция, а не форма волны, которая может существовать в электронной схеме. Поскольку оба и исходный аналоговый сигнал, и импульсная последовательность являются непрерывными формами волны, мы можем проводить между ними сравнение “яблок с яблоками”.

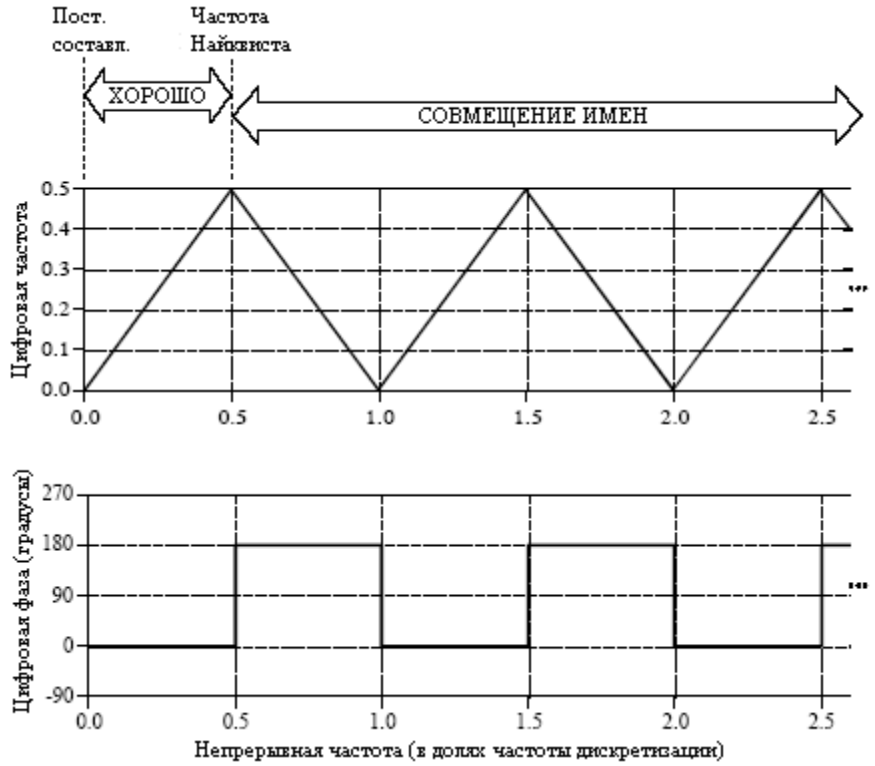


Рис. 3.4 Преобразование аналоговой частоты в цифровую частоту в процессе дискретизации

Теперь мы должны исследовать соотношения между импульсной последовательностью и дискретным сигналом (множеством чисел). Сделать именно это - легко, с точки зрения *содержания информации* они *идентичны*. Если один из них известен, вычисление другого становится тривиальным. Рассматривайте их как различные концы моста, переброшенного между аналоговым и цифровым мирами. Это означает, что раз мы понимаем последствия замены формы волны на рис. 3.5a формой волны на рис. 3.5c, то мы достигли нашей конечной цели.

В левом столбце на рис. 3.5 показаны три непрерывных формы волны. Соответствующие *частотные спектры* этих сигналов показаны в правом столбце. Такая концепция должна быть Вам знакома из электроники; любая форма волны может рассматриваться как состоящая из синусоид различной амплитуды и частоты. Частотная область будет подробно обсуждаться в следующих главах. (Вы можете повторно рассмотреть данное обсуждение после более подробного знакомства с частотными спектрами).

На рисунке 3.5a показан аналоговый сигнал, дискретизацию которого мы хотим произвести. Как видно из его частотного спектра, приведенного на рис. 3.5b, он состоит только из частотных составляющих, находящихся между 0 и приблизительно 0,33 f_s, где f_s – частота дискретизации, которую мы решили использовать. Например, это мог бы быть

сигнал речи, прошедший через фильтр с целью удаления всех частот, лежащих выше, чем 3,3 кГц. Соответственно, выбранная нами частота дискретизации f_s была бы 10 кГц (10000 отсчетов/сек.).

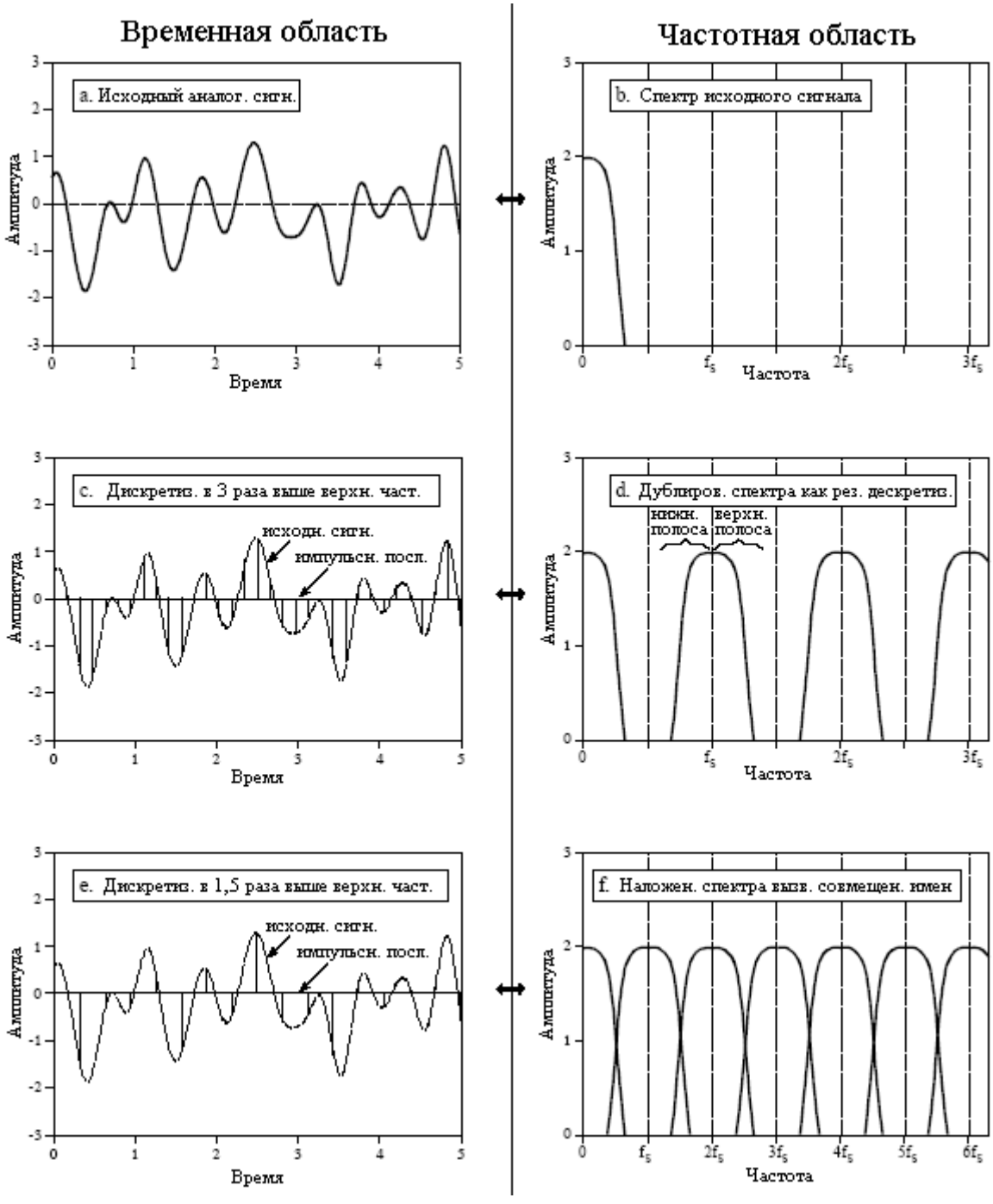


Рис. 3.5 Теорема о дискретизации во временной и в частотной областях

В результате дискретизации сигнала на рис. 3.5a с помощью импульсной последовательности получается сигнал, показанный на рис. 3.5c, частотный спектр которого приведен на рис. 3.5d. Этот спектр представляет собой *дублирование* спектра исходного сигнала. Каждая частота кратная частоте дискретизации f_s, $2f_s$, $3f_s$, $4f_s$, и т.д. получает *копию* исходного спектра частот и еще одну, *перевернутую слева направо копию* этого же исходного спектра частот. Копия исходного спектра называется **верхней боковой полосой** частот, в то время как перевернутая копия **нижней боковой полосой** частот. Дискретизация создает *новые* частоты. Является ли она надлежащей

дискретизацией? Ответом является да, поскольку сигнал на рис. 3.6c может быть преобразован обратно в сигнал на рис. 3.6a удалением всех частот выше $1/2f_s$. То есть, аналоговый фильтр нижних частот преобразует импульсную последовательность на рис. 3.6c обратно в исходный аналоговый сигнал на рис. 3.6a.

Если Вы уже познакомились с основами ЦОС, приведем техническое объяснение того, почему происходит такое дублирование спектра. (Опустите этот параграф, если Вы новичок в ЦОС.) Во временной области дискретизация достигается умножением исходного сигнала на импульсную последовательность пиков *единичной амплитуды*. Частотный спектр этой импульсной последовательности единичной амплитуды тоже является импульсной последовательностью единичной амплитуды с пиками, расположенными на частотах кратных частоте дискретизации f_s, $2f_s$, $3f_s$, $4f_s$, и т.д. Когда выполняется умножение двух сигналов во временной области, происходит свертка их частотных спектров. Это приводит к тому, что в спектре импульсной последовательности в местах каждого пика происходит дублирование исходного спектра. Взгляд на исходный сигнал, как на сигнал, состоящий из положительных и отрицательных частот, объясняет, соответственно, верхние и нижние боковые полосы. Это то же самое, что и амплитудная модуляция, обсуждаемая в Главе 10.

На рисунке 3.5e показан пример ненадлежащей дискретизации, возникающей при очень низкой частоте дискретизации. Аналоговый сигнал все также содержит частоты до 3,3 кГц, однако, частота дискретизации была снижена до 5 кГц. Обратите внимание, что f_s, $2f_s$, $3f_s$, … на рис. 3.5f расположены по горизонтальной оси ближе друг к другу, чем на рис. 3.5d. Частотный спектр на рис. 3.5f раскрывает проблему: дублированные части спектра захватывают полосу между нулем и половиной частоты дискретизации. Хотя на рис. 3.5f эти накладывающиеся частоты показаны с сохранением идентификации каждой из них, в реальной практике они складываются вместе, формируя единый запутанный беспорядок. Поскольку для разделения накладывающихся частот нет никакого способа, происходит потеря информации, и исходный сигнал не может быть восстановлен. Такое наложение происходит тогда, когда аналоговый сигнал содержит частоты большие чем половина частоты дискретизации, это означает, что мы доказали теорему о дискретизации.

Цифро-аналоговое преобразование

В теории самый простой способ осуществления цифро-аналогового преобразования состоит в том, чтобы считать отсчеты из памяти и преобразовать их в *импульсную последовательность*. Это иллюстрируется рис. 3.6a с соответствующим частотным спектром на рис. 3.6b. Как только что описывалось, исходный аналоговый сигнал может быть достаточно точно восстановлен. Это можно осуществить, пропуская выше упомянутую импульсную последовательность через фильтр нижних частот с частотой среза, равной половине частоты дискретизации. Другими словами, ниже частоты Найквиста (половина частоты дискретизации) исходный сигнал и импульсная последовательность обладают идентичным частотным спектром. На более высоких частотах импульсная последовательность содержит дублирование этой информации, в то время как первоначальный аналоговый сигнал здесь ничего не содержит (предполагаемого совмещения имен не произошло).

Не смотря на то, что этот метод математически точен, в электронике трудно генерировать требуемые узкие импульсы. Для того чтобы это обойти, почти все ЦАП работают, удерживая последнее значение до тех пор, пока не будет получен другой отсчет. Это называется **хранением нулевого порядка** (в отечественной литературе часто называют интерполяцией нулевого порядка – прим. перев.), в цифро-аналоговом преобразовании это эквивалент, используемым во время аналого-цифрового преобразования, "выборке и хранению". (Хранение первого порядка - это прямые линии между точками, хранение второго порядка использует параболы и т.д.). Хранение

нулевого порядка дает появление лестницы, показанной на рис 3.6c. В частотной области хранение нулевого порядка выражается в том, что спектр импульсной последовательности умножается на жирную кривую, показанную на рис. 3.6d и задаваемую уравнением:

$$H(f) = \left| \frac{\sin(\pi f / f_s)}{\pi f / f_s} \right| \qquad (3.1)$$

Это общая форма $\sin(\pi x) / (\pi x)$, называемая **синк функцией** или ***sinc*(*x*)**. Функция синк в ЦОС является очень обычной и будет обсуждена более подробно в следующих главах. Если у Вас уже есть подготовка по этому материалу, то хранение нулевого порядка может быть понято как свертка импульсной последовательности с прямоугольным импульсом, имеющим ширину равную периоду дискретизации. В частотной области это приводит к *умножению* на результат преобразования Фурье прямоугольного импульса, т.е. на синк функцию. На рис. 3.6d тонкая линия показывает частотный спектр импульсной последовательности ("правильный" спектр), в то время как жирная линия показывает синк функцию. Частотный спектр сигнала хранения нулевого порядка равен произведению этих двух кривых.

Аналоговый фильтр, используемый для преобразования сигнала хранения нулевого порядка (рис. 3.6c) в восстановленный сигнал (рис. 3.6f), должен делать две вещи: (1) удалять все частоты выше половины частоты дискретизации и (2) увеличивать амплитуды частотных составляющих на обратную величину от эффекта хранения нулевого порядка, то есть, на *1/sinc(x)*. Величина этого увеличения амплитуды приблизительно 36% на частоте равной половине частоты дискретизации. На рисунке 3.6e показана идеальная частотная характеристика этого аналогового фильтра.

С повышением амплитуды частотных составляющих на *1/sinc(x)* можно поступить четырьмя способами: (1) игнорировать его и согласиться с последствиями, (2) спроектировать аналоговый фильтр, добавляющий *1/sinc(x)* в частотную характеристику, (3) использовать виртуальную *мультичастотную* технику, описанную позже в этой главе или (4) делать коррекцию с помощью программного обеспечения перед ЦАП (см. Главу 24).

Перед тем как покинуть этот раздел по дискретизации, мы должны рассеять всеобщий миф, касающийся сравнения аналоговых сигналов против цифровых. Как показала эта глава, количество информации переносимое в цифровом сигнале ограничивается двумя факторами. Первое, число битов, приходящихся на один отсчет, ограничивает разрешение *зависимой* переменной. То есть, небольшие изменения в амплитуде сигнала могут затеряться в шуме квантования. Второе, частота дискретизации ограничивает разрешение *независимой* переменной, т.е. близко расположенные события в аналоговом сигнале могут быть потеряны между отсчетами. Это еще один способ сказать, что частоты выше половины частоты дискретизации теряются.

А вот и миф: “Поскольку аналоговые сигналы используют непрерывные параметры, они обладают бесконечно хорошим разрешением по обеим независимой и зависимой переменным”. Не верно! Аналоговые сигналы ограничиваются теми же самыми двумя проблемами, что и цифровые сигналы: *шумы* и *ширина полосы* (самая высокая частота допустимая в сигнале). Шумы в аналоговом сигнале ограничивают измерение амплитуды формы волны тоже, что делает шум квантования в цифровом сигнале. Аналогично, способность разделить близко расположенные события в аналоговом сигнале зависит от самой высокой частоты, допустимой в форме волны. Для того чтобы понять это, представьте аналоговый сигнал, содержащий два близко расположенных импульса. Если мы пропустим сигнал через низкочастотный фильтр

(удаляющий недопустимые высокие частоты), импульсы будут смазаны в отдельную каплю. Например, аналоговый сигнал, сформированный из частот, расположенных между

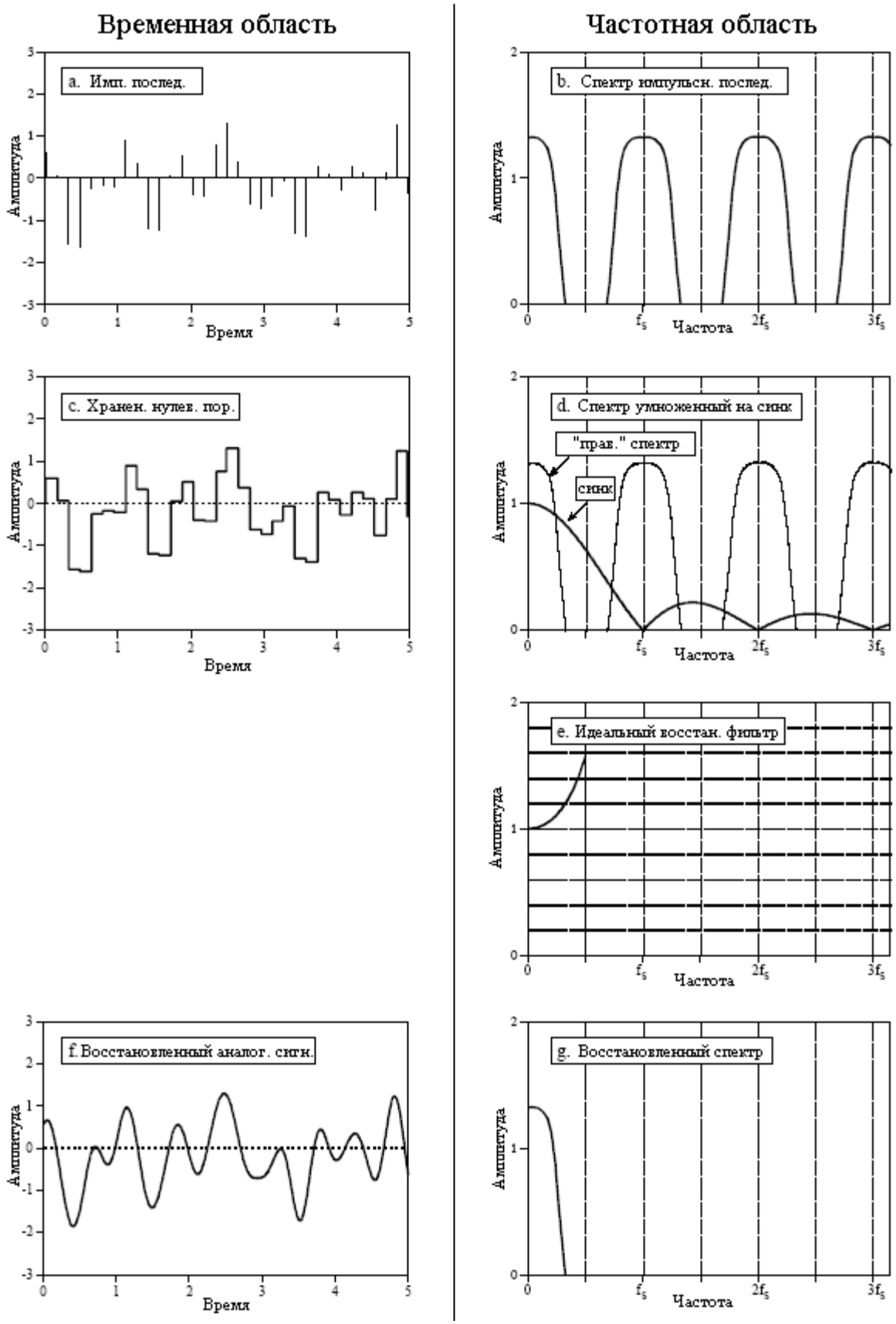


Рис. 3.6 Анализ цифро-аналогового преобразования

постоянной составляющей и 10 кГц, будет иметь *точно* такое же разрешение, как и цифровой сигнал с частотой дискретизации 20 кГц. Они должны содержать ту же самую информацию, поскольку это гарантирует теорема о дискретизации.

Аналоговые фильтры для преобразования данных

На рисунке 3.7 приведена блок схема системы с ЦОС такая, какой диктует ей быть теорема о дискретизации. Перед тем как попасть на аналого-цифровой преобразователь, входной сигнал обрабатывается электронным фильтром низких частот, удаляющим все частоты лежащие выше частоты Найквиста (половина частоты дискретизации). Фильтр делает это для предотвращения совмещения имен во время дискретизации, и соответственно называется **фильтром антисовмещения**. На другом конце оцифрованный сигнал проходит через цифро-аналоговый преобразователь и другой низкочастотный фильтр, настроенный на частоту Найквиста. Этот выходной фильтр называется **восстанавливающим фильтром** и может включать ранее описанное повышение амплитуды частотных составляющих на обратную величину от эффекта хранения нулевого порядка. К сожалению, у этой простой модели есть серьезная проблема: ограничения, накладываемые электронными фильтрами, могут быть настолько же плохими, как и проблемы, которые они пытаются предотвратить.

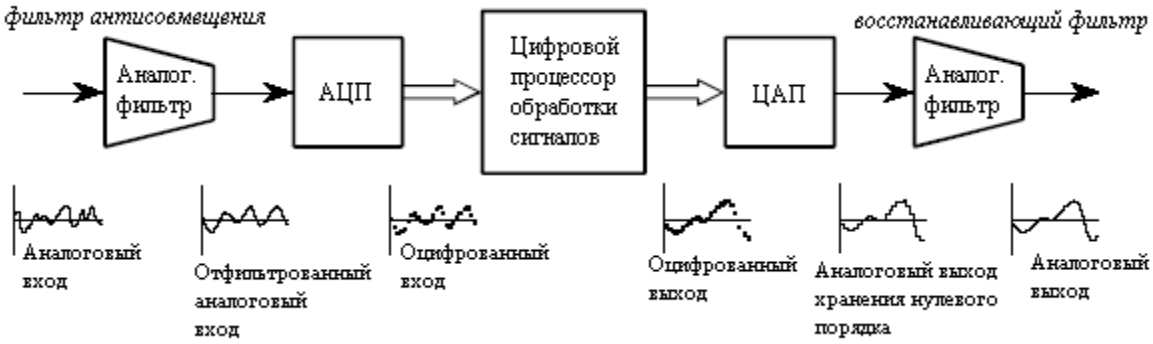


Рис. 3.7 Аналоговые электронные фильтры, используемые в соответствии с теоремой о дискретизации

Если Ваши главные интересы сосредоточены на программном обеспечении, Вы, вероятно думаете, что Вам не нужно читать этот раздел. *Неверно*! Даже если Вы поклялись никогда не касаться осциллографа, понимание свойств аналоговых фильтров является важным для успешной реализации ЦОС. Во-первых, характеристики каждого цифрового сигнала, с которым Вы сталкиваетесь, будут зависеть от типа, используемого во время получения сигнала, фильтра антисовмещения. Если Вы не понимаете природу фильтра антисовмещения, Вы не сможете понять природу цифрового сигнала. Во-вторых, будущее ЦОС состоит в замене *аппаратуры* на *программное* обеспечение. Например, *мультичастотная* техника, рассматриваемая позднее в этой главе, снижает потребность в фильтрах антисовмещения и в восстанавливающих фильтрах с помощью фантастических программных трюков. Если Вы не понимаете аппаратуру, Вы не сможете разрабатывать заменяющие ее программы. В-третьих, многое в ЦОС связано с проектированием цифровых фильтров. Обычный подход к проектированию в таких случаях это начать с эквивалентных *аналоговых фильтров* и преобразовать их в программное обеспечение. Поэтому следующие главы подразумевают, что Вы обладаете базовыми знаниями в технике аналоговых фильтров.

Обычно используется три типа аналоговых фильтров: **Чебышева**, **Баттерворта** и **Бесселя** (называемого также фильтром Томпсона). Каждый из них разрабатывался для

оптимизации различных рабочих параметров. Сложность каждого фильтра может быть задана выбором числа **полюсов** и **нулей**, математические термины, которые будут обсуждены в более поздних главах. Чем больше в фильтре полюсов, тем больше электроники ему требуется и тем лучше он работает. Каждое из этих названий описывает то, что *делает* фильтр, а не конкретный порядок расположения резисторов и конденсаторов. Например, шести полюсный фильтр Бесселя может быть реализован большим количеством схем разного типа, каждая из которых имеет те же самые обобщающие характеристики. Для целей ЦОС характеристики этих фильтров более важны чем то, как они построены. Однако мы начнем с короткого сегмента по проектированию электронных фильтров для получения общего представления.

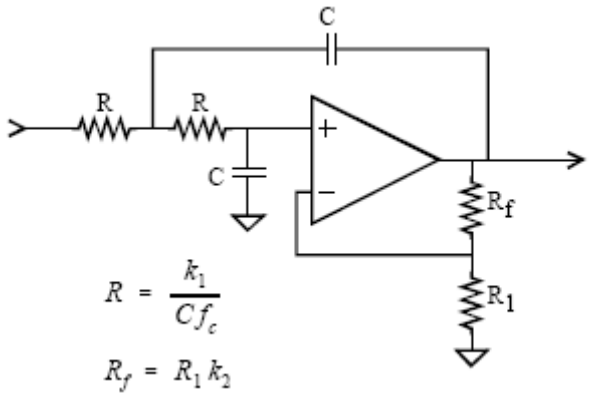


Рис. 3.8 Модифицированная схема Саллена-Ки, конструктивного блока для проектирования активных фильтров

Таблица 3.1

Число полюсов		Фильтр Бесселя		Фильтр Баттерворта		Фильтр Чебышева (6% пульсация)	
		k_1	k_2	k_1	k_2	k_1	k_2
2	Блок 1	0,1251	0,268	0,1592	0,586	0,1293	0,842
4	Блок 1	0,1111	0,084	0,1592	0,152	0,2666	0,582
	Блок 2	0,0991	0,759	0,1592	1,235	0,1544	1,660
6	Блок 1	0,0990	0,040	0,1592	0,068	0,4019	0,537
	Блок 2	0,0941	0,364	0,1592	0,586	0,2072	1,448
	Блок 3	0,0834	1,023	0,1592	1,483	0,1574	1,846
8	Блок 1	0,0894	0,024	0,1592	0,038	0,5359	0,522
	Блок 2	0,0867	0,213	0,1592	0,337	0,2657	1,379
	Блок 3	0,0814	0,593	0,1592	0,889	0,1848	1,711
	Блок 4	0,0726	1,184	0,1592	1,610	0,1582	1,913

На рисунке 3.8 показан типичный конструктивный блок для проектирования аналогового фильтра, называемый схемой Саллена-Ки. Схема названа по имени авторов после того, как в 1950-х появилась статья, описывающая данную технику. Приведенная схема является двухполюсным низкочастотным фильтром, который может иметь конфигурацию любого из трех основных типов. В таблице 3.1 приведена необходимая информация для выбора соответствующих значений резисторов и конденсаторов. Например, для разработки двухполюсного фильтра с частотой среза 1 кГц таблица 3.1 дает параметры: $k_1 = 0{,}1592$ и $k_2 = 0{,}586$. Произвольно выбирая $R_1 = 10$ кОм и $C = 0.01$ мкФ (типичные значения для схем на операционных усилителях), R и R_f могут быть вычислены

как 15,92 кОм и 5,86 кОм, соответственно. Округление этих двух последних значений до ближайших стандартных значений резисторов с 1% допуском в результате даст R = 15,8 кОм и R_f = 5,90 кОм. Все компоненты должны иметь 1% точность и выше.

До тех пор, пока частота общего усиления более чем от 30 до 100 раз превышает частоту среза фильтра, использование конкретного типа операционного усилителя не является критическим. Это легко выполнимое требование до тех пор, пока частота среза фильтра лежит ниже приблизительно 100 кГц.

Четырех, шести и восьми полюсные фильтры формируются последовательным соединением 2, 3, и 4 таких блоков, соответственно. Для примера на рис. 3.9 приведена схема шести полюсного фильтра Бесселя с частотой среза 1 кГц, полученная последовательным соединением трех блоков. Каждый блок имеет разные коэффициенты k_1 и k_2, как предусмотрено таблицей 3.1, дающие в итоге разные величины используемых резисторов и конденсаторов. Нуждаетесь в высокочастотном фильтре? Просто поменяйте в схеме местами компоненты R и C (оставьте R_f и R_1 в покое).

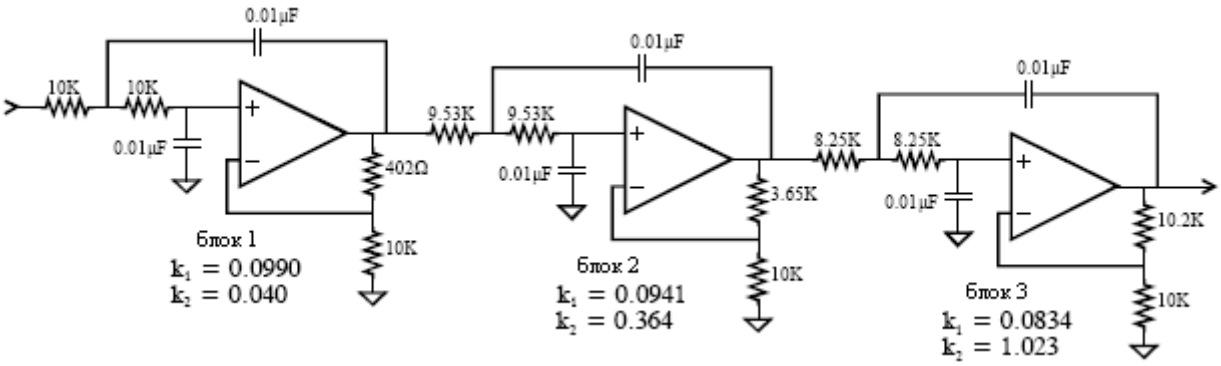


Рис.3.9 Шестиполюсный фильтр Бесселя, сформированный последовательным соединением трех схем Саллена-Ки

Такой вид схем очень типичен для мелкосерийного производства, научно-исследовательских приложений и опытных разработок; однако, серьезное производство требует, чтобы фильтр был сделан как *интегральная микросхема*. Проблема состоит в том, что делать резисторы непосредственно в кремнии достаточно трудно. Ответом служат **фильтры на переключаемых конденсаторах**. Рисунок 3.10 иллюстрирует их работу при помощи сравнения их с простой *R-C* цепью. Если на низкочастотный *R-C* фильтр поступает ступенчатая функция, сигнал на выходе экспоненциально нарастает, пока не сравняется с входным сигналом. Напряжение на конденсаторе не может измениться мгновенно, поскольку резистор ограничивает поток электрического заряда.

Фильтр на переключаемом конденсаторе работает, заменяя основную резисторно-конденсаторную цепь двумя конденсаторами и электронным ключом. Новый добавленный конденсатор намного меньше по величине, чем уже существующий конденсатор, скажем, 1 % от его значения. Ключ попеременно с очень высокой частотой, обычно в 100 раз выше, чем частота среза фильтра, переключает маленький конденсатор между входом и выходом. Когда ключ присоединен к выходу, заряд маленького конденсатора передается большому конденсатору. В резисторе величина передаваемого заряда определяется значением его сопротивления. В схеме с переключаемым конденсатором величина передаваемого заряда определяется значением маленького конденсатора *и* частотой переключения. Следствием этого является очень полезная характеристика фильтров на переключаемых конденсаторах: *частота среза фильтра прямо пропорциональна тактовой частоте используемой для управления ключами*. Это делает фильтр на переключаемых конденсаторах идеальным для систем сбора данных, работающих с большим, чем одна, числом частот дискретизации. Эти приборы достаточно

легко использовать; плати десять долларов и у тебя есть работающий восьми полюсный фильтр внутри одной 8 - контактной интегральной микросхемы.

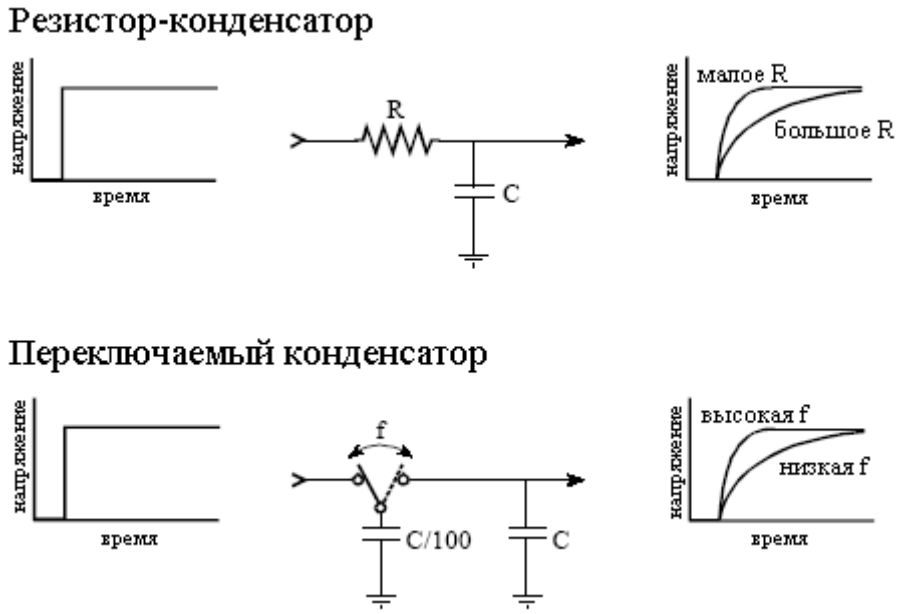


Рис. 3.10 Функционирование фильтра на переключаемых конденсаторах

Ну а сейчас к важной части: характеристикам трех фильтров классического типа. Первый рабочий параметр, который мы хотим исследовать, это **крутизна частоты среза**. Низкочастотный фильтр разрабатывается для того, чтобы не пропускать все частоты, лежащие выше частоты среза (**заграждающий**), и в то же время пропускать все частоты, лежащие ниже частоты среза (**пропускающий**). На рис. 3.11 показаны частотные характеристики всех трех фильтров в логарифмическом (dB) масштабе. Эти графики приведены для фильтров с частотой среза один герц, но они непосредственно могут быть смасштабированы на любую частоту среза, которую Вам нужно использовать. Какова крутизна частотных характеристик этих фильтров? Фильтр Чебышева, несомненно, лучший, Баттерворта похуже и Бесселя абсолютно ужасный! Как Вы, вероятно и предполагали, то, для чего предназначен фильтр Чебышева, это осуществить **спад** (уменьшить по амплитуде) нежелательных частот так быстро, насколько это возможно.

К сожалению, даже 8 полюсный фильтр Чебышева не так хорош в качестве фильтра антисовмещения, как Вам бы хотелось. Например, представьте 12 битовую систему, снимающую отсчеты со скоростью 10000 отсчетов/сек. Согласно теореме о дискретизации любая частота выше 5 кГц подвергнется совмещению имен, это как раз то, чего Вы хотите избежать. После небольших раздумий Вы решаете, что все частоты выше 5 кГц должны быть уменьшены по амплитуде в 100 раз, гарантируя, что любые частоты с совмещенными именами будут иметь амплитуду меньше одного процента. Глядя на рис. 3.11 Вы найдете, что 8 полюсный фильтр Чебышева с частотой среза 1 герц не достигает подавления (уменьшения сигнала) в 100 раз вплоть до 1,35 герца. Приводя все это к масштабу нашего примера, для того чтобы все, что свыше 5 кГц имело требуемое подавление, частота среза фильтра должна быть 3,7 кГц. Это приведет к тому, что полоса частот между 3,7 кГц и 5 кГц будет напрасно и неадекватно завалена аналоговым фильтром.

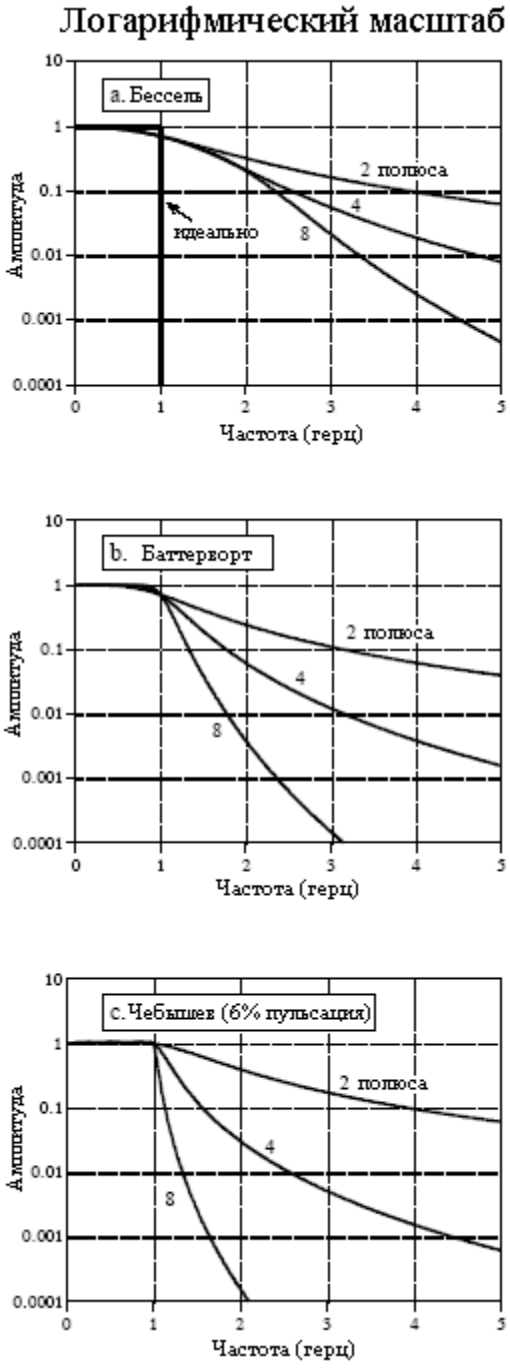


Рис. 3.11
Частотные характеристики фильтров
в *логарифмическом* масштабе

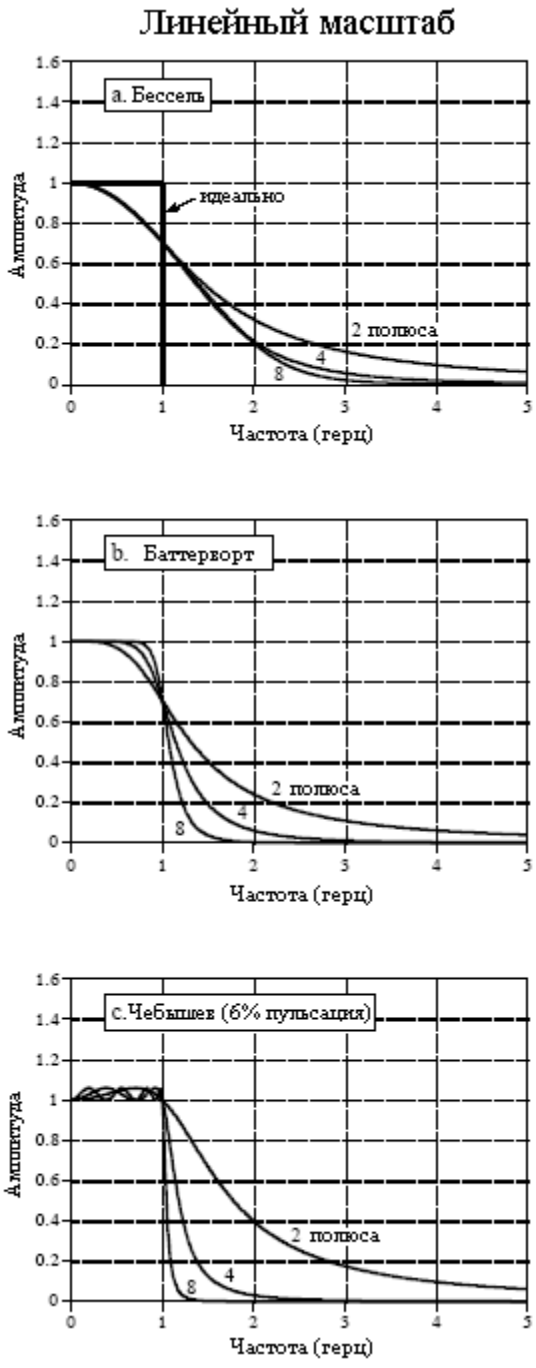


Рис. 3.12
Частотные характеристики фильтров
в *линейном* масштабе

Тонкий момент: коэффициент затухания 100 в этом примере, вероятно, достаточен, даже не смотря на то, что при 12 битах имеется 4096 ступеней. Как видно из рисунка 3.4, 5100 Гц будут совмещены с 4900 Гц, 6000 Гц будут совмещены с 4000 Гц и т.д. Не заботьтесь о том, каковы амплитуды сигналов между 5000 и 6300 герц, потому что они перемещаются в не используемую зону между 3700 герц и 5000 герц. Для того чтобы произошло перемещение частоты в область полосы пропускания фильтра (от 0 до 3,7 кГц), она должна лежать выше, чем 6300 герц, или быть в 1,7 раза больше частоты среза фильтра равной 3700 герц. Как показано на рис. 3.11c, затухание, обеспечиваемое 8 полюсным фильтром Чебышева на частоте в 1,7 раза больше частоты среза, около 1300, что гораздо больше соответствует поставленной задаче, чем 100, с которого мы начали

анализ. Мораль этой истории: *В большинстве систем полоса частот между приблизительно 0,4 и 0,5 частоты дискретизации - неиспользуемая свалка ослабляющей способности фильтра и сигналов с совмещенными именами*. Это прямой результат ограничений накладываемых аналоговыми фильтрами.

Частотная характеристика идеального низкочастотного фильтра плоская во всей полосе пропускания. С этой точки зрения все фильтры на рис. 3.11 выглядят великолепно, но только потому, что вертикальные оси показаны в *логарифмическом* масштабе. Совсем другая история, когда графики преобразуются к линейному масштабу по вертикальной оси, как это показана на рис. 3.12. Теперь **в полосе пропускания** у фильтра Чебышева может быть замечена **рябь** (волнистые изменения в амплитуде пропускаемых частот). Фактически фильтр Чебышева получает свое превосходное затухание с *позволения* этой ряби в полосе пропускания. Чем больше допускается количество пульсаций в полосе пропускания фильтра, тем более быстрое затухание может быть достигнуто. У всех фильтров Чебышева, рассчитанных с использованием коэффициентов, приведенных в табл. 3.1, пульсация амплитудно-частотной характеристики в полосе пропускания достигает 6% (0,5 dB), это хороший компромисс и типичный выбор. Подобная конструкция с **эллиптическим фильтром** допускает наличие пульсаций, как в полосе пропускания, так *и* в полосе подавления. Хотя проектировать эллиптические фильтры тяжелее, у них может быть достигнут лучший компромисс между крутизной спада и величиной пульсаций в полосе пропускания.

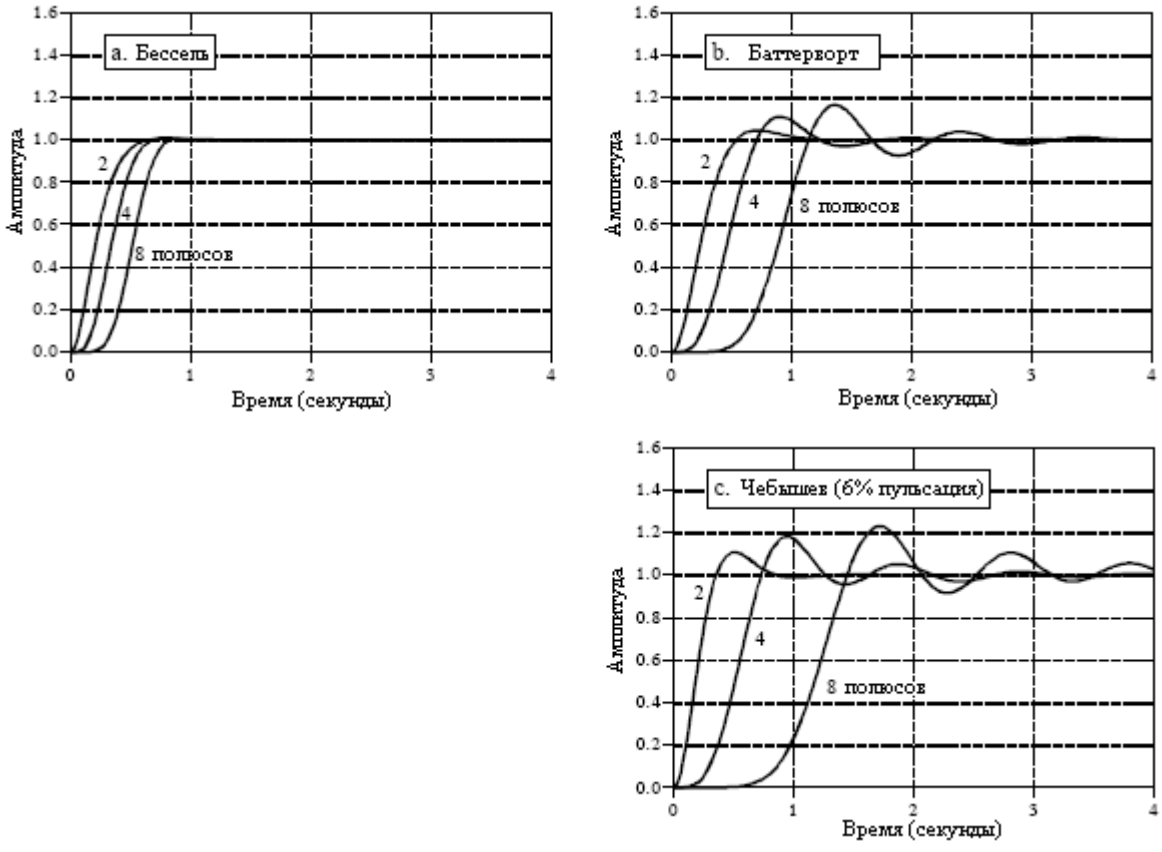


Рис 3.13 Переходные характеристики фильтров

Для сравнения, фильтры Баттерворта оптимизированы для обеспечения, насколько это возможно, наиболее крутого спада амплитудно-частотной характеристики при *отсутствии* пульсаций в полосе пропускания. Они обычно называются *максимально плоскими фильтрами* и идентичны фильтрам Чебышева, сконструированным при условии нулевых пульсаций в полосе пропускания. Фильтры Бесселя не имеют пульсаций в полосе

пропускания, но крутизна спада амплитудно-частотной характеристики у них хуже, чем у фильтров Баттерворта.

Последний параметр, который мы исследуем - это **переходная характеристика -** реакция фильтра на скачкообразное изменение значения на его входе от одного уровня до другого. На рис. 3.13 показаны переходные характеристики каждого из трех фильтров. Значения по горизонтальной оси показаны для фильтра с частотой среза 1 Гц, но их можно смасштабировать (обратно пропорционально) для более высоких частот среза. Например, для частоты среза 1000 герц переходная характеристика будет показана вместо *секунд* в *миллисекундах*. Фильтры Баттерворта и Чебышева **перерегулированы** (имеют выбросы – прим. перев.) и демонстрируют **звон** (колебания с медленно уменьшающейся амплитудой). Для сравнения, фильтры Бесселя не имеют ни одной из этих неприятных проблем.

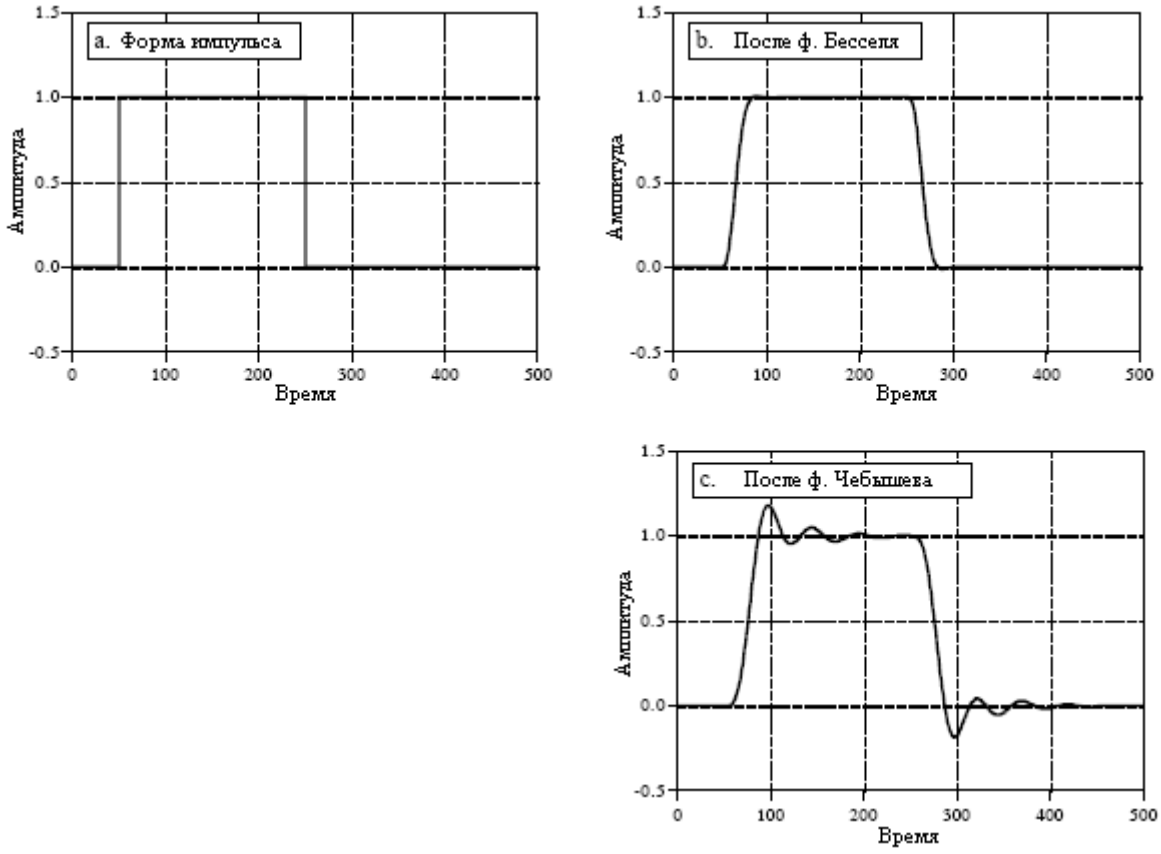


Рис 3.14 Отклик фильтров на прямоугольный импульс

Дальнейшая иллюстрация этой очень благоприятной характеристики фильтра Бесселя приведена на рис. 3.14. Рис. 3.14a показывает форму импульса, который может рассматриваться как резкий скачек вверх, за которым следует резкий скачек вниз. На рис. 3.14b и рис. 3.14c показано, как импульс такой формы будет выглядеть после прохождения его через фильтры Бесселя и Чебышева, соответственно. Например, если бы это был видео сигнал, искажения, внесенные фильтром Чебышева, стали бы разрушительными! Перерегулирование изменило бы яркость *краев* объектов по сравнению с их *центрами*. Еще хуже, левая сторона объектов выглядела бы светлой, в то время как правая сторона объектов будет выглядеть темной. Большое количество приложений просто не могут допустить плохого вида переходной характеристики. Вот здесь и сияют фильтры Бесселя; никакого перерегулирования и симметричные фронты импульса.

Выбор фильтра антисовмещения

Таблица 3.2 суммирует характеристики этих трех фильтров, показывая, как каждый из них оптимизирует конкретный параметр за счет всех остальных. Фильтр Чебышева оптимизирует *крутизну спада* амплитудно-частотной характеристики, фильтр Баттерворта оптимизирует *гладкость полосы пропускания*, а фильтр Бесселя оптимизирует *переходную характеристику*. Выбор фильтра антисовмещения почти полностью зависит от одной проблемы: *каким образом представлена информация в сигналах*, *которые Вы намереваетесь обрабатывать*. В то время как существует множество способов представления информации в аналоговом сигнале, только два способа типичны: **представление во временной области**, и **представление в частотной** области. Различие между ними в ЦОС – критическое, и в этой книге будет повторно встречающейся темой повсюду.

Таблица 3.2

	K_U (по постоянной сост.)	**Переходная характеристика**			**Частотная характеристика**		
		Перерегулирование	Время до установления переходного процесса к 1% (сек.)	Время до установления переходного процесса к 0,1% (сек.)	Пульсации в полосе пропускания	Частота для х100 ослабления (Гц)	Частота для х1000 ослабления (Гц)
Фильтр Бесселя							
2 полюса	*1,27*	*0,4%*	*0,60*	*1,12*	*0%*	*12,74*	*40,4*
4 полюса	*1.91*	*0.9%*	*0.66*	*1.20*	*0%*	*4.74*	*8.45*
6 полюсов	*2.87*	*0.7%*	*0.74*	*1.18*	*0%*	*3.65*	*5.43*
8 полюсов	*4.32*	*0.4%*	*0.80*	*1.16*	*0%*	*3.35*	*4.53*
Фильтр Баттерворта							
2 полюса	*1.59*	*4.3%*	*1.06*	*1.66*	*0%*	*10.0*	*31.6*
4 полюса	*2.58*	*10.9%*	*1.68*	*2.74*	*0%*	*3.17*	*5.62*
6 полюсов	*4.21*	*14.3%*	*2.74*	*3.92*	*0%*	*2.16*	*3.17*
8 полюсов	*6.84*	*16.4%*	*3.50*	*5.12*	*0%*	*1.78*	*2.38*
Фильтр Чебышева							
2 полюса	*1.84*	*10.8%*	*1.10*	*1.62*	*6%*	*12.33*	*38.9*
4 полюса	*4.21*	*18.2%*	*3.04*	*5.42*	*6%*	*2.59*	*4.47*
6 полюсов	*10.71*	*21.3%*	*5.86*	*10.4*	*6%*	*1.63*	*2.26*
8 полюсов	*28.58*	*23.0%*	*8.34*	*16.4*	*6%*	*1.34*	*1.66*

Примечание: Частота среза фильтров *1* Гц.

При *представлении информации в частотной области* информация содержится в *синусоидальных волнах*, которые для формирования сигнала объединяются. Великолепным примером этого являются звуковые сигналы. Когда кто-нибудь слышит музыку или речь, воспринятый звук зависит от присутствующих частот, а не от конкретной *формы* кривой сигнала. Это можно показать, пропуская звуковой сигнал через цепь, которая изменяет фазу различных синусоид, но сохраняет их частоту и амплитуду. На осциллографе результирующий сигнал *выглядит* совершенно по-другому, но *звуки* воспринимается точно также. Даже тогда, когда форма сигнала была значительно изменена, существенная информация осталась не поврежденной. Поскольку совмещение

имен переставляет не на свое место и при этом накладывает частотные компоненты друг на друга, это непосредственно разрушает информацию, кодируемую в частотной области. Как следствие, оцифровка таких сигналов обычно включает фильтр антисовмещения с круто падающей амплитудно-частотной характеристикой, такой как фильтр Чебышева, эллиптический фильтр или фильтр Баттерворта. А как же плохая переходная характеристика этих фильтров? Это не имеет значения; такой тип искажений не воздействует на закодированную информацию.

Напротив, при *представлении информации во временной области* для хранения информации используется *форма сигнала*. Например, врачи могут наблюдать электрическую активность сердца человека, прикладывая электроды к его груди и рукам (электрокардиограмма или ЭКГ). *Форма* кривой ЭКГ дает искомую информацию, такую как, когда во время сердцебиения сокращаются различные желудочки. Изображения являются другим примером сигналов этого типа. Точно так же, как в кривой, которая изменяется в зависимости от *времени*, в изображении, информация представлена в форме кривой, которая изменяется в зависимости от *расстояния*. Картины формируются из областей яркости и цвета, и того, как они соотносятся с другими областями яркости и цвета. Вы не смотрите на *Мону Лизу* и не говорите, "*Ой, какое интересное собрание синусоид*".

А вот и проблема: Теорема о дискретизации это анализ того, что происходит во время оцифровки сигнала в частотной области. Это позволяет идеально понять аналого-цифровое преобразование сигналов, содержащих свою информацию представленной в частотной области. Однако теорема о дискретизации мало помогает в понимании того, как должны быть оцифрованы сигналы, в которых информация представлена во временной области. Давайте посмотрим на это поближе.

Рисунок 3.15 иллюстрирует варианты оцифровки сигнала, в котором информация представлена во временной области. На рис. 3.15a приведен пример оцифровываемого аналогового сигнала. В этом случае, информация, которую мы хотим отследить - это *форма* прямоугольных импульсов. В данный пример сигнала, также включен короткий всплеск высокочастотной синусоидальной волны. Этот всплеск представляет широкополосный шум, интерференцию и подобный сор, который всегда появляется в аналоговых сигналах. На других рисунках показано, каким был бы сигнал при выборе различных вариантов фильтров антисовмещения: фильтра Чебышева; фильтра Бесселя и вообще без фильтра.

Важно понимать, что ни один из этих вариантов не позволяет восстановить исходный сигнал по дискретным данным. Это потому, что исходный сигнал неотъемлемо содержит частотные компоненты большие, чем половина частоты дискретизации. Поскольку эти частоты не могут существовать в цифровом сигнале, восстановленный сигнал также не сможет их содержать. Высокие частоты происходят от двух источников: (1) шума и интерференции, которые Вам хотелось бы устранить, и (2) крутых фронтов в форме сигнала, вероятно содержащих информацию, которую Вам хотелось бы сохранить.

Фильтр Чебышева, показанный на рис. 3.15b атакует проблему, агрессивно удаляя все высокочастотные составляющие. Это дает отфильтрованный аналоговый сигнал, который *может* быть подвергнут дискретизации и позже великолепно восстановлен. Однако восстановленный аналоговый сигнал будет идентичен *отфильтрованному сигналу*, а не *исходному сигналу*. Хотя при осуществлении дискретизации ничего не потеряно, форма сигнала была сильно искажена фильтром антисовмещения. Как показано на рис. 3.15b, лекарство оказалось хуже, чем болезнь! Не делайте этого!

Фильтр Бесселя, (рис. 3.15c), разработан как раз для этой проблемы. Его выходной сигнал сильно похож на исходный, только с нежным округлением фронтов. Регулированием частоты среза фильтра плавность фронтов можно обменять на устраненные высокочастотные компоненты в сигнале. Использование фильтра с большим количеством полюсов позволяет достичь *наилучшего* компромисса между этими двумя

параметрами. Обычно рекомендуют, сделать частоту среза равной одной четвертой частоты дискретизации. Это даст около двух отсчетов приходящихся на каждый из фронтов. Заметьте, что и фильтр Бесселя и фильтр Чебышева удалили всплеск высокочастотного шума, присутствовавшего в исходном сигнале.

Последний вариант, не использовать никакого фильтра вообще, как это показано на рис. 3.15d. Это имеет сильное преимущество, так как значение каждого отсчета идентично значению исходного аналогового сигнала. Другими словами, это дает великолепную крутизну фронтов; изменения в исходном сигнале мгновенно отражаются в цифровых данных. Неудобство в том, что сигнал может искажаться совмещением имен. Это принимает две различные формы. Первое, высокочастотная интерференция и шумы, такие как синусоидальный всплеск в примере, превратятся в отсчеты, не имеющие смысла, как показано на рис. 3.15d. То есть, любой высокочастотный шум, присутствующий в аналоговом сигнале, в цифровом сигнале будет появляться как шум совмещения имен. Но снижение шума и интерференции не является целью аналого-цифрового преобразования; до того момента, когда происходит оцифровка - зона ответственности аналоговой электроники. Может оказаться так, что для управления этой проблемой, перед устройством, осуществляющим дискретизацию, нужно будет поместить фильтр Бесселя. Однако это означает, что фильтр должен рассматриваться как часть аналоговой обработки, а не как что-то, что было сделано ради устройства, осуществляющего дискретизацию.

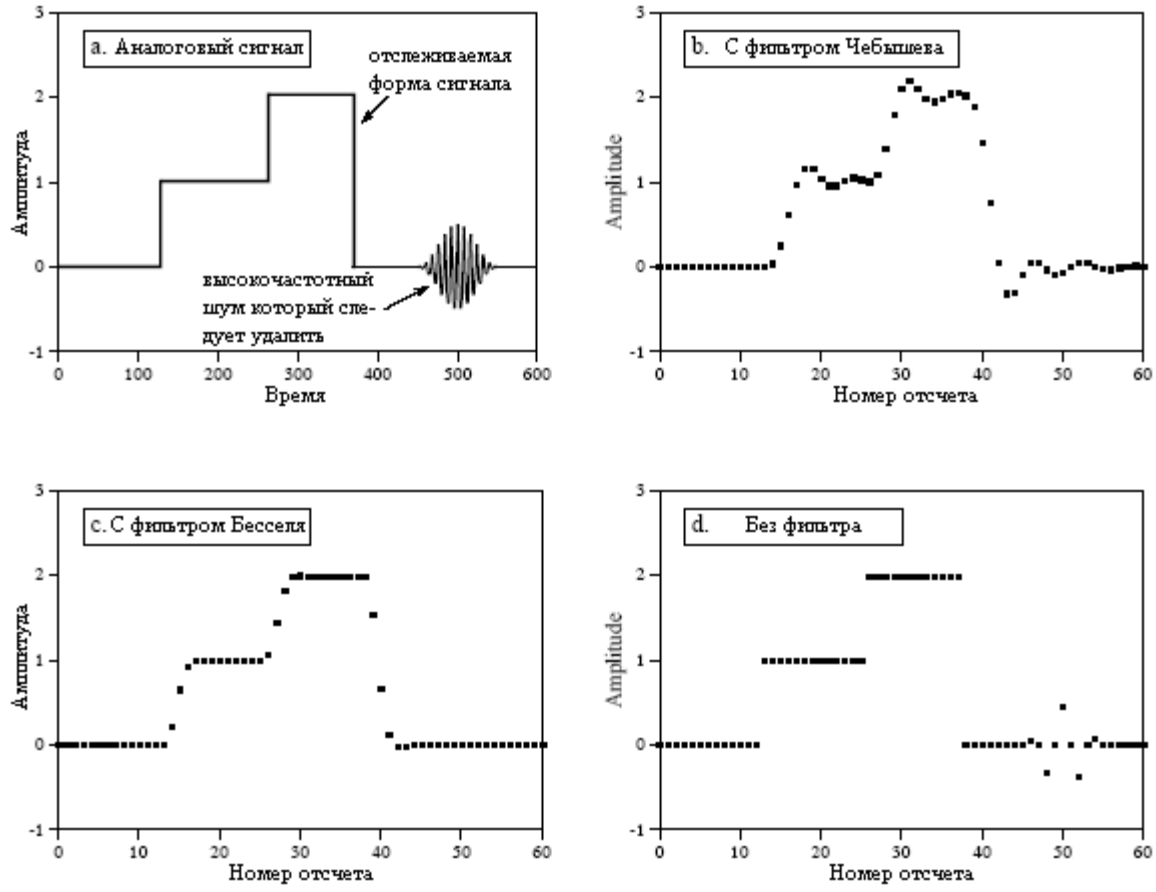


Рис. 3.15 Три варианта оцифровки сигнала, в котором информация представлена во временной области

Второе проявление совмещения имен более тонко. Когда в аналоговом сигнале происходит некоторое событие (такое как фронт), изменения в цифровом сигнале, показанном на рис. 3.15d, обнаруживаются лишь в *следующем* отсчете. В цифровых

данных нет никакой информации показывающей, что произошло *между* отсчетами. Теперь, для этой проблемы сравните *отсутствие использования фильтра* с *использованием фильтра Бесселя*. Например, представьте нарисованные прямые линии между отсчетами на рис. 3.15c. Момент времени, когда эти построенные линии пересекут половину шага амплитуды между отсчетами, дает *промежуточную* оценку того, когда возник фронт в аналоговом сигнале. Эта информация полностью теряется, если не используется никакой фильтр. Для того чтобы оценить, как это повлияет на Вашу конкретную ситуацию, Вам не нужна никакая гипотетическая теорема, нужно просто хорошее понимание того, что Вы собираетесь делать с данными, когда они будут получены.

Мультичастотное преобразование данных

В электронике существует сильная тенденция по замене *аналоговых схем цифровыми алгоритмами*. Превосходным примером этому служит преобразование данных. Рассмотрим проект цифрового регистратора голоса, системы, которая будет оцифровывать сигнал голоса, хранить данные в цифровой форме и позже восстанавливать сигнал для воспроизведения. Для воссоздания понятной речи система должна охватывать частоты между приблизительно 100 и 3000 герц. Однако, аналоговый сигнал, на выходе микрофона содержит также значительно более высокие частоты, скажем до 40 кГц. При лобовом подходе к решению этой задачи следует пропустить аналоговый сигнал через восьми полюсный низкочастотный фильтр Чебышева с частотой среза 3 кГц и затем произвести дискретизацию с частотой 8 кГц. С другой стороны, ЦАП восстанавливает аналоговый 8 кГц сигнал с хранением нулевого порядка. Для воспроизведения заключительного сигнала голоса используется другой фильтр Чебышева с частотой среза 3 кГц.

Имеется множество полезных выгод в осуществлении дискретизации *быстрее*, чем предлагает этот прямой анализ. Например, представьте повторное проектирование цифрового регистратора голоса, использующего частоту дискретизации 64 кГц. На фильтр антисовмещения теперь возложена более легкая задача: пропустить все частоты ниже 3 кГц при подавлении всех частот лежащих выше, чем 32 кГц. Подобное упрощение происходит и с восстанавливающим фильтром. Короче говоря, высокая частота дискретизации позволяет заменить восьми полюсные фильтры простой резисторно - конденсаторной (RC) цепью. Проблема в том, что цифровая система теперь завалена данными из-за высокой частоты дискретизации.

Следующий уровень сложности включает **мультичастотную** технику, использующую больше чем одну частоту дискретизации в одной и той же системе. Ее работа подобна описанной в примере цифрового регистратора голоса. Первое, звуковой сигнал пропускается через простой низкочастотный RC фильтр и подвергается дискретизации с частотой 64 кГц. Результирующие цифровые данные содержат желаемую полосу голосовых частот между 100 и 3000 герц, но также содержат неиспользуемую полосу частот между 3 кГц и 32 кГц. Второе, *программным* способом с помощью *цифрового* низкочастотного фильтра с частотой среза 3 кГц, эти неиспользуемые частоты удаляются. Третье, производится повторная дискретизация цифрового сигнала от 64 кГц до 8 кГц путем простого отбрасывания каждых семи из восьми отсчетов, эта процедура называется **децимацией** (в древности это казнь одного из десяти человек по жребию, позже так называли наказание каждого десятого в случае ненахождения виновного – прим. перев.). Результирующие цифровые данные эквивалентны данным, полученным после агрессивной аналоговой фильтрации и непосредственной дискретизации с частотой 8 кГц.

Мультичастотная техника может быть также использована в выходной части системы нашего примера. Данные, полученные в результате дискретизации с частотой 8

кГц, извлекаются из памяти и преобразуются к частоте дискретизации 64 кГц, эта процедура называется **интерполяцией**. Интерполяция заключается в том, что между каждыми отсчетами, извлекаемыми из памяти, вставляются семь нулей. Результирующий сигнал представляет собой *импульсную последовательность*, содержащую желаемую полосу голосовых частот между 100 и 3000 герц, плюс дублированные спектры между 3 кГц и 32 кГц. Для того чтобы понять, что это справедливо, вернитесь назад к рис. 3.6a и рис. 3.6b. Затем все что выше 3 кГц удаляется с помощью *цифрового* низкочастотного фильтра. После преобразования полученных данных с помощью ЦАП в аналоговый сигнал, простая RC цепь - это все что нужно для воспроизведения конечного голосового сигнала.

Мультичастотное преобразование данных очень ценно по двум причинам: (1) оно заменяет аналоговые компоненты программным обеспечением, это ясное экономическое преимущество в серийном производстве, и (2) оно может обеспечить более высокий уровень работы в критических приложениях. Например, звуковые системы компакт-дисков используют методы этого типа для достижения возможно лучшего качества звука. Улучшенное качество работы является результатом замены аналоговых компонент (точность 1 %) цифровыми алгоритмами (точность 0.0001 % от ошибки округления). Как будет показано в следующих главах, цифровые фильтры в ключевых областях превосходят аналоговые фильтры в *сотни раз*.

Однобитовое преобразование данных

Популярной техникой в телекоммуникации и высококачественном воспроизведении музыки является **однобитовое аналого-цифровое** и **цифро-аналоговое преобразования**. Это мультичастотная техника, где более высокая частота дискретизации является платой за более низкое число битов. В пределе для каждого отсчета нужен только один бит. Несмотря на то, что существует большое число схемных конфигураций, большинство из них базируется на использовании **дельта-модуляции**. Для того чтобы Вы получили представление об этой области, будут рассмотрены три примера схем. Все данные схемы разработаны в виде интегральной микросхемы, так что не переживайте о том, куда войдут все эти индивидуальные транзисторы и операционные усилители. Никто не собирается просить Вас построить одну из этих базовых схем на дискретных компонентах.

На рис. 3.16 показана блок схема типичного дельта-модулятора. На аналоговый вход дельта-модулятора подается голосовой сигнал с амплитудой в несколько вольт, в то время как его выходным сигналом является поток цифровых единиц и нулей. Компаратор принимает решение, где в данный момент времени напряжение большее: в поступающем аналоговом сигнале или на конденсаторе. Это решение в виде цифровой единицы или нуля прикладывается к входу триггера-защелки. При каждом тактовом импульсе, обычно с частотой следования несколько сотен килогерц, защелка передает на свой выход цифровое состояние, появившееся на ее входе. Эта защелка обеспечивает синхронизацию выходного сигнала с тактовой частотой, определяя, таким образом, скорость дискретизации, то есть скорость, с которой может модифицироваться 1 битовый выход.

Сигнал с цифрового выхода с помощью петли обратной связи используется для ведения электронного ключа. Если на цифровом выходе *единица*, ключ соединяет конденсатор с *инжектором положительного заряда*. Это очень неточный термин для цепи, которая *увеличивает* напряжение на конденсаторе на фиксированную величину, скажем на 1 милливольт за один период тактовой частоты. Такая цепь может быть ничем иным, как резистором, присоединенным к мощному источнику положительного напряжения. Если на цифровом выходе *ноль*, ключ соединяет конденсатор с *инжектором отрицательного заряда*. Это *уменьшает* напряжение на конденсаторе на такую же фиксированную величину.

Рис. 3.17 иллюстрирует сигнал, вырабатываемый этой схемой. В момент времени равный нулю, как на аналоговом входе, так и на конденсаторе напряжение равно нулю. Как показано на рис. 3.17a, внезапно, на восьмом такте входной сигнал увеличивается до 9,5 вольт. Поскольку теперь входной сигнал более положителен чем напряжение на конденсаторе, цифровой выходной сигнал принимает значение *единицы*, как это показано на рис. 3.17b. Это приводит к тому, что ключ подключается к инжектору положительного заряда, и напряжение на конденсаторе в каждом тактовом периоде начинает увеличиваться на небольшую величину. Хотя на рис. 3.17a показано приращение в 1 вольт за тактовый период, это сделано только для иллюстрации, и более типичным значением приращения является значение в 1 милливольт. Такое ступенчатое увеличение напряжения на конденсаторе продолжается до тех пор, пока не будет превышено значение напряжения входного сигнала. Здесь система достигает равновесия, и выходной цифровой сигнал начинает испытывать колебания между единицей и нулем, вызывая на конденсаторе изменения напряжения между 9 и 10 вольтами. Обратная связь схемы вынуждает напряжение на конденсаторе отслеживать напряжение входного сигнала в такой манере. Если изменения входного сигнала могут происходить очень быстро, то напряжение на конденсаторе всегда изменяется с постоянной скоростью и до тех пор, пока не будет достигнуто соответствие. Эта постоянная скорость изменения называется **скоростью нарастания**, так же, как и в других электронных устройствах наподобие операционных усилителей.

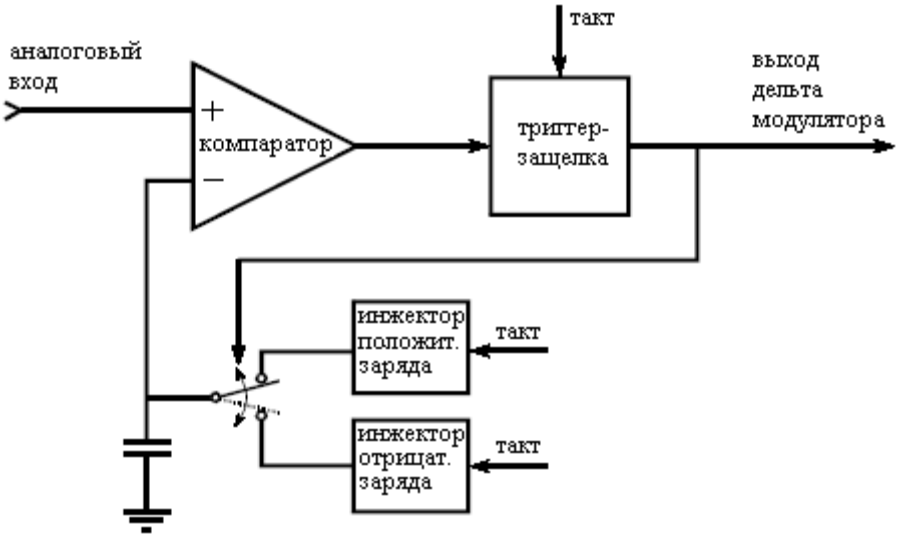


Рис. 3.16 Блок-схема дельта-модулятора

А сейчас рассмотрим характеристики дельта-модулированного сигнала. Если значение напряжения на аналоговом входе *увеличивается*, выходной сигнал будет содержать большее количество единиц, чем нулей. Аналогично, если значение напряжения на аналоговом входе *уменьшается*, выходной сигнал будет содержать большее количество нулей, чем единиц. Если значение напряжения на аналоговом входе не меняется, в цифровом выходном сигнале будут одинаково чередоваться нули и единицы. Выражаясь в более обобщенном виде, относительное число единиц по сравнению с нулями прямо пропорционально *наклону* (производной) входного аналогового сигнала.

Эта схема является недорогим способом преобразования аналогового сигнала в последовательный поток нулей и единиц для цифровой передачи данных или их хранения. Особенно привлекательной чертой здесь является то, что все биты, имеют одно и тоже назначение, в отличие от обычного последовательного формата: *стартовый бит*,

младший значащий разряд, …, *старший значащий разряд*, *стоповый бит*. Схема в приемнике идентична цепи обратной связи в передающей схеме. То же, что происходит с отслеживающим сигнал на аналоговом входе напряжением на конденсаторе в передающей схеме, то же происходит и с напряжением на конденсаторе в схеме приемника. То есть напряжение на конденсаторе, представленное на рис. 3.17a, иллюстрирует еще и процесс появления восстановленного сигнала.

Критическим ограничением этой схемы является неизбежный компромисс между (1) максимальной скоростью нарастания, (2) величиной шага квантования и (3) скоростью передачи данных. В частности, если максимальная скорость нарастания и величина шага квантования отрегулированы до приемлемых значений для передачи звуковых сигналов, скорость передачи данных должна находиться в конце мегагерцового диапазона. Это слишком высоко, чтобы представлять коммерческий интерес. Например, обычная дискретизация звукового сигнала требует только около 64000 битов в секунду.

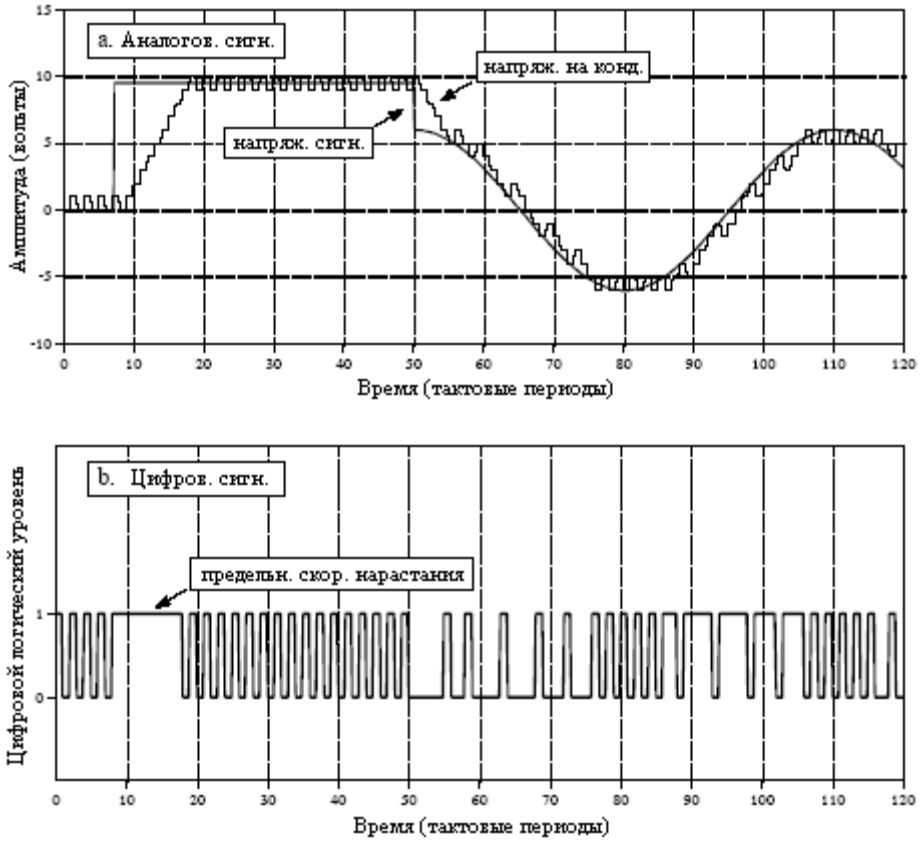


Рис. 3.17 Пример сигналов с дельта-модулятора, показанного на рис. 3.17

Решение этой проблемы показывается на рис. 3.18, дельта-модулятор с непрерывно изменяемым наклоном (**CVSD**), техника, реализованная в микросхеме MC3518 семейства фирмы Motorola. В этом подходе, тактовая частота и величина шага квантования установлены до некоторого приемлемого значения, скажем 30 кГц, и 2000 уровней. Это приводит к отвратительной скорости нарастания, которую Вы корректируете с помощью дополнительных схемотехнических приемов. В процессе работы, сдвиговый регистр непрерывно следит за последними четырьмя битами, вырабатываемыми системой. Если цепь находится в условиях предельной скорости нарастания, все последние четыре бита будут единицами (положительный наклон) или нулями (отрицательный наклон). Логическая схема обнаруживает эту ситуацию и вырабатывает аналоговый сигнал, который увеличивает уровень заряда, производимого инжекторами заряда. Это повышает

скорость нарастания за счет увеличения величины шага напряжения, прикладываемого к конденсатору.

Между инжекторами заряда и логической схемой обычно включается фильтр. Он позволяет сделать величину шага зависящей от того, насколько долго схема находилась в условиях предельной скорости нарастания. Чем больше схема находится в условиях предельной скорости нарастания, тем больше и больше продолжает становиться величина шага. Такой фильтр часто называется *силлабическим* (слоговым - прим. перев.) *фильтром*, так как его характеристики зависят от средней длины слогов, составляющих речь. С соответствующей оптимизацией (из технического описания изготовителя микросхем - Вам не нужно делать оптимизацию) скорости передачи данных от 16 до 32 кГц позволяют получить приемлемое качество речи. Непрерывно изменяющаяся величина шага делает цифровые данные трудно понимаемыми, но к счастью, Вам это не нужно. В приемнике, аналоговый сигнал восстанавливается с помощью встроенного силлабического фильтра идентичного фильтру в передающей схеме. Если два фильтра согласованы, CVSD модуляция дает небольшие искажения. Вероятно, CVSD является самым легким способом передачи сигнала голоса в цифровой форме.

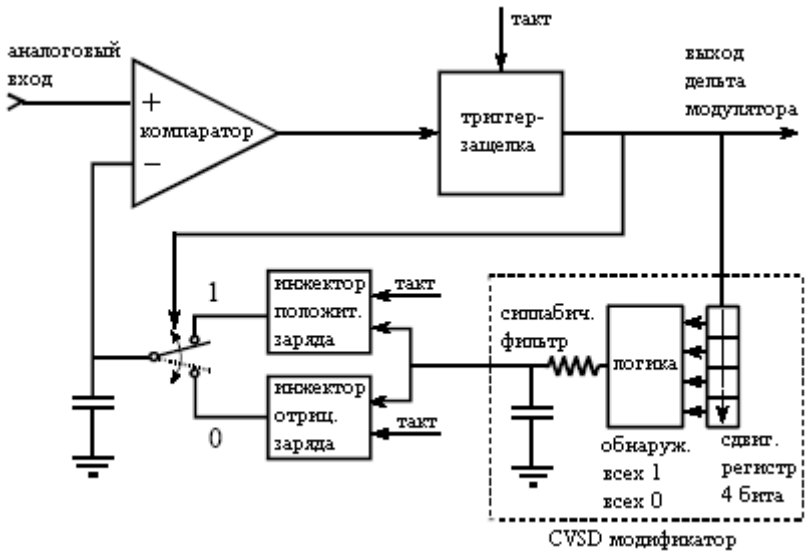


Рис. 3.18 Блок-схема CVSD модулятора

В то время как CVSD модуляция хороша для кодирования голосовых сигналов, она не может использоваться для общих целей аналого-цифрового преобразования. Даже если Вы обойдете тот факт, что цифровые данные связаны с *производной* входного сигнала, *изменяющаяся величина шага* будет вносить беспорядок в восстановление сигнала. Кроме того, в цифровых данных обычно не представлен уровень постоянной составляющей аналогового сигнала.

Дельта-сигма преобразователь, показанный на рис. 3.19, устраняет эти проблемы, разумно сочетая аналоговую электронику с алгоритмами ЦОС. Обратите внимание, что напряжение на конденсаторе теперь сравнивается потенциалом земли. Петля обратной связи изменена таким образом, чтобы напряжение на конденсаторе *уменьшалось*, когда на выходе схемы логическая *единица*, и *увеличивалось*, когда на выходе схемы логический *ноль*. Увеличение и уменьшение напряжения входного сигнала вызывает повышение и понижение напряжения на конденсаторе. Эти изменения напряжения обнаруживаются компаратором и приводят к инжекции *противодействующего* заряда, стремящегося удержать напряжение на конденсаторе близким к нулю вольт.

Если напряжение на входе положительно, выходной сигнал будет содержать большее количество единиц, чем нулей. Избыточное количество единиц необходимо для

генерирования *отрицательного* заряда, который гасится *положительным* входным сигналом. Аналогично, если входное напряжение отрицательно, выходной сигнал будет содержать большее количество нулей, чем единиц, обеспечивая инжекцию только положительного заряда. Если входной сигнал равен нулю вольт, на выходе будет появляться одинаковое количество единиц и нулей, в результате суммарная инжекция заряда будет также равна нулю.

Относительное количество единиц и нулей в выходном сигнале, теперь связано с уровнем входного напряжения, а не с его наклоном, как в предыдущей схеме. Это значительно проще. Например, Вы можете сформировать 12 битовое аналого-цифровое преобразование, подавая цифровой выходной сигнал на счетчик и подсчитывая количество *единиц* в течение 4096 тактовых периодов. Результирующий цифровой код счетчика 4095 будет соответствовать величине максимально возможного положительного входного напряжения. Аналогично, цифровой код 0 будет соответствовать величине максимально возможного отрицательного входного напряжения, а код 2048 будет соответствовать нулевому входному напряжению. Это также показывает происхождение названия *дельта-сигма*: дельта-модуляция сопровождается суммированием (сигма).

Единицы и нули, произведенные этим типом дельта-модулятора очень легко обратно преобразовать в аналоговый сигнал. Все, что требуется - это аналоговый низкочастотный фильтр, который может быть таким же простым, как одна единственная RC цепочка. Высокие и низкие напряжения, соответствующие цифровым единицам и нулям, в среднем, формируют на выходе правильное аналоговое напряжение. Например, предположим, что единицы и нули представлены соответственно 5 вольтами и 0 вольт. Если 80 % битов в потоке данных *единицы*, а 20 % - *нули*, на выходе фильтра нижних частот будет 4 вольта.

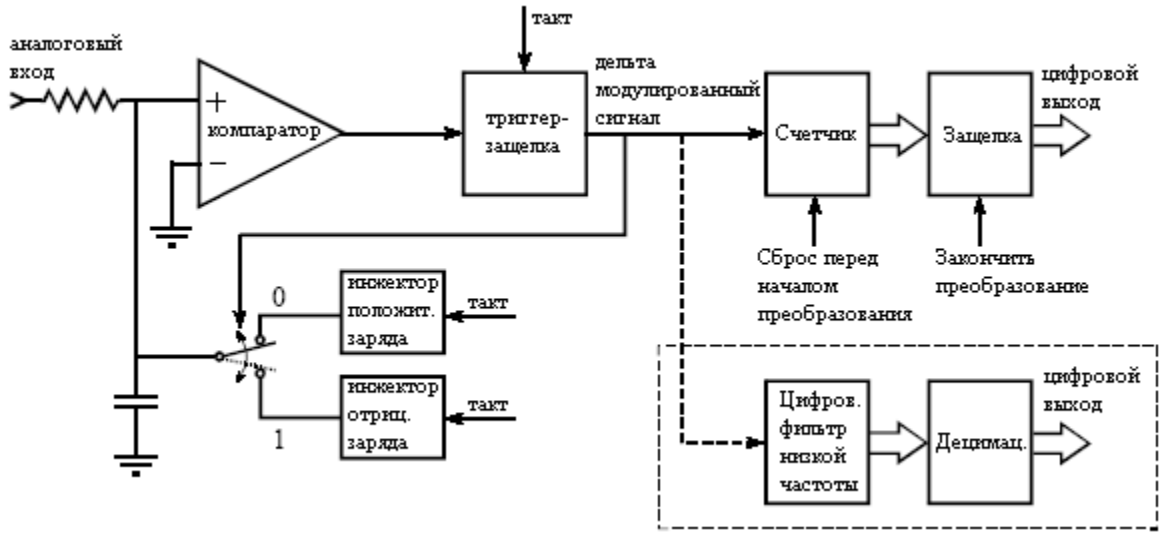


Рис. 3.19 Блок-схема дельта-сигма АЦП

Этот метод преобразования однобитового потока данных обратно в первоначальную форму волны, важен по нескольким причинам. Первое, он описывает гладкий способ замены счетчика в схеме дельта-сигма АЦП. Вместо простого подсчета импульсов от дельта-модулятора двоичный сигнал пропускают через *цифровой* фильтр нижних частот, а затем проводят *децимацию*, чтобы снизить частоту дискретизации. Например, эта процедура могла бы начинаться заменой каждой единицы и каждого нуля в цифровом потоке на 12 битовые отсчеты; единицы станут значением 4095, в то время как нули станут значением 0. Обрабатывая этот сигнал с помощью цифрового фильтра низкой частоты, будет получена оцифрованная версия первоначального сигнала, так же, как и при

восстановлении исходного сигнала с помощью аналогового фильтра низкой частоты. Затем с помощью децимации осуществляется снижение частоты дискретизации за счет отбрасывания большинства из отсчетов. Это приводит к цифровому эквиваленту прямой дискретизации первоначального сигнала.

Такой подход используется во многих коммерческих АЦП для оцифровки голоса и других звуковых сигналов. Примером является АЦП ADC16071 фирмы National Semiconductor, который обеспечивает 16 битовое аналого-цифровое преобразование с частотой дискретизации до 192 кГц. При частоте дискретизации в 100 кГц дельта-модулятор работает с тактовой частотой 6,4 МГц. Низкочастотный фильтр - это 246 точечный КИХ-фильтр такой, как описан в Главе 16. Он удаляет все частоты в цифровых данных более чем 50 кГц, 1/2 возможной частоты дискретизации. Концептуально это может рассматриваться как формирование цифрового сигнала в 6,4 МГц, каждый отсчет которого представлен 16 битами. Затем сигнал подвергается децимации с 6,4 МГц до 100 кГц, выполняемой за счет удаления каждых 63 из 64 отсчетов. В действительности внутри этого микроэлектронного прибора происходит значительно больше, чем затронуто в этой простой дискуссии.

Дельта-сигма преобразователи могут также быть использованы для осуществления цифро-аналогового преобразования голосовых и звуковых сигналов. Цифровой сигнал извлекается из памяти и преобразуется в дельта-модулированный поток единиц и нулей. Как упоминалось выше, этот однобитовый сигнал легко может быть преобразован в восстановленный аналоговый сигнал с помощью простого аналогового фильтра низкой частоты. Как и с фильтром, предотвращающим совмещение имен, обычно для этого требуется только одна единственная RC цепь. Это в связи с тем, что большая часть фильтрации выполнена высококачественными цифровыми фильтрами.

У дельта-сигма АЦП есть несколько причуд, которые ограничивают их использование в некоторых конкретных приложениях. Например, их входы очень сложно мультиплексировать. Когда происходит переключение входа с одного сигнала на другой, надлежащее функционирование преобразователя не может установиться до тех пор, пока цифровой фильтр не сможет очиститься от данных предыдущего сигнала. Дельта-сигма преобразователи ограничены также в другом отношении: Вы не знаете точно, *когда* был принят каждый отсчет. Каждый полученный отсчет является смесью информации одиночных битов сегмента входного сигнала. Это не является проблемой для сигналов, содержащих информацию в частотной области, таких как звук, но это значительное ограничение для сигналов, содержащих информацию во временной области. Для понимания вида формы волны сигнала, Вам часто нужно знать точный момент снятия каждого отсчета. Наконец, большинство этих устройств определенно разработано для приложений связанных с обработкой звука, и их рабочие характеристики соответственно нормированы. Например, АЦП - точность 16 бит, используемый для голосовых сигналов, не обязательно подразумевает, что каждый отсчет имеет 16 бит точности. Гораздо более вероятно, что изготовитель утверждает, что *сигналы голоса* могут быть оцифрованы до 16 бит *динамического диапазона*. Не надейтесь получить все 16 бит полезной информации от этого устройства для универсальных систем сбора данных.

В то время как эти объяснения и примеры обеспечивают представление об однобитовых АЦП и ЦАП, должно быть подчеркнуто, что они являются сильно упрощенными описаниями и сложнейшей ЦОС, и технологии интегральных схем. Вы же не будете ожидать, что производители расскажут своим *конкурентам* все внутреннее устройство своих микросхем, так что не ждите, что они расскажут это и *Вам*.

ГЛАВА
4

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЦОС

Приложения ЦОС обычно программируются на тех же языках, что и другие научные и технические задачи, таких как: C, BASIC и ассемблер. Мощность и многосторонность C делают его языком выбора ученых, занимающихся компьютерной наукой и других профессиональных программистов. С другой стороны, простота BASIC делает его идеальным для ученых и инженеров, которые посещают мир программирования от случая к случаю. Независимо от языка, используемого Вами, большинство важного, выпущенного в свет программного обеспечения ЦОС, уходит корнями гораздо глубже, в царство, в котором вращаются единицы и нули. Оно включает в себя такие темы как: представление чисел с помощью двоичных кодов, ошибка округления в компьютерной арифметике, скорость вычислений различных типов процессоров и т.д. Эта глава посвящена тем вещам, которые Вам просто необходимо делать на *высоком уровне*, чтобы не быть растоптанными на *низком уровне*, уровне внутренней работы вашего компьютера.

Компьютерные числа

Цифровые компьютеры могут очень искусно хранить и запоминать числа, но, к сожалению, этот процесс не обходится без ошибок. Например, Вы даете своему компьютеру команду сохранить число: 1,41421356. Компьютер пытается сделать все, что он *может*, сохраняя самое близкое число из тех, которые он только может представить: 1,41421354. В некоторых случаях такое представление содержит ничего незначащую ошибку, в то время как в других случаях эта ошибка является губительной. В качестве другой иллюстрации рассмотрим классическую вычислительную ошибку, следующую из сложения двух чисел с очень разными значениями, например 1 и 0,00000001. Мы хотели бы получить ответ равный 1,00000001, но ответ компьютера *1*. Понимание того, как компьютеры хранят числа и манипулируют ими, позволит Вам прогнозировать и корректировать подобные проблемы *до того*, как ваша программа выдаст бессмысленные данные.

Подобные проблемы возникают из-за того, что для хранения каждого числа используется фиксированное число разрядов, обычно 8, 16, 32 или 64. Рассмотрим в качестве примера случай, когда для хранения значения переменной используется восемь битов. Поскольку имеется $2^8 = 256$ различных комбинаций, переменная может принимать только 256 различных значений. Это фундаментальное ограничение ситуации, и с ним мы ничего не можем поделать. Единственная вещь, которой мы *можем* управлять, это объявить, какую величину представляет каждое сочетание битов. В простейших случаях 256 комбинаций битов могут представлять целые от 0 до 255, от 1 до 256, от –127 до +128 и т.д. В более необычной схеме 256 комбинаций битов могут представлять числа в экспоненциальной форме: 1, 10, 100, 1000, …, 10^{254}, 10^{255}. Каждый, кто обращается к данным, должен понимать, какую величину представляет та или иная комбинация битов. Такое понимание обычно обеспечивается алгоритмом или формулой для прямого и обратного преобразования между представляемой величиной и соответствующей комбинацией битов.

Несмотря на то, что возможно большое количество различных схем кодирования, обычными стали только два основных формата: с *фиксированной запятой* (называемый также форматом целых чисел) и с *плавающей запятой* (называемый также форматом действительных чисел). В программах этой книги на языке BASIC переменные с фиксированной запятой отмечаются символом %, входящим в ее имя в качестве последней буквы, например: I%, N%, SUM% и т.д. Все другие переменные являются переменными с плавающей запятой, например: X, Y, MEAN и т.д. Когда Вы будете оценивать форматы, представленные на нескольких следующих страницах, постарайтесь понимать их в терминах их **диапазона** (самых больших и самых маленьких чисел, которые могут быть представлены ими) и их **точности** (размера интервала между соседними числами).

Фиксированная запятая (Целые)

Представление с фиксированной запятой используется для хранения *целых* положительных и целых отрицательных чисел: …, -3, -2, -1, 0, 1, 2, 3, … . Программы на языках высокого уровня, таких как C и BASIC, для хранения каждого целого обычно предоставляют 16 битов. В простейшем случае 2^{16}=65536 возможных комбинаций битов, которые присваиваются числам от 0 до 65535. Это называется, форматом **целого числа без знака**, его упрощенный пример показан на рис. 4.1 (здесь для каждого числа используются только 4 бита). Преобразование между комбинацией битов и представляемым числом - ничто иное, как замена основания 2 (двоичное) на основание 10 (десятичное). Недостатком формата целого числа без знака является то, что здесь не могут быть представлены отрицательные числа.

Двоичный формат **со смещением** подобен формату целого числа без знака за исключением того, что десятичные величины с целью учета отрицательных чисел сдвинуты. В четырех битовом примере на рис. 4.1 десятичные числа смещены на *семь*, что приводит в 4 битовых комбинациях к соответствующим целым числам от –7 до 8. Используя точно в такой же манере в качестве смещения 32767, 16 битовое представление даст диапазон целых чисел между –32767 и 32768. Двоичный формат со смещением не является стандартизированным форматом, и Вы можете встретить другие используемые смещения подобные рассмотренному 32767. Наиболее важным использованием двоичного формата со смещением являются аналого-цифровое и цифро-аналоговое преобразования. Например, для 12 битового преобразования диапазон входных напряжений от –5 до +5 вольт мог бы быть приведен к цифровым кодам от 0 до 4095.

Формат **целого числа со знаком** является другим простым способом представления отрицательных целых чисел. Самый левый бит называется **знаковым разрядом**, он равен *нулю* для положительных чисел и *единице* для отрицательных чисел. Другие биты являются стандартным двоичным представлением абсолютного значения числа. В этом формате одна комбинация битов тратится впустую, поскольку здесь существует два представления нуля: 0000 (положительный ноль) и 1000 (отрицательный ноль). Такая схема кодирования чисел приводит к 16 битовым числам в диапазоне от –32767 до 32767.

Три первых формата являются концептуально простыми, но трудно осуществимыми в аппаратных средствах ЭВМ. Не забывайте о том, что когда в компьютерную программу вводится A=B+C, инженеры, разрабатывающие аппаратную часть компьютера, должны определить, как реализовать объединение комбинации битов, представляющих B, с комбинацией битов, представляющих C, для формирования комбинации битов, представляющих A.

Двоичный дополнительный формат - любимый формат инженеров, разрабатывающих аппаратную часть компьютера, он является форматом, в котором в компьютере обычно представляются целые числа. Для того чтобы понять такую кодовую

комбинацию, рассмотрим сначала десятичный ноль на рис. 4.1, соответствующий двоичному нулю 0000. Поскольку мы считаем вверх, десятичное число является просто двоичным эквивалентом (0 = 0000, 1 = 0001, 2 = 0010, 3 = 0011 и т.д.). Теперь вспомните, что эти четыре бита, хранятся в регистре, состоящем из 4 триггеров. Если мы снова начнем с 0000 и примемся вычитать, цифровые аппаратные средства ЭВМ автоматически высчитывают дополнение до двойки: от 0 до 0000, от -1 до 1111, от -2 до 1110, от -3 до 1101 и т.д. Это аналогично счетчику пробега в новом автомобиле. Если двигаться вперед, он изменяется: 00000, 00001, 00002, 00003, и так далее. Когда двигаешься назад, изменения счетчика пробега: 00000, 99999, 99998, 99997 и т.д.

ЦЕЛОЕ ЧИСЛО БЕЗ ЗНАКА		ДВОИЧНЫЙ СО СМЕЩЕНИЕМ		ЦЕЛОЕ ЧИСЛО СО ЗНАКОМ		ДВОИЧНЫЙ ДОПОЛНИТЕЛЬНЫЙ	
Десятичн.	Комб. бит	Десятичн.	Комб. бит	Десятичн.	Комб. бит	Десятичн.	Комб. бит
15	1111	8	1111	7	0111	7	0111
14	1110	7	1110	6	0110	6	0110
13	1101	6	1101	5	0101	5	0101
12	1100	5	1100	4	0100	4	0100
11	1011	4	1011	3	0011	3	0011
10	1010	3	1010	2	0010	2	0010
9	1001	2	1001	1	0001	1	0001
8	1000	1	1000	0	0000	0	0000
7	0111	0	0111	0	1000	-1	1111
6	0110	-1	0110	-1	1001	-2	1110
5	0101	-2	0101	-2	1010	-3	1101
4	0100	-3	0100	-3	1011	-4	1100
3	0011	-4	0011	-4	1100	-5	1011
2	0010	-5	0010	-5	1101	-6	1010
1	0001	-6	0001	-6	1110	-7	1001
0	0000	-7	0000	-7	1111	-8	1000
16 битовый диапазон: от 0 до 65535		16 битовый диапазон: от -32767 до 32768		16 битовый диапазон: от -32767 до 32767		16 битовый диапазон: от -32768 до 32767	

Рис. 4.1 Обычные форматы представления чисел с фиксированной запятой (целые)

При использовании 16 битов, с помощью двоичного дополнительного формата можно представить числа от -32768 до 32767. Самый левый бит равен 0, если число положительно или ноль, и 1, если число отрицательно. Следовательно, самый левый бит называется **знаковым разрядом**, точно также как и в формате целого числа со знаком. Преобразование между десятичным числом и двоичным дополнительным форматом для положительных чисел прямое, простое преобразование десятичного числа в двоичное. Для отрицательных чисел очень часто используется следующий алгоритм: (1) Взять абсолютное значение десятичного числа, (2) преобразовать его в двоичное число, (3) дополнить все биты (единицы заменить нулями, а нули единицами), (4) к двоичному числу добавить единицу. Например: -5 → 5 → 0101 → 1010 → 1011. Двоичный дополнительный формат труден для человека, но прост для цифровой электроники.

Плавающая запятая (Действительные числа)

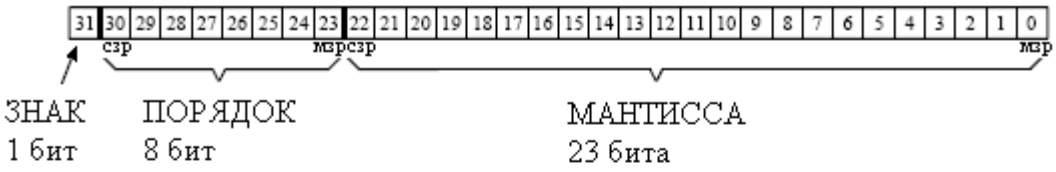
Схема кодирования чисел с плавающей запятой более сложна, чем для чисел с фиксированной запятой. Основная идея здесь та же, что используется в научной системе обозначений, где **мантисса** умножается на десятку, увеличенную в число раз кратное некоторому **показателю степени**. Например, $5{,}4321\times10^{6}$, где 5,4321 – *мантисса*, а 6 - *показатель степени*. Научная система обозначений исключительна при представлении очень больших и очень маленьких чисел. Например, $1{,}2\times10^{50}$ - число атомов на земле, или $2{,}6\times10^{-23}$ – расстояние, которое черепаха проползет за одну секунду, в сравнении с

диаметром нашей галактики. Обратите внимание, что числа, представленные в научной системе обозначений нормализованы таким образом, что слева от запятой находится только одна единственная цифра отличная от нуля. Это достигается необходимым изменением показателя степени.

Представление чисел с плавающей запятой подобно научной системе обозначений за исключением того, что вместо основания десять, все выполняется с основанием два. Не смотря на это, используются несколько подобных форматов, наиболее типичным из них является формат ANSI/Стандарт IEEE 754-1985 (IEEE - институт электро- и электронных инженеров, международная общественная организация – прим. перев.). Этот стандарт определяет формат для 32 разрядных чисел **одинарной точности**, а также 64 разрядных чисел **двойной точности**. Как показано на рис. 4.2, 32 разряда, используемые при одинарной точности, разделены на три отдельных группы: разряды с 0 по 22 формируют мантиссу, разряды с 23 по 30 формируют порядок числа, и 31 разряд является знаковым разрядом. Эти разряды формируют число v с плавающей запятой в соответствии со следующим соотношением:

$$v = (-1)^S \times M \times 2^{E-127}. \qquad (4.1)$$

Член $(-1)^S$ просто означает, что знаковый разряд S равен нулю для положительных чисел и 1 для отрицательных чисел. Переменная E - это число со значением, находящимся между 0 и 255, представляющим восемь битов порядка числа. Вычитание 127 из порядка числа позволяет показателю степени изменяться от 2^{-127} до 2^{128}. Другими словами, показатель степени хранится в *двоичном* формате *со смещением*, где величина смещения равна 127.



Пример 1

0 00000111 11000000000000000000000

\+ 7 0.75

$$+1.75 \times 2^{(7-127)} = +1.316554 \times 10^{-36}$$

Пример 2

1 10000001 01100000000000000000000

\- 129 0.375

$$-1.375 \times 2^{(129-127)} = -5.500000$$

Рис. 4.2 Формат хранения чисел одинарной точности с плавающей запятой

Мантисса M формируется из 23 разрядов в виде *двоичной дроби*. Например, десятичная дробь 2,783 интерпретируется как: 2 + 7/10 + 8/100 + 3/1000. Двоичная дробь 1,0101 означает: 1 + 0/2 + 1/4 + 0/8 + 1/16. Числа с плавающей запятой *нормализуются* таким же образом, как и числа в научной системе обозначений, то есть слева от запятой

имеется только одна отличная от нуля цифра (называемая, при использовании основания 2, *двоичной запятой*). Поскольку при использовании основания два единственной существующей цифрой, отличной от нуля, является 1, первой цифрой в мантиссе всегда будет 1, и поэтому хранить ее нет необходимости. Удаление этой избыточности позволяет увеличить точность числа за счет появления дополнительного бита. 23 хранимых разряда, обозначаемых как: m_{22}, m_{21}, m_{20}, …, m_0 формируют мантиссу согласно:

$$M = 1, m_{22}m_{21}m_{20}m_{19}...m_2m_1m_0. \qquad (4.2)$$

Другими словами, $M = 1 + m_{22} \cdot 2^{-1} + m_{21} \cdot 2^{-2} + m_{20} \cdot 2^{-3} + \ldots$. Если все разряды с 0 по 22 равны *нулю*, *M* принимает значение единицы. Если же все разряды с 0 по 22 равны *единице*, *M* всего на волосок меньше двух, т.е. $2–2^{23}$.

Используя эту схему кодирования, максимально представимым числом является: $\pm(2\text{-}2^{-23}) \times 2^{128} = \pm 6{,}8 \times 10^{38}$. В то время как минимально представимым числом будет: $\pm 1{,}0 \times 2^{-127} = \pm 5{,}9 \times 10^{-39}$. Стандарт IEEE слегка уменьшает этот диапазон, чтобы освободить несколько комбинаций разрядов для специального назначения. В частности самые большие и самые маленькие числа, допустимые в стандарте - это $\pm 3{,}4 \times 10^{38}$ и $\pm 1{,}2 \times 10^{-38}$, соответственно. Освобожденные комбинации допускают три специальных класса чисел: (1) ±0 - определен как число, у которого все разряды мантиссы и порядка равны нулю; (2) ±∞ - определена как число, у которого все разряды мантиссы равны нулю, а все разряды порядка равны единице; (3) группа очень маленьких *ненормализованных* чисел между $\pm 1{,}2 \times 10^{-38}$ $\pm 1{,}4 \times 10^{-45}$ - это числа более низкой точности, полученные за счет отказа от требования, что первая цифра мантиссы должна быть равна единице. Помимо этих трех специальных классов имеются комбинации битов, которым значения не присвоены, обычно упоминаемые как NANs (не числа).

Стандарт IEEE для двойной точности просто добавляет большее количество разрядов, как к мантиссе, так и к показателю степени. Из 64 разрядов, используемых для хранения чисел двойной точности, разряды с 0 по 51 являются мантиссой, разряды с 52 по 62 являются порядком, а разряд 63 является знаковым разрядом. Также как и ранее мантисса принимает значения между единицей и двойкой, т.е. $M = 1 + m_{51} \cdot 2^{-1} + m_{50} \cdot 2^{-2} + m_{49} \cdot 2^{-3}\ldots$. Одиннадцать разрядов порядка формируют числа между 0 и 2047, использование в качестве смещения 1023, позволяет изменять показатель степени от 2^{-1023} до 2^{1024}. Максимально и минимально представимыми числами являются $\pm 1{,}8 \times 10^{308}$ и $\pm 2{,}2 \times 10^{-308}$, соответственно. Каждое из них является невероятно большим и невероятно маленьким числом! Весьма необычно найти приложение, которому неадекватна одинарная точность. Ну а случай, в котором двойная точность будет ограничивать то, что Вы собираетесь реализовать, вероятно, Вы никогда не встретите.

Точность числа

Ошибки, связанные с представлением числа, очень похожи на ошибки квантования в процессе аналого-цифрового преобразования. Вы *желаете* хранить непрерывный диапазон значений; однако, Вы *можете* представлять только конечное число уровней квантования. Поэтому каждый раз, когда генерируется новое число, например, после математического вычисления, оно должно быть округлено к ближайшему значению, которое может быть сохранено в используемом Вами формате.

В качестве примера представьте, что Вы выделяете для хранения числа 32 разряда. Поскольку возможно точно 2^{32} = 4294967296 различных комбинаций битов, Вы можете представить точно 4294967296 различных чисел. Некоторые языки программирования

допускают переменные, называемые **длинным целым**, хранимые в виде 32 разрядного дополнения до двух с фиксированной запятой. Это означает, что 4294967296 возможных комбинаций битов представляют целые числа между -2147483648 и 2147483647. Для сравнения, одинарная точность с плавающей запятой распространяет эти 4294967296 комбинаций битов на значительно больший диапазон чисел: от $-3{,}4 \times 10^{38}$ до $3{,}4 \times 10^{38}$.

В переменных с фиксированной запятой промежуток между соседними числами всегда *строго один и тот же*. В системе обозначений с плавающей запятой промежуток между соседними числами вдоль представляемого диапазона чисел, изменяется. Если мы выберем число с плавающей запятой произвольным образом, величина промежутка между этим и следующим числом приблизительно в *десять миллионов раз меньшее*, чем непосредственно само это число (чтобы быть точным, $2^{-24} \div 2^{-23}$ от величины числа). Это является ключевой концепцией системы обозначений с плавающей запятой: большие числа имеют между ними большие промежутки, в то время как маленькие числа имеют маленькие промежутки. Это иллюстрируется на рисунке 4.3, где показаны последовательные числа с плавающей запятой и промежутки, которые их отделяют.

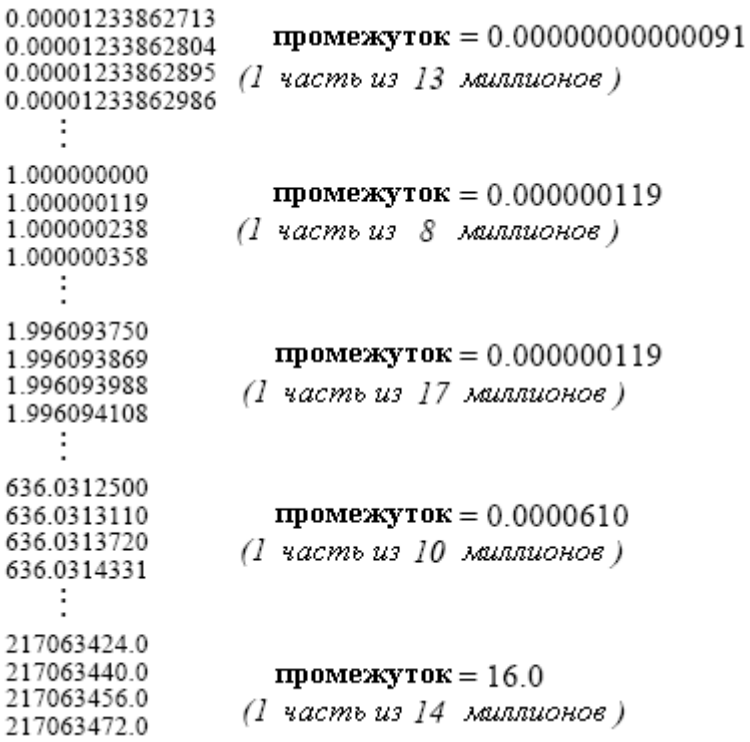


Рис. 4.3 Примеры промежутков между числами одинарной точности с плавающей запятой

Программа в Таблице 4.1 показывает, как ошибка округления (ошибка квантования в математических вычислениях) может создавать проблемы в ЦОС. В пределах программного цикла к переменной с плавающей запятой X добавляются два случайных числа, а затем в таком же порядке вычитаются обратно. В идеале, *ничего* не должно измениться. В действительности ошибка округления в каждой математической операции становится причиной постепенного дрейфа значения X от его первоначальной величины. Этот дрейф может принимать одну из двух форм в зависимости от того, каким образом ошибки суммируются вместе. Если ошибки округления случайным образом принимают положительные и отрицательные значения, величина переменной будет беспорядочно увеличиваться и уменьшаться. Если ошибки имеют преимущественно один и тот же знак, то значение переменной будет дрейфовать в сторону от первоначального значения значительно более быстро и монотонно.

Рисунок 4.4 показывает, как в этом примере программы переменная X дрейфует по величине. Очевидное беспокойство вызывает то, что *аддитивная ошибка* значительно хуже, чем *случайная ошибка*. Это происходит потому, что случайные ошибки имеют тенденцию взаимно компенсировать друг друга, в то время как аддитивные ошибки просто накапливаются. Грубо, аддитивная ошибка равна ошибке округления от каждого отдельного действия, умноженного на общее количество действий. Для сравнения, случайная ошибка увеличивается только пропорционально *квадратному корню* количества действий. Как показано в этом примере для обычных алгоритмов ЦОС, аддитивная ошибка может быть в сотни раз хуже, чем случайная ошибка.

Таблица 4.1

```
100 X = 1                          'initialize X
110 '
120 FOR I% = 0 TO 2000
130 A = RND                        'load random numbers
140 B = RND                        'into A and B
150 '
160 X = X + A                       'add A and B to X
170 X = X + B
180 X = X - A                       'undo the additions
190 X = X - B
200 '
210 PRINT X                         'ideally, X should be 1
220 NEXT I%
230 END
```

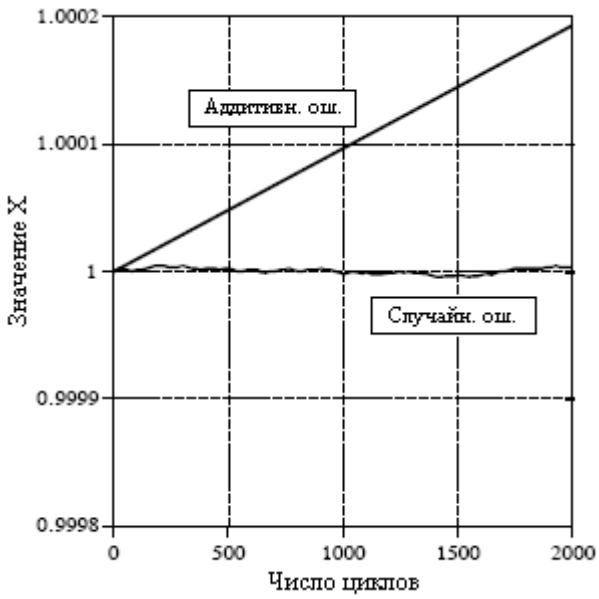


Рис. 4.4 Накопление ошибки округления в переменных с плавающей запятой

К сожалению, почти невозможно управлять тем или предсказать то, какое из этих двух поведений покажет на практике конкретный алгоритм. Например, программа в таблице 4.1 дает аддитивную ошибку. Ошибка может быть заменена на случайную просто за счет небольшой модификации добавляемых и вычитаемых чисел. В частности кривая случайной ошибки на рис. 4.4 была получена заданием A = EXP(RND) и B = EXP(RND),

вместо A = RND и B = RND. Вместо случайно распределенных между 0 и 1 чисел A и B, они стали экспоненциально распределенными значениями между 1 и 2.718. Даже такого небольшого изменения достаточно для того, чтобы уйти от накапливания ошибки.

Поскольку мы не можем управлять тем, каким способом будет накапливаться ошибка, держите в уме сценарий наихудшего случая. Ждите, что каждое число одинарной точности будет иметь ошибку приблизительно равную *одной части от сорока миллионов*, *умноженной на количество выполненных действий*. Все это базируется на предположении наличия аддитивной ошибки при средней ошибке от каждого отдельного действия в одну четвертую от уровня квантования. Точно такой же анализ показывает, что каждое число двойной точности имеет ошибку приблизительно в *одну часть от сорока квадрильонов*, *умноженную на количество выполненных действий*.

Таблица 4.2

```
100 'Floating Point Loop Control
110 FOR X = 0 TO 10 STEP 0.01
120 PRINT X
130 NEXT X
```

Выход программы:

```
0,00
0,01
0,02
0,03
.
.
.
9,960132
9,970133
9,980133
9,990133
```

```
100 'Integer Loop Control
110 FOR I% = 0 TO 1000
120 X = I%/100
130 PRINT X
140 NEXT I%
```

Выход программы:

```
0,00
0,01
0,02
0,03
.
.
.
9,96
9,97
9,98
9,99
10,00
```

Таблица 4.2 иллюстрирует особенно раздражающую проблему, связанную с ошибкой округления. Каждая из двух программ в этой таблице исполняет одну и ту же задачу: распечатать 1001 число, расположенное через равные интервалы между 0 и 10. Программа слева использует в качестве индекса переменную X с плавающей запятой. Выполнение цикла компьютер начинает с установки индексной переменной в первоначальное значение (в этом примере 0). В конце каждого цикла к индексу *прибавляется* значение шага (в нашем случае 0,01). Затем принимается решение: нужно ли еще повторять цикл или цикл завершен? Цикл завершается тогда, когда компьютер определит, что значение индекса *превышает* некоторое значение завершения (в этом примере 10,0). Как показано полученными на выходе данными, из-за ошибки округления при сложении в течение цикла величина X накапливает значительное расхождение. Фактически, накопленная ошибка *отменяет* выполнение последнего повторения цикла. Вместо того чтобы X на последнем цикле приняло значение 10,0, ошибка делает последнее значение X равным 10,000133. Поскольку X теперь превышает значение завершения, компьютер считает, что его работа проделана, и цикл преждевременно

заканчивается. Такая потеря последнего значения является типичной ошибкой многих компьютерных программ.

Для сравнения, программа справа для управления циклом использует в качестве переменной I% целое число. Сложение, вычитание или умножение двух целых чисел всегда дают другое целое число. Это означает, что система обозначений с фиксированной запятой не имеет с этими действиями абсолютно никакой ошибки округления. Целые числа идеальны для управления циклами, также как и другими переменными, которые подвергаются многократным математическим действиям. Последний проход цикла выполнится гарантированно! Всегда используйте для индексов цикла и счетчиков целые числа, если у Вас нет некоторой сильной причины, чтобы делать иначе.

Если Вы должны в качестве индекса использовать переменные с плавающей запятой старайтесь использовать дроби кратные степени *двойки* (такие как: 1/2, 1/4, 3/8, 27/16) вместо кратных степени *десяти* (такие как: 0,1, 0,6, 1,4, 2,3 и т.д.). Например, будет лучше использовать FOR X = 1 TO 10 STEP 0.125, чем FOR X = 1 TO 10 STEP 0.1. Это позволит индексу всегда иметь *точное* двоичное представление, снижая, таким образом, ошибку округления. Например, десятичное число 1,125 может быть точно представлено в двоичной системе обозначений: $1{,}00100000000000000000000 \times 2^0$. Для сравнения, десятичное число 1,1 находится *между* двумя числами с плавающей запятой: 1,0999999046 и 1,1000000238 (в двоичном виде это числа: $1{,}00011001100110011001100 \times 2^0$ и $1.00011001100110011001101 \times 2^0$). Это приводит к присущей погрешности каждый раз, когда в программе вычисляется значение 1,1.

Полезно запомнить, что числа с плавающей запятой одинарной точности имеют *точное* двоичное представление для каждого целого числа, находящегося между ±16,8 миллионами (чтобы быть точными $\pm 2^{24}$). Выше этого значения промежутки между уровнями становятся больше чем единица, приводя к тому, что некоторые значения целых чисел будут пропущены. Это позволяет складывать, вычитать и умножать целые числа с плавающей запятой (между ±16,8 миллионами) без ошибки округления.

Скорость выполнения: язык программирования

Программирование ЦОС может быть нестрого разделено на три уровня сложности: *языки низкого уровня*, *языки высокого уровня* и *специфические приложения*. Чтобы понять разницу между ними нам нужно начать с основ цифровой электроники. Все микропроцессоры базируются на наборе внутренних двоичных регистров, то есть группе триггеров, которые могут хранить последовательности единиц и нулей. Например, микропроцессор 8088, ядро первого IBM PC, содержит *четыре* регистра общего назначения, каждый из которых включает 16 разрядов. Они идентифицируются именами: AX, BX, CX, и DX. Микропроцессор содержит так же *девять* дополнительных регистров специального назначения называемых: SI, DI, SP, BP, CS, DS, SS, ES, и IP. Например, IP – счетчик команд содержит адрес размещения в памяти, следующей исполняемой команды.

Предположим, что Вы пишите программу сложения двух чисел 1234 и 4321. Когда начинается выполнение программы, IP содержит адрес сегмента памяти, в котором находятся комбинации единиц и нулей, приведенные в таблице 4.3. Несмотря на то, что для большинства людей эти комбинации выглядят малозначащими, они содержат все команды и данные, необходимые для выполнения нашей задачи. Например, когда микропроцессор сталкивается с комбинациями битов 00000011 11000011, он интерпретирует их как команду: взять 16 битов, хранимых в регистре BX, выполнить их двоичное сложение с 16 битами, хранимыми в регистре AX и сохранить результат в регистре AX. Этот уровень программирования называется **машинными кодами,** и он только на волосок выше, чем работа с фактическими электронными цепями.

Поскольку работа в двоичных кодах, в конечном счете, приведет даже очень терпеливого и увлеченного инженера к сумасшествию, этим комбинациям единиц и нулей

присвоены имена в соответствии с функцией, которую они выполняют. Этот уровень программирования называется **ассемблером**, а его пример показан в таблице 4.4. Хотя программа на ассемблере значительно проще для понимания, по существу это то же самое, что и программирование в машинных кодах, поскольку и здесь соответствие между командами программы и действиями, выполняемыми микропроцессором совпадают один в один. Например, ADD AX, BX переводится как: 00000011 11000011. Для преобразования кода, написанного на языке **низкого уровня** ассемблере, приведенного в таблице 4.4 (называемого кодом источника), в комбинации единиц и нулей, показанных в таблице 4.3 (называемые объектным кодом или исполняемым кодом), используется программа называемая **транслятором**. Такой исполняемый код может быть непосредственно запущен на микропроцессоре. Очевидно, что программирование на ассемблере требует широкого понимания внутреннего устройства конкретного микропроцессора, который Вы собираетесь использовать.

Таблица 4.3

```
10111001
11010010
00000100
10001001
00001110
00000000
00000000
10111001
11100001
00010000
10001001
00001110
00000010
00000000
10100001
00000000
00000000
10001011
00011110
00000010
00000000
00000011
11000011
10100011
00000100
00000000
```

Таблица 4.4

```
MOV CX,1234          ;store 1234 in register CX, and then
MOV DS:[0],CX        ;transfer it to memory location DS:[0]
MOV CX,4321          ;store 4321 in register CX, and then
MOV DS:[2],CX        ;transfer it to memory location DS:[2]
MOV AX,DS:[0]        ;move variables stored in memory at
MOV BX,DS:[2]        ;DS:[0] and DS:[2] into AX & BX
```

```
ADD AX,BX                    ;add AX and BX, store sum in AX
MOV DS:[4],AX                ;move the sum into memory at DS:[4]
```

Программирование на языке ассемблера позволяет непосредственно манипулировать цифровой электроникой: регистрами, распределением памяти, битами статуса (флагами – прим. перев.) и т.д. Следующий уровень сложности манипулирует *абстрактными* переменными без какой-либо ссылки на конкретные аппаратные средства компьютера. Он называется **компилятором** или языком **высокого уровня**. Обычно в использовании находится дюжина или около того языков высокого уровня таких как: C, BASIC, FORTRAN, PASCAL, APL, COBOL, LISP и т.д. В таблице 4.5 показана программа на языке BASIC, выполняющая сложение чисел 1234 и 4321. Программист знает только о переменных A, B, и C и ничего об аппаратных средствах компьютера.

Таблица 4.5

```
100 A = 1234
110 B = 4321
120 C = A+B
130 END
```

Для преобразования высокоуровневого исходного кода непосредственно в машинный код используется программа называемая **компилятором**. Непосредственное преобразование требует от компилятора *закрепления* за каждой упомянутой абстрактной переменной места в аппаратной памяти. Например, когда компилятор сталкивается с переменной A в Таблице 4.5 в первый раз (строка 100), он понимает, что программист использует этот символ для обозначения переменной одинарной точности с плавающей запятой. Соответственно компилятор выделяет четыре байта памяти, которые будут использоваться ни для чего более, кроме как для хранения значений этих переменных. Каждый следующий раз, когда в программе появится A, компьютер знает, что при необходимости нужно обновить эти четыре байта. Компилятор разбивает так же сложные математические выражения, такие как $Y = LOG(X^{COS(Z)})$, на более простую арифметику. Дело в том, что микропроцессоры знают только, как складывать, вычитать, умножать и делить. Что-нибудь более сложное может быть выполнено только как последовательность этих четырех элементарных действий.

Языки высокого уровня изолируют программиста от аппаратных средств компьютера. Это делает программирование значительно более легким и позволяет передавать исходный код между различными типами микропроцессоров. Наиболее важно то, что программисту, использующему компилируемый язык, не нужно знать *ничего* о внутренней работе компьютера. Эту ответственность взял на себя другой программист, тот, кто написал компилятор.

Большинство компиляторов преобразуют *всю* программу в машинные коды *до* ее исполнения. Исключением из этого является тип компилятора называемый **интерпретатором**, среди которых **интерпретатор BASIC** является наиболее типичным примером. Интерпретатор преобразует *одну строку* исходного кода в машинный код, исполняет этот машинный код, а затем переходит к следующей строке исходного кода. Это обеспечивает интерактивную среду для простых программ, хотя скорость выполнения программы при этом предельно низка (думается, раз в 100 меньше).

Самый высокий уровень сложности программирования находится в пакетах **приложений** для ЦОС. Они существуют в разнообразных формах и часто предназначены для поддержки конкретных аппаратных средств. Предположим, Вы приобрели недавно разработанный микропроцессор для использования в вашем текущем проекте. Такие приборы часто обладают большим количеством встроенных характеристик для ЦОС:

аналоговые входы, аналоговые выходы, цифровые входы/выходы, фильтры антисовмещения и восстанавливающие фильтры и т.д. Вопрос в том, как их Вам программировать? В самом плохом случае, изготовитель даст Вам *ассемблер*, и будет надеяться, что Вы изучите внутреннюю архитектуру устройства. В более типичном сценарии будет обеспечен *компилятор* с языка высокого уровня С, позволяющий Вам заниматься программированием, не беспокоясь о том, как фактически работает микропроцессор.

В наилучшем случае, производитель обеспечит Вас сложным пакетом программ, помогающим осуществлять программирование: библиотеки алгоритмов, предварительно написанные подпрограммы для ввода/вывода, отладочное оборудование и т.д. Для формирования желаемой системы Вам нужно будет просто соединять иконки на простом в использовании дисплее. *Вещи*, которыми Вы манипулируете здесь, это пути прохождения сигналов, алгоритмы для обработки сигналов, параметры аналоговых входов/выходов и т.д. Когда Вы будете удовлетворены проектом, он трансформируется в соответствующий машинный код, предназначенный для исполнения в аппаратных средствах. Различные типы пакетов приложений используются для обработки изображения, спектрального анализа, наблюдения и управления, проектирования цифровых фильтров и т.д. И это только очертания будущего.

Различие между этими тремя уровнями может быть очень нечетким. Например, большинство компилируемых языков позволяют Вам непосредственно управлять аппаратными средствами. В то же время язык высокого уровня, снабженный хорошей библиотекой с функциями ЦОС, очень близок к пакетам приложений. Чтобы остановиться на одной из этих трех категорий главное понимать, чем Вы собираетесь манипулировать: (1) аппаратными средствами, (2) абстрактными переменными, (3) всей процедурой и алгоритмами.

За этой классификацией стоит также и другая важная концепция. Когда вы пользуетесь языком высокого уровня, Вы полагаетесь на программиста, написавшего компилятор и понимающего лучшие способы манипуляции аппаратными средствами. Точно также, когда Вы пользуетесь пакетом приложений, Вы полагаетесь на программиста, написавшего пакет и понимающего лучшую технику ЦОС. Здесь и скрывается препятствие, эти программисты никогда не видели конкретных проблем, с которыми Вы имеете дело. Следовательно, они не всегда могут обеспечить Вам оптимальное решение. Если Вы работаете с языками *высокого* уровня, то будьте готовы, что полученный в итоге машинный код будет *менее* эффективным в смысле использования памяти, скорости и точности.

Какой язык программирования Вы должны использовать? Это зависит от того, *кто* Вы такой и *что* Вы планируете делать. Большинство ученых в области компьютерной техники и программистов используют С (или его более продвинутую версию С++). Мощность, гибкость, модульность все это есть у С. С настолько популярен, что вопрос ставится так: “Почему некто программирует свои ЦОС приложения на чем-то другом, а не на С?” На ум приходят три ответа. Во-первых, ЦОС выросла настолько быстро, что некоторые организации и индивидуумы, увязли в типах данных других языков, таких как FORTRAN и PASCAL. Это особенно справедливо для военных и правительственных ведомств, которые, как печально известно, не спешат изменяться. Во-вторых, некоторые приложения требуют предельной эффективности, достижимой только при программировании на языке ассемблера. Они попадают в категорию “чуть больше скорости за очень большой объем работы”. В-третьих, С - не особенно легкий язык даже для мастера, ну а тем более для тех, кто занимается программированием время от времени. Сюда входит широкий круг инженеров и ученых, которым техника ЦОС нужна от случая к случаю для вспомогательных целей в их исследованиях или проектной деятельности. Эта группа часто обращается к BASIC из-за его простоты.

Почему для этой книги выбран BASIC? Эта книга посвящена не стилю программирования, а *алгоритмам*. Вы должны будете сконцентрироваться на технике ЦОС и не будете отвлекаться на особенности конкретного языка. Например, все программы в этой книге содержат *номера строк*. Это позволяет более легко описывать то, как работает программа: “строка 100 выполняет то-то и то-то, строка 110 делает это и то”, и т.д. Конечно, Вы, вероятно, никогда не будете использовать номера строк в ваших реальных программах. Дело в том, что *изучение* ЦОС преследует совсем другие цели, чем *использование* ЦОС. На рынке присутствует большое количество книг, которые предлагают изящные исходные коды для алгоритмов ЦОС. Если Вы просто ищете уже написанные коды чтобы перенести их в свою программу, то Вы не туда попали.

Сравнивать скорость работы аппаратных или программных средств задача неблагодарная, независимо от того, каков будет результат, проигравший будет кричать, что состязание было несправедливо! Программисты, которым нравятся языки высокого уровня (такие, как ученые в области традиционных компьютеров), будут аргументировать тем, что ассемблер только на 50% быстрее, чем компилированный код, но в пять раз более труден. Те, кто любит ассемблер (обычно ученые и инженеры в области аппаратных средств) будут призывать все пересмотреть: ассемблер в пять раз быстрее и только на 50% труднее в использовании. Как и в большинстве споров обе стороны могут привести отдельные данные, подтверждающие их утверждения.

Как правило, следует ждать, что подпрограмма, написанная на ассемблере, будет от 1,5 до 3,0 раз быстрее, чем сопоставимая программа языка высокого уровня. Единственный способ узнать точное значение этого, написать код и провести тест на скорость. Поскольку персональные компьютеры увеличивают свою скорость ежегодно на 40%, написание программы на ассемблере эквивалентно приблизительно двухлетнему скачку в компьютерной технологии.

Большинство профессиональных программистов сильно обижаются на идею использовать ассемблер и хихикают, если Вы предлагаете BASIC. Их рациональность очень проста: ассемблер и BASIC не позволяют использовать хорошие навыки программирования. Хороший код должен быть *компактным* (способным переноситься с одного компьютера на другой), *модульным* (разбитым на хорошо определенные структурированные подпрограммы) и *легко понимаемым* (содержать большое количество комментариев и описаний имен переменных). Слабая структурированность ассемблера и BASIC делают трудным достижение этих стандартов. Это усложняется еще тем фактом, что люди, привлеченные к составлению программ на ассемблере и BASIC, часто имеют небольшой формальный опыт в надлежащем структурировании и документировании программного обеспечения.

Любители ассемблера отвечают на это нападение своими собственными доводами. Допустим, Вы пишите программу на C, а ваш соперник пишет такую же программу на ассемблере. Первое впечатление конечного пользователя будет таково, что ваша программа утиль-сырье, потому что она в два раза медленнее. Никто и не предполагает, что Вы будете писать большие программы на ассемблере, а только лишь те куски программы, которые требуют быстрого исполнения. Например, большое число функций в библиотеках программного обеспечения по ЦОС написано на ассемблере, к которым затем обращаются из больших программ написанных на C. Даже самый верный пурист (человек, стремящийся к очищению языка от иноязычных элементов, неологизмов и вульгаризмов, иногда показному – прим. перев.) программного обеспечения будет *использовать* код ассемблера до тех пор, пока ему самому не нужно будет *писать* на нем.

Скорость выполнения: аппаратные средства

Вычислительная мощность компьютеров увеличивается так быстро, что любая книга на эту тему устаревает еще до публикации. Это настоящий кошмар для автора!

Первая IBM PC появилась в 1981 году, ее основу составлял микропроцессор 8088 с тактовой частотой 4,77 МГц и 8 битовой шиной данных. За ним последовало новое поколение персональных компьютеров появляющихся каждые 3-4 года: 8088 → 80286 → 80386 → 80486 → 80586 (Pentium). Каждая из этих новых систем увеличивала скорость вычислений приблизительно в *пять* раз по сравнению с предыдущей технологией. К 1996 году тактовая частота увеличилась до 200 МГц, а шина данных до 32 битов. Совместно с другими улучшениями это привело к увеличению вычислительной мощности приблизительно в *тысячу* раз в течение всего лишь 15 лет. В *следующие* 15 лет, Вы можете ожидать *другого* увеличения в тысячу раз.

Единственный путь получить современную информацию в этой быстро изменяющейся области это непосредственно от производителя: рекламные объявления, листы спецификации, прейскуранты, и т.д. Забудьте о книгах, по обработке данных, читайте журналы и свои ежедневные газеты. Ожидайте, что исходная скорость вычислений будет более чем удваиваться каждые два года. Изучение текущего состояния мощности компьютеров просто недостаточно; Вам необходимо понимать и отслеживать их эволюцию. Помня об этом, мы можем перейти к краткому обзору того, как скорость работы ограничивается аппаратными средствами компьютера. Поскольку компьютеры состоят из большого числа подсистем, время, требуемое для исполнения конкретной задачи, будет зависеть от двух первичных факторов: (1) скорости индивидуальных подсистем и (2) необходимого времени для передачи данных между этими блоками. На рис. 4.5 показана упрощенная диаграмма наиболее важных компонент ограничивающих скорость для типичного персонального компьютера. **Центральный процессор (ЦП)** это сердце системы. Как упоминалось ранее, он состоит из дюжины или около того регистров, каждый из которых способен хранить 32 бита (в текущем поколении персональных компьютеров). В ЦП включена также цифровая электроника необходимая для выполнения элементарных действий, таких как циклического переноса битов и выполнения арифметики с фиксированной запятой.

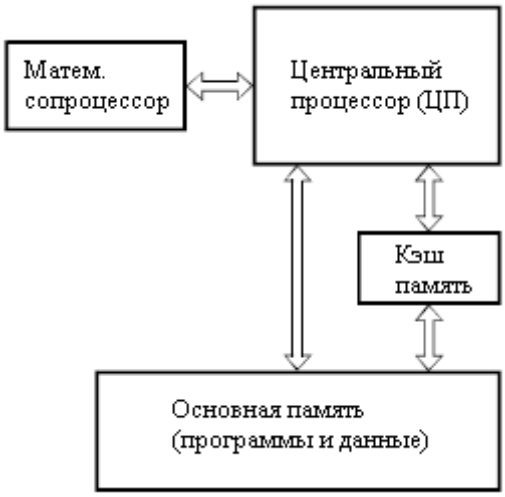


Рис. 4.5 Архитектура типичной компьютерной системы

Большая часть используемой математики обрабатывается за счет передачи данных в специальную схему аппаратных средств называемую **математическим сопроцессором** (также называют **арифметико-логическим устройством** или **АЛУ**). Математический сопроцессор может находиться в той же самой микросхеме, что и ЦП, или может быть выполнен в виде отдельного электронного компонента. Например, сложение двух чисел с плавающей запятой требует, чтобы ЦП передал в математический сопроцессор 8 байтов данных (4 для каждого числа) и несколько байтов, описывающих, что нужно делать с

этими данными. После короткого времени вычислений математический сопроцессор передаст 4 байта, содержащих число с плавающей запятой и являющееся суммой, обратно в ЦП. В большинстве недорогих компьютерных систем математический сопроцессор отсутствует или добавляется в систему как дополнительный аксессуар (начиная с микропроцессора Pentium и выше, математический сопроцессор является неотъемлемой составной частью микросхемы ЦП IBM PC совместимых компьютеров – прим. перев.). Например, микропроцессор 80486DX имеет встроенный математический сопроцессор, в то время как 80486SX нет. В таких низко производительных системах *аппаратные* средства заменяются *программными*. Каждая математическая функция разбивается на элементарные двоичные операции, которые могут быть выполнены непосредственно ЦП. Хотя это дает тот же самый результат, время выполнения программы становится значительно больше, скажем от 10 до 20 раз.

Большинство программного обеспечения для персональных компьютеров может использоваться как с математическим сопроцессором, так и без него. Это достигается за счет использования компилятора, генерирующего машинные коды для работы в обоих этих случаях, при этом все коды хранятся в конечной исполняемой программе. Если в конкретном используемом компьютере присутствует математический сопроцессор, будет запущена одна секция кодов. Если математического сопроцессора нет, будет использована другая секция кодов. Компилятор может быть так же настроен на генерирование кодов только для одного из этих случаев. Например, Вы можете случайно столкнуться с программой, которая требует наличия математического сопроцессора, и если запустите ее на компьютере, где он отсутствует, то потерпите неудачу. Обычно, никакой выгоды из наличия математического сопроцессора не извлекают приложения подобные текстовым процессорам. Это связано с тем, что они только перемещают используемые данные по памяти и не используют их в вычислении математических выражений. Аналогично, присутствие математического сопроцессора безразлично для вычислений, включающих переменные с фиксированной запятой (целые числа), поэтому они обрабатываются в ЦП. С другой стороны, скорости выполнения программ ЦОС и других вычислительных программ, использующих вычисления с плавающей запятой, с математическим сопроцессором и без него могут отличаться на порядок.

В большинстве компьютерных систем ЦП и основная память находятся в разных микросхемах. Причины этого очевидны, Вы желаете, чтобы основная память была очень большой и очень быстрой. К сожалению, это приводит к тому, что память становится очень дорогой. Самое типичное узкое место для быстродействия это передача данных между основной памятью и ЦП. ЦП по конкретному адресу *запрашивает* двоичную информацию из основной памяти, а затем *ожидает* ее получения. Обычным приемом, позволяющим обойти эту проблему, является использование **кэш памяти**. Это очень быстрая память небольшого объема, используемая в качестве буфера между ЦП и основной памятью. Ее емкость обычно составляет несколько сотен килобайт. Когда ЦП по конкретному адресу запрашивает двоичные данные из основной памяти, высокоскоростная цифровая электроника копирует содержимое *раздела* основной памяти вокруг этого адреса в кэш память. В следующий раз, когда ЦП запросит информацию из памяти, весьма вероятно, что она уже будет содержаться в кэш памяти, таким образом, поиск становится очень быстрым. В основу этого приема положен тот факт, что программы имеют тенденцию обращаться к месту в памяти, являющемуся соседним с ближайшими данными, к которым обращение происходило в предыдущий раз. В типичном приложении для персонального компьютера добавление кэш памяти может улучшить общее быстродействие в несколько раз. Кэш память может быть расположена в той же микросхеме что и ЦП, или может быть внешним электронным устройством.

Скорость, с которой данные могут быть переданы между подсистемами, зависит от количества имеющихся параллельных линий данных, и максимальной скорости, с которой цифровые сигналы могут быть переданы по каждой из этих линий. Вообще, в пределах

одной и той же микросхемы, цифровые данные могут передаваться с намного более высокой скоростью по сравнению с передачей данных между микросхемами. Точно так же, пути данных, которые должны пройти к другим цепям на печатных платах через электрические разъемы (то есть, шинные структуры), будут еще медленнее. Это является сильной мотивацией для размещения внутри ЦП как можно большего количества электроники.

Особенно плохой с точки зрения быстродействия компьютера является *обратная совместимость*. Когда компьютерная компания представляет новый продукт, скажем карту сбора данных или программное обеспечение, она желает продавать все это на самом большом, на сколько это возможно, рынке. Это означает, что все это должно быть совместимым с большинством компьютеров, находящихся сейчас в использовании, которые могут включать несколько технологических поколений. Это очень часто ограничивает работу аппаратных средств или программного обеспечения на уровне более старых систем. Например, предположим, что Вы купили карту ввода/вывода, которая вставляется в шину вашего персонального компьютера Pentium с тактовой частотой 200 МГц, предоставляя Вам восемь цифровых линий, по которым за один раз может быть передан и получен один байт данных. Затем Вы пишете программу на ассемблере для высокоскоростного обмена данных между вашим компьютером и некоторым внешним прибором для научных экспериментов или другим компьютером. Большим сюрпризом для вас стало то, что максимальная скорость передачи данных всего около 100000 байт в секунду, более чем в *тысячу* раз меньше, чем тактовая частота микропроцессора! Злодеем здесь является шина ISA, технология *обратно совместимая* с компьютерами начала 1980-ых.

В таблице 4.6 приведено время выполнения некоторых операций в микросекундах для компьютеров нескольких поколений. Очевидно, что Вам нужно обращаться с ними как с очень грубым приближением. Если Вы хотите понять *Вашу* систему, проведите измерения *ее* характеристик. Это делается очень просто, напишите цикл, который выполняет *миллион* некоторых операций, и используйте свои часы, для того чтобы засечь, сколько времени это займет. Первые три системы 80286, 80486 и Pentium являются стандартными настольными персональными компьютерами 1986, 1993 и 1996 годов, соответственно. Четвертая, микропроцессор 1994 года, разработанный специально для задач ЦОС фирмой Texas Instruments TMS320C40.

Таблица 4.6

	80286 (12 МГц)	80486 (33 МГц)	PENTIUM (100 МГц)	TMS320C40 (40 МГц)
ЦЕЛЫЕ				
A% = B%+C%	1,6	0,12	0,04	
A% = B%-C%	1,6	0,12	0,04	
A% = B%×C%	2,7	0,59	0,13	
A% = B%÷C%	64	9,2	1,5	
ПЛАВАЮЩАЯ ЗАПЯТАЯ				
A = B+C	33	2,5	0,50	0,10
A = B-C	35	2,5	0,50	0,10
A = B×C	35	2,5	0,50	0,10
A = B÷C	49	4,5	0,87	0,8
A = SQR(B)	45	5,3	1,3	0,9
A = LOG(B)	186	19	3,4	1,7
A = EXP(B)	246	25	5,5	1,7
A = B^C	311	31	5,3	2,4

A = SIN(B)	262	30	6,6	1,1
A = ARCTAN(B)	168	21	4,4	2,2

Pentium быстрее чем система 80286 в четыре раза, (1) более высокая тактовая частота, (2) больше линий в шине данных, (3) добавлена кэш память, (4) более эффективное внутреннее устройство, обеспечивающее меньшее количество тактов на выполнение одной команды.

Если Pentium это Кадилак, то TMS320C40 это Феррари: меньше комфорта, но зато сумасшедшая скорость. Эта микросхема - представитель семейства микропроцессоров специально разработанных для того, чтобы увеличить время выполнения алгоритмов ЦОС. Другими микросхемами этой категории являются i860 фирмы Intel, DSP3210 фирмы AT&T, DSP96002 фирмы Motorola и ADSP-2171 фирмы Analog Devices (отечественным представителем этой категории является микросхема Л1879ВМ1 Научно технического центра “Модуль” – прим. перев.). Их часто называют: **цифровые процессоры обработки сигналов**, **цифровыми сигнальными процессорами** и **процессорами с RISC** (Reduced Instruction Set Computer – компьютер с сокращенным набором команд) **архитектурой**. Это последнее название отражает то, что увеличенная скорость является результатом меньшего количества команд уровня ассемблера, доступных программисту. Для сравнения, более традиционные микропроцессоры, такие как Pentium, называются процессорами с **CISC** (Complex Instruction Set Computer - компьютер с полным набором команд) архитектурой.

Микропроцессоры ЦОС используются двумя способами: в качестве ведомых модулей под управлением более традиционных компьютеров или как встроенный процессор в специализированном приложении, таком как сотовый телефон. Некоторые модели обрабатывают только числа с фиксированной запятой, в то время как другие могут работать с плавающей запятой. Внутренняя архитектура, используемая для получения повышенной скорости, включает: (1) значительный объем очень быстрой кэш памяти, содержащейся внутри микросхемы, (2) раздельные шины для программ и данных, позволяющие осуществлять одновременное обращение к обоим (так называемая **Гарвардская архитектура**), (3) содержащиеся непосредственно в микропроцессоре аппаратные средства для осуществления быстрых математических вычислений и (4) конвейерную организацию.

В **конвейерной** архитектуре *аппаратные средства*, требуемые для некоторой задачи, разбиваются на несколько последовательных элементов. Например, сложение двух чисел в конвейерной архитектуре может быть выполнено на трех элементах конвейера. Первый элемент конвейера не выполняет никаких функций кроме извлечения из памяти чисел, которые будут складываться. Единственной задачей второго элемента является сложение двух чисел вместе. Третий элемент не выполняет ничего кроме занесения результата в память. Если каждый элемент может выполнить свою задачу за один машинный такт, выполнение всей процедуры займет три такта. Ключевым моментом в конвейерной структуре является то, что следующая задача может быть начата до окончания предыдущей задачи. В этом примере мы можем начать сложение *других* двух чисел сразу же после того, как будет освобожден первый элемент, т.е. по окончании первого такта. Для большого числа операций скорость системы будет оцениваться как одно сложение за один такт, хотя для осуществления одного сложения любых двух чисел требуется три машинных такта. Конвейеры обладают огромной скоростью, но их очень трудно программировать. Алгоритм должен давать возможность начать новое вычисление даже тогда, когда результат предыдущего вычисления еще не известен (поскольку он находится еще в конвейере).

Более детально микропроцессоры для цифровой обработки сигналов обсуждаются в Главах 28 и 29. Это удивительные устройства; их огромная мощность и низкая стоимость сделали ЦОС доступной широкому кругу потребителей и научных

приложений. Это одна из технологий, которая будет формировать двадцать первое столетие.

Скорость выполнения: стиль программирования

Несмотря на то, что аппаратные средства компьютера и языки программирования важны для максимизирования скорости выполнения, они не являются той основой, которую Вы можете менять ежедневно. Для сравнения, то, *как Вы программируете*, может быть изменено в любой момент и радикально затронет время необходимое на выполнение программы. Вот три предложения на эту тему.

Первое, **всякий раз когда это возможно, используйте целые числа вместо переменных с плавающей запятой**. Обычные микропроцессоры, такие как используются в персональных компьютерах, обрабатывают целые числа от 10 до 20 раз быстрее, чем числа с плавающей запятой. В системах без математического сопроцессора эта разница может иметь отношение 200 к 1. Исключение этому составляет деление целых чисел, которое часто сопровождается преобразованием целых величин в значения с плавающей запятой. Это делает эту операцию ужасно медленной по сравнению с другими операциями над целыми числами. Для более детального изучения смотрите таблицу 4.6.

Второе, **избегайте использования функций таких как: *sin(x)*, *log(x)*, y^x и т.д.** Такие трансцендентные функции вычисляются как последовательность сложений, вычитаний и умножений. Например, ряд Маклорена дает:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \ldots$$

$$\cos(x) = x - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \ldots \qquad (4.3)$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \ldots$$

Несмотря на то, что эти соотношения бесконечны, их члены быстро становятся достаточно маленькими, чтобы ими можно было пренебречь. Например:

$$\sin(1) = 1 - 0{,}166666 + 0{,}008333 - 0{,}000198 + 0{,}000002 - \ldots.$$

Такие функции требуют приблизительно в десять раз более длительных вычислений, чем простое сложение или умножение (см. таблицу 4.6). Для того чтобы обойти такие вычисления, могут быть использованы несколько ухищрений таких как: $x^3 = x \cdot x \cdot x$; $sin(x) = x$ в тех случаях, когда x очень мал; $sin(-x) = -sin(x)$ там, где Вы уже знаете одно из значений и должны найти другое и т.д. Большинство языков обеспечивают только несколько трансцендентных функций и надеются, что другие Вы получите посредством соотношений приведенных в Таблице 4.7. Не удивительно, что эти полученные вычисления еще медленнее.

Другим подходом является предварительное вычисление таких медленных функций и хранение их значений в таблице поиска (ТП). Например, представьте восьмибитовую систему сбора данных, используемую для непрерывного наблюдения за величиной падения *напряжения* на резисторе. Если интересующим параметром является рассеиваемая резистором *мощность*, измеряемое напряжение может быть использовано

для ее вычисления в соответствии с $P = V^2/R$. В качестве более быстрой альтернативы мощность, соответствующая каждому из 256 возможных значений измерения напряжения рассчитывается заранее и сохраняется в ТП. Во время работы системы измеренное напряжение цифровое число между 0 и 255 становится индексом для нахождения соответствующего значения мощности в ТП. Таблицы поиска могут быть в сотни раз быстрее, чем прямые вычисления.

Таблица 4.7

ФУНКЦИЯ	УРАВНЕНИЕ ДЛЯ ВЫЧИСЛЕНИЯ
Secant (X) =	1/COS(X)
Cosecant (X) =	1/SIN(X)
Cotangent (X) =	1/TAN(X)
Arc Sine (X) =	ATN(X/SQR(1-X*X))
Arc Cosine (X) =	-ATN(X/SQR(1-X*X)) + PI/2
Arc Secant (X) =	ATN(SQR(X*X-1)) + (SGN(X)-1) * PI/2
Arc Cosecant (X) =	ATN(1/SQR(X*X-1)) + (SGN(X)-1) * PI/2
Arc Cotangent (X) =	-ATN(X) + PI/2
Hyperbolic Sine (X) =	(EXP(X)-EXP(-X))/2
Hyperbolic Cosine (X) =	(EXP(X)+EXP(-X))/2
Hyperbolic Tangent (X) =	(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
Hyperbolic Secant (X) =	1/HYPERBOLIC COSINE
Hyperbolic Cosecant (X) =	1/HYPERBOLIC SINE
Hyperbolic Cotangent (X) =	1/HYPERBOLIC TANGENT
Arc Hyperbolic Sine (X) =	LOG(X+SQR(X*X+1))
Arc Hyperbolic Cosine (X) =	LOG(X+SQR(X*X-1))
Arc Hyperbolic Tangent (X) =	LOG((1+X) /(1-X))/2
Arc Hyperbolic Secant (X) =	LOG((SQR(1-X*X)+1)/X)
Arc Hyperbolic Cosecant (X) =	LOG(1+SGN(X)*SQR(1+X*X))/X
Arc Hyperbolic Cotangent (X) =	LOG((X+1)/(X-1))/2
$LOG_{10}(X)$ =	LOG(X)/LOG(10) = 0.4342945 LOG(X)
PI =	4*ATN(1) = 3.141592653589794

Третье, **изучайте**, **что является *быстрым* и что является *медленным* в Вашей конкретной системе**. Это придет с экспериментами и опытом, и здесь всегда будут сюрпризы. Особое внимание уделяйте командам *графики* и *ввода/вывода*. Существует обычно несколько способов осуществления этих требований, для которых скорости могут быть чрезвычайно различными. Например, команда BASIC-а *BLOAD* преобразует файл данных непосредственно в раздел памяти. Считывание байта за байтом того же самого файла в память (в цикле) может быть в 100 раз медленнее. Другой пример, команда BASIC-а *LINE* может использоваться, для того чтобы нарисовать на видео экране цветную коробку. Изображение пикселя за пикселем той же самой коробки может быть также в 100 раз дольше. Даже размещение оператора *печати* в пределах цикла (для отслеживания того, что происходит) может замедлить работу в *тысячи* раз!

ГЛАВА
5

Линейные системы

Большинство методов ЦОС основано на стратегии разделяй и властвуй, носящей название *суперпозиции*. Обрабатываемый сигнал разбивается на простые компоненты, при этом каждая компонента обрабатывается индивидуально, а затем результаты объединяются. Этот подход деления отдельной сложной проблемы на множество простых обладает огромной силой. Суперпозиция – термин, означающий, что применяются некоторые математические правила, может использоваться только с линейными системами. К счастью большинство приложений, встречающихся в науке и инженерной практике, попадают в эту категорию. Эта глава знакомит с основами ЦОС: что означает линейная система, различные пути разбиения сигналов на более простые компоненты и как суперпозиция обеспечивает разнообразие методов обработки сигнала.

Сигналы и системы

Сигнал - это описание того, как один параметр изменяется в зависимости от изменения другого параметра. Например, изменение напряжения в электронной цепи во времени или изменение яркости изображения в зависимости от расстояния. **Система** - это некоторый процесс, который в ответ на *входной сигнал* производит *выходной сигнал*. Сказанное иллюстрируется блок-схемой на рис. 5.1. На входе и выходе непрерывной системы находятся непрерывные сигналы, такие же, как сигналы в аналоговой электронике. На входе и выходе дискретной системы находятся дискретные сигналы, такие же, как значения, хранимые в управляемых программами массивах.

Для обозначения сигналов используются несколько правил. Им не всегда следуют в ЦОС, хотя они очень обычны, и Вы должны их запомнить. Математика достаточно трудна без ясной системы обозначений. Первое, *непрерывные* сигналы используют круглые скобки типа $x(t)$ и $y(t)$, в то время как *дискретные* сигналы используют квадратные скобки, как $x[n]$ и $y[n]$. Второе, для обозначения сигналов используют строчные буквы. Заглавные буквы зарезервированы для частотной области, обсуждаемой в следующих главах. Третье, название, присваиваемое сигналу, обычно описывает параметры, которые он представляет. Например, *напряжение*, зависящее от времени, может быть названо $v(t)$ (слова *напряжение* и *время* в английском языке пишутся соответственно как ***v****oltage* и ***t****ime* – прим. перев.), или *цена* фондового рынка, измеряемая каждый *день*, может обозначаться как $p[d]$ (слова *цена* и *день* в английском языке пишутся соответственно как ***p****rice* и ***d****ay* – прим. перев.).

Часто сигналы и системы обсуждаются без знания точно представляемых параметров. Это так же, как использование x и y в алгебре, без присвоения переменным конкретного физического смысла. Данное приводит к четвертому правилу обозначения сигналов. Если нет более наглядного названия, то входной сигнал дискретной системы обычно называется $x[n]$, а выходной сигнал $y[n]$. Для непрерывных систем в качестве обозначения сигналов используют $x(t)$ и $y(t)$.

Существует множество разных причин, для того чтобы возникло желание понять

систему. Например, Вы можете захотеть спроектировать систему для удаления шума в электрокардиограмме, сделать более резким изображение находящееся не в фокусе или удалить из звукозаписи эхо. В других случаях, система может вносить искажения или обладать интерференционным эффектом, который Вам нужно охарактеризовать или измерить. Например, когда Вы говорите по телефону, Вы ожидаете, что другой человек будет слышать что-то похожее на ваш голос. К сожалению, входной сигнал у передающей линии редко идентичен выходному сигналу. Если Вы понимаете, как передающая линия (система) изменяет сигнал, возможно Вы сумеете компенсировать ее влияние. В некоторых случаях система может представлять какой-нибудь физический процесс, который Вы хотите изучить или проанализировать. Хорошим примером этому являются радары и гидролокаторы. Для нахождения характеристик удаленных объектов, здесь сравниваются посланные и отраженные сигналы. В терминах теории систем задача состоит в том, чтобы найти систему, преобразующую переданный сигнал в полученный.

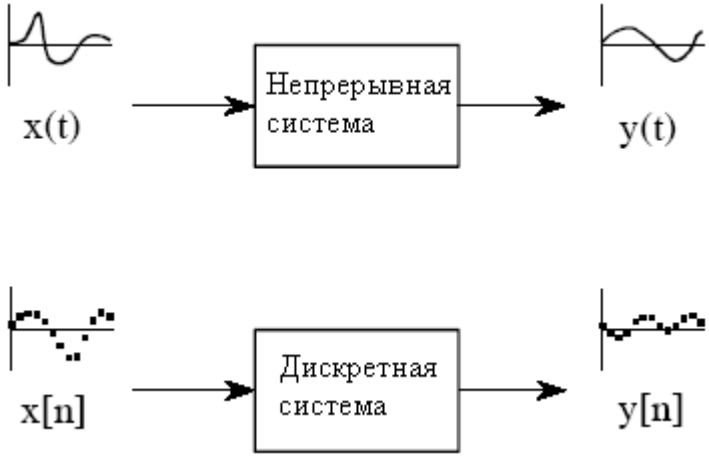


Рис. 5.1 Терминология для сигналов и систем

На первый взгляд задача, понять все возможные системы в мире, может показаться удручающей. К счастью, большинство используемых систем попадают в категорию, называемую **линейные системы**. Этот факт чрезвычайно важен. *Без* концепции линейных систем мы были бы вынуждены исследовать индивидуальные характеристики множества несвязанных систем. *Обладая* такой концепцией, мы можем сосредоточиться на свойствах категории линейных систем в целом. Наша первая задача – идентифицировать, какие свойства делают систему линейной, и как они вписываются в повседневные представления электроники, программного обеспечения и других систем обработки сигнала.

Условия обеспечения линейности

Система называется *линейной*, если ей присущи два математических свойства: **однородность** и **аддитивность**. Если Вы можете показать, что система обладает двумя этими свойствами, значит Вы доказали что система линейна. Аналогично, если Вы можете показать, что система не имеет одного или обоих свойств, Вы доказали что, она нелинейная. Третье свойство - **инвариантность сдвига**, не является строгим требованием линейности, но оно является обязательным свойством для большинства методов ЦОС. Когда Вы видите использование в ЦОС термина *линейная система*, Вы должны предполагать, что он содержит *инвариантность сдвига* до тех пор, пока у вас не появится веская причина поверить обратному. Эти три свойства формируют математику того, как определяется и используется теория линейной системы. Позже в этой главе мы посмотрим

на более интуитивные способы понимания линейности. А пока давайте пройдемся по этим формальным математическим свойствам.

Как проиллюстрировано на рис. 5.2, однородность означает, что изменения в амплитуде входного сигнала приводят к соответствующим изменениям в амплитуде выходного сигнала. Выражаясь математическим языком, если входной сигнал $x[n]$ приводит в результате к выходному сигналу $y[n]$, то для любого входного сигнала и постоянной k, входной сигнал $kx[n]$ приводит в результате к выходному сигналу $ky[n]$.

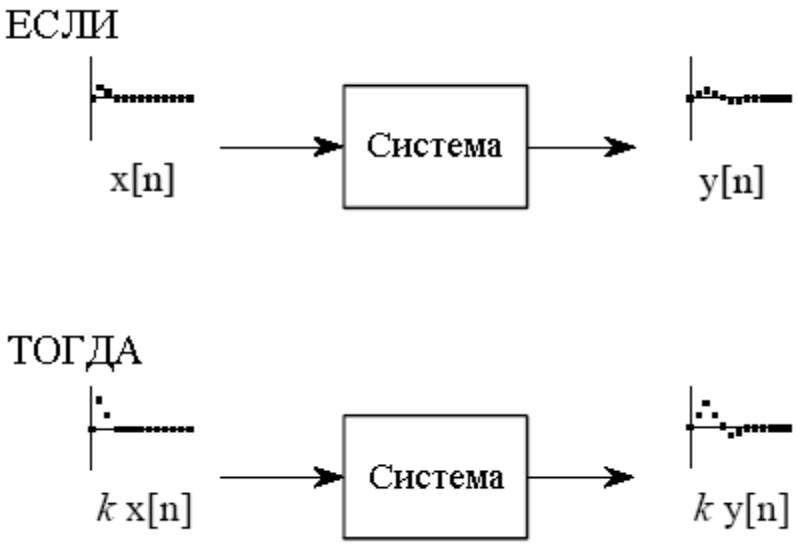


Рис. 5.2 Определение однородности

Хороший пример однородности и неоднородности системы дает простой резистор. Если входным сигналом системы является падение напряжения на резисторе $v(t)$, а выходным сигналом системы является ток, протекающий через резистор $i(t)$, то система является однородной. Это обеспечивается законом Ома: если напряжение увеличивается или уменьшается, то будет иметь место, соответствующее увеличение или уменьшение тока. Теперь рассмотрим другую систему, где входным сигналом является падение напряжения на резисторе $v(t)$, а выходным сигналом является рассеиваемая резистором мощность $p(t)$. Поскольку мощность пропорциональна квадрату напряжения, то если входной сигнал увеличивается в *два* раза, выходной сигнал увеличивается в *четыре* раза. Эта система неоднородна и, следовательно, не может быть линейной.

Свойство аддитивности иллюстрируется на рис. 5.3. Рассмотрим систему, в которой входной сигнал $x_1[n]$ приводит к появлению на выходе сигнала $y_1[n]$. Далее предположим, что другой входной сигнал $x_2[n]$ приводит к появлению на выходе сигнала $y_2[n]$. Говорят, что система *аддитивна*, если для всех возможных входных сигналов входной сигнал $x_1[n]+x_2[n]$ приводит к появлению на выходе сигнала $y_1[n]+y_2[n]$. Выражаясь словами, суммирование входных сигналов приводит к суммированию выходных сигналов.

Важным моментом является то, что *добавленные* сигналы проходят через систему без взаимодействия друг с другом. В качестве примера поразмышляйте о телефонном разговоре с вашей тетушкой Эдной или дядюшкой Берни. Тетя Эдна начинает довольно длинный рассказ о том, как хорошо растет в этом году ее редиска. На заднем плане, дядя Берни кричит на собаку, совершившую проступок на его любимом кресле. Два голосовых сигнала суммируются и в электронном виде передаются по телефонной сети. Поскольку эта система является аддитивной, Вы слышите звук, являющийся суммой двух голосов, так же, как если бы они звучали, если бы передавались раздельно. Вы слышите *Эдну* и *Берни*, а не существо *Эднаберни*.

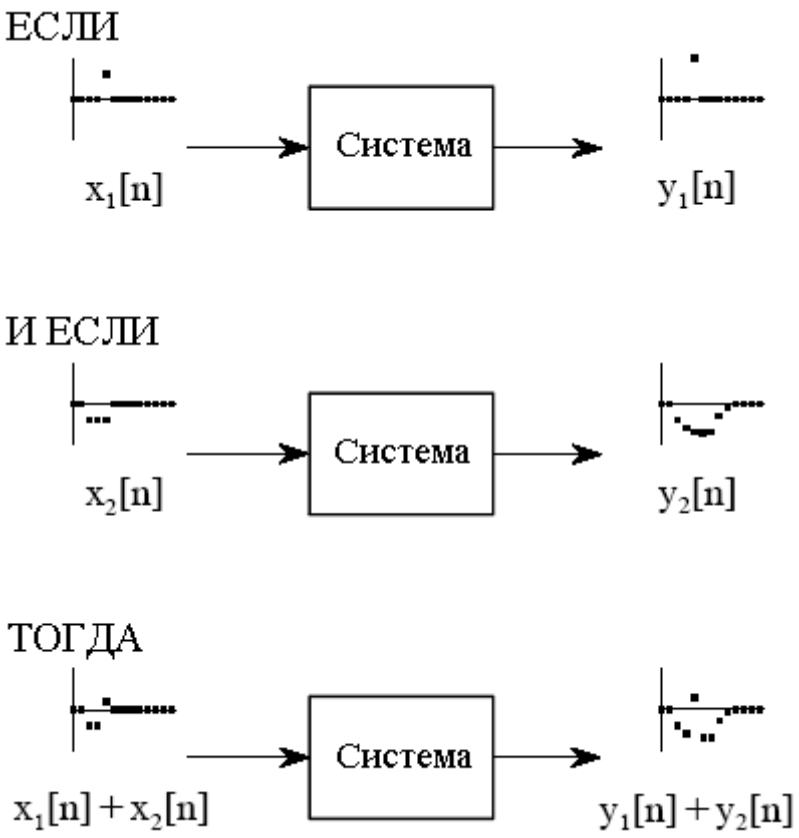


Рис. 5.3 Определение аддитивности

Хорошим примером *неаддитивной* схемы является модулятор радиопередатчика. Имеются два сигнала: звуковой сигнал, содержащий голос или музыку, и несущее колебание, которое будучи поданным на антенну, может распространяться в пространстве. Два сигнала суммируются и прикладываются к нелинейности наподобие p-n перехода диода. Это приводит к *слиянию* сигналов и формированию третьего сигнала модулированной радиоволны, способной переносить информацию на огромные расстояния.

Как показано на рис. 5.4, инвариантность сдвига означает, что сдвиг входного сигнала не приведет ни к чему более, чем к идентичному сдвигу выходного сигнала. Говоря более формальным языком, если входной сигнал $x[n]$ приводит к выходному сигналу $y[n]$, то для любых входных сигналов и любых значений константы s, входной сигнал $x[n+s]$ приводит к выходному сигналу $y[n+s]$. Обратите особое внимание на математическую запись этого сдвига, она будет использоваться в следующих главах. Добавлением константы s к независимой переменной n форма волны может быть сдвинута в горизонтальном направлении вперед или назад. Например, когда $s = 2$, сигнал сдвигается *влево* на два отсчета, когда $s = -2$, сигнал сдвигается *вправо* на два отсчета.

Инвариантность сдвига важна потому, что она означает неизменность характеристик системы во времени (или неизменность характеристик системы от значений, принимаемых независимой переменной). Если *пип* на входе вызывает *пап* на выходе, то можете быть уверены, другой *пип* вызовет точно такой же *пап*. Большинство систем, с которыми Вы столкнетесь, будут обладать инвариантностью сдвига. И это счастье, потому что иметь дело с системами, которые изменяют свои характеристики во время работы, достаточно трудно. Например, представьте, что Вы спроектировали цифровой фильтр для компенсации в передающей телефонной линии эффектов ухудшающих рабочие характеристики. Ваш фильтр делает голос более естественным и легче разбираемым. К большому вашему удивлению, с приходом зимы Вы обнаружили,

что из-за изменения температуры характеристики телефонной линии изменились. Ваш компенсационный фильтр теперь ничему не соответствует, и работает не особенно хорошо. Эта ситуация может потребовать более сложного алгоритма, который умеет *адаптироваться* к изменяющимся условиям.

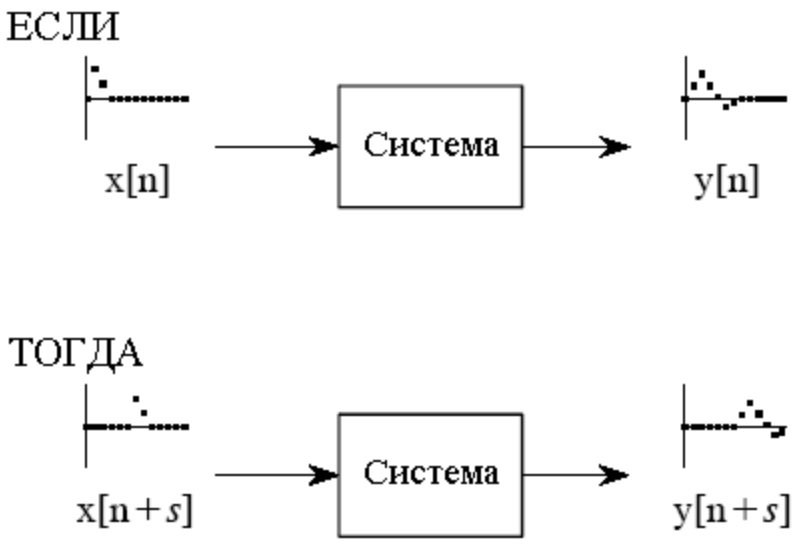


Рис 5.4 Определение инвариантности сдвига

Почему однородность и аддитивность играют в линейности критическую роль, в то время как инвариантность сдвига является чем-то, находящимся в стороне? Это потому, что линейность является очень широкой концепцией, затрагивающей намного больше, чем просто сигналы и системы. Например, рассмотрите фермера, продающего апельсины по два доллара за корзинку, и яблоки по пять долларов за корзинку. Если фермер продает только апельсины, он получит двадцать долларов за десять корзинок и сорок долларов за двадцать корзинок, делая обмен *однородным*. Если он продает двадцать корзинок апельсин и десять корзинок яблок, фермер получит: 20×$2+10×$5 = $90. Это точно такая же величина, как если бы оба товара были проданы по отдельности, делая сделку *аддитивной*. Будучи и однородной и аддитивной, такая продажа товаров является линейным процессом. Однако, поскольку сюда не вовлечены никакие сигналы, это не является *системой*, и *инвариантность сдвига* не имеет никакого значения. Об инвариантности сдвига можно думать, как о дополнительном аспекте линейности, необходимой, когда в процесс вовлечены сигналы и системы.

Статическая линейность и синусоидальное качество

Однородность, аддитивность и инвариантность сдвига важны потому, что они обеспечивают математический базис для определения линейных систем. К сожалению одни эти свойства не обеспечивают большинство ученых и инженеров интуитивным чувством относительно того, что представляют собой линейные системы. Часто здесь помогают свойства **статической линейности** и **синусоидальное качество**. Они не особенно важны с математической точки зрения, но имеют отношение к тому, как люди мыслят о линейных системах и понимают их. Вы должны обратить особое внимание на этот раздел.

Статическая линейность определяет реакцию линейной системы на не изменяющиеся сигналы, т.е. когда они являются сигналами постоянного тока или *статическими* сигналами. Статический отклик линейной системы очень прост: *выходной сигнал является входным сигналом*, *умноженным на некоторую константу*. То есть,

график зависимости соответствующих выходных значений от возможных входных значений - это прямая линия, проходящая через начало координат. Для двух обычных линейных систем это показано на рис. 5.5: закон Ома для резисторов, и закон Гука для пружин. Для сравнения на рис. 5.6 показаны статические соотношения для двух нелинейных систем: p-n перехода диода и магнитных свойств железа.

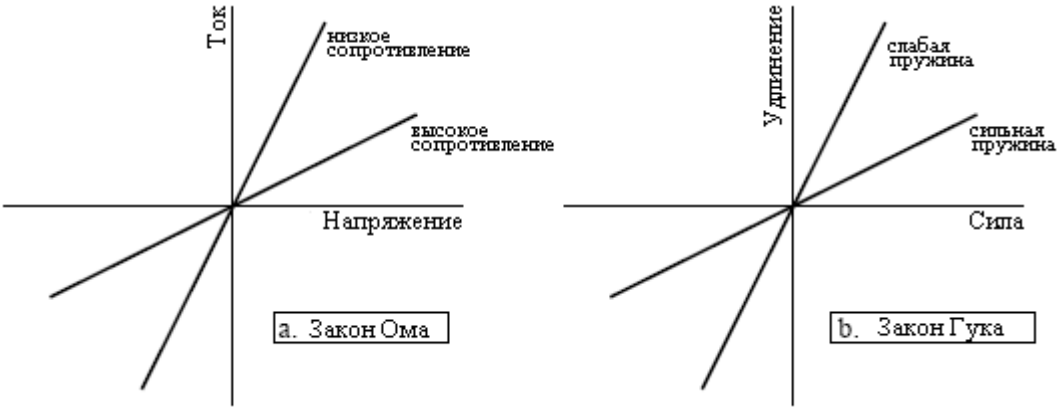


Рис. 5.5 Два примера статической линейности

Все линейные системы обладают свойством *статической линейности*. Обычно справедливо и обратное, но не всегда. Существуют системы, которые обнаруживают статическую линейность, но не являются линейными по отношению к изменяющимся сигналам. Однако очень широкий класс систем может быть полностью понят с помощью только одной статической линейности. В таких системах не имеет значения, является ли входной сигнал статическим или изменяется. Они называются системами **без памяти**, это потому, что сигнал на их выходе зависит не от предыдущих, а только от текущего состояния сигнала на их входе. Например, мгновенные значения тока в резисторе зависят только от приложенных к нему мгновенных значений напряжения, а не от того, каким образом поступающие сигналы достигли того значения, которое они сейчас имеют. Если система обладает статической линейностью и является системой без памяти, тогда система должна быть линейной. Это дает важный путь в понимании (и доказательстве) линейности таких простых систем.

Рис. 5.6 Два примера нелинейности

Важной характеристикой линейных систем является то, как они поступают с синусоидами, свойство, которое мы будем называть **синусоидальным качеством**: *Если на вход линейной системы подается синусоидальная волна, на выходе системы также*

будет синусоидальная волна и точно с такой же частотой как на входе. Синусоида единственная форма волны, которая обладает этим свойством. Например, нет ни одной причины, по которой следует ждать, что прямоугольная волна, войдя в линейную систему, даст прямоугольную волну на ее выходе. Хотя синусоида на входе гарантирует синусоиду на выходе, обе они могут отличаться по *амплитуде* и *фазе*. Это должно быть вам знакомо из знаний по электронике: цепь может быть описана ее *частотными характеристиками*, графиками того, как изменяются с частотой усиление и фазовый сдвиг цепи.

Ну а теперь к обратному вопросу: Будет ли система гарантированно линейной, если в ответ на синусоидальный входной сигнал система всегда дает синусоидальный выходной сигнал? Ответом будет - нет, но такие исключения бывают редкими и обычно всегда очевидны. Например, представьте злого демона, скрывающегося внутри системы с целью попытаться ввести Вас в заблуждение. У демона есть осциллограф для наблюдения входного сигнала и генератор синусоидальных колебаний для вырабатывания выходного сигнала. Когда Вы подаете синусоидальный сигнал на вход, демон быстро измеряет частоту и регулирует свой генератор сигнала, чтобы выработать соответствующий выходной сигнал. Конечно же, эта система не является линейной, поскольку она не аддитивна. Чтобы показать это, подайте сумму двух синусоид на вход системы. Демон может ответить только одной единственной синусоидой на выходе. Этот пример не такая уж и выдумка, как Вы можете подумать; *петли захвата фазы* во многом работают подобным образом.

Для того чтобы лучше почувствовать линейность, поразмышляйте о технике, пытающемся определить, является ли линейным некоторый электронный прибор. Техник подключит к входу прибора генератор синусоидальных колебаний и осциллограф к его выходу. Подав синусоидальный сигнал на вход, техник попытается посмотреть, является ли сигнал на выходе тоже синусоидальным. Например, выходной сигнал не должен быть срезан на вершине или у основания, верхняя половина не должна выглядеть отличной от нижней половины, в месте перехода сигнала через ноль не должно быть никаких искажений и т.д. Затем техник будет изменять амплитуду сигнала на входе, и смотреть, как это сказывается на сигнале на выходе. Если система линейна, амплитуда сигнала на выходе будет отслеживать амплитуду сигнала на входе. В конце техник будет изменять частоту входного сигнала и проверять то, что частота выходного сигнала изменяется соответствующим образом. По мере изменения частоты, возможно, будут наблюдаться изменения амплитуды и фазы выходного сигнала, но такие изменения, безусловно, допустимы в линейных системах. На некоторых частотах выходной сигнал может быть даже равен *нулю*, т.е. синусоида с нулевой амплитудой. Если техник будет наблюдать все эти вещи, он сделает заключение, что система является линейной. Несмотря на то, что это заключение не является строгим математическим доказательством, уровень его доверительности оправданно высок.

Примеры линейных и нелинейных систем

В таблице 5.1 представлены примеры обычных линейных и нелинейных систем. По мере просмотра этого перечня вспоминайте о математическом взгляде на линейность (*однородность*, *аддитивность* и *инвариантность сдвига*) точно так же, как это делают неформальным образом большинство ученых и инженеров (*статическая линейность* и *синусоидальная точность*).

Таблица 5.1

Примеры *линейных* систем

Распространение волн, таких как звуковые или радиоволны
Электрические схемы, состоящие из резисторов конденсаторов и индуктивностей

Электронные схемы, такие как усилители и фильтры
Механическое движение, вызванное взаимодействием масс, пружин и рессор (демпферов)
Системы, описываемые дифференциальными уравнениями наподобие резисторно- конденсаторно-индуктивных цепей
Умножение на константу, означающее усиление или ослабление сигнала
Изменения сигнала, такие как эхо, резонанс и смазанность изображения
Единичная система, где выходной сигнал всегда равен входному
Нулевая система, где выходной сигнал всегда равен нулю, независимо от входного сигнала
Дифференцирование и интегрирование, а также аналогичные операции первой разности и бегущей суммы для дискретных сигналов
Малые возмущения нелинейной в других отношениях системы, например, малый сигнал, усиливаемый должным образом смещенным транзистором
Свертка, математическая операция, где каждое выходное значение выражается суммой входных значений, умноженных на набор взвешенных коэффициентов
Рекурсия, техника подобная свертке, за исключением того, что предварительно вычисленные выходные значения используются для сложения с входными значениями

Примеры *нелинейных* систем

Системы, не обладающие статической линейностью, например, напряжение и мощность на резисторе: $P=V^2R$, эмиссия лучистой энергии нагретого объекта, зависящая от его температуры: $R=kT^4$,интенсивность света, передаваемого через толщу прозрачного материала: $I=e^{-\alpha T}$ и т.д.
Системы, не обладающие синусоидальным качеством наподобие электронных схем: пиковых детекторов, возведения в квадрат, преобразования синусоидальных волн в прямоугольные волны, удвоения частоты и т.д.
Обычные электронные искажения, такие как отсечки, перекрестные и фазовые искажения
Умножение одного сигнала на другой подобно амплитудной модуляции и автоматической регулировке усиления
Явления гистерезиса подобные зависимости плотности магнитного потока от напряженности магнитного поля в железе или механического напряжения от растяжения в вулканизированном каучуке
Насыщение подобное тому, которым так трудно управлять в электронных усилителях и трансформаторах
Системы с порогом, например, цифровые логические элементы или сейсмические колебания достаточно сильные, чтобы рассыпать находящуюся на их пути скалу

Специальные свойства линейности

Линейность **коммутативна**, свойство, включающее комбинацию двух или более систем. Основная идея показана на рис. 5.7. Представьте две **последовательно** соединенные системы, т.е. выход одной системы является входом следующей системы. Если каждая из систем линейна, то общая комбинация этих систем также будет линейной. Свойство коммутативности утверждает, что положение систем в последовательном соединении может быть изменено без изменения характеристик всей комбинации. Вероятно, Вы уже пользовались этим принципом в электронных схемах. Например, представьте схему, состоящую из двух каскадов, один из которых служит для усиления, а другой для осуществления фильтрации. Что лучше, усиливать, а затем осуществлять фильтрацию или фильтровать, а затем усиливать? Если оба каскада линейны, порядок не имеет значения, и конечный результат будет тем же самым. Помните, что реальная электроника обладает *нелинейными* эффектами, которые делают порядок расположения важным, например: интерференция, смещение по постоянному току, внутренний шум, фазовые искажения и т.д.

На рис. 5.8 показан следующий шаг в теории линейных систем: системы с несколькими входами и выходами. Система с несколькими входами и/или выходами будет линейной, если она состоит из *линейных подсистем* и *суммирований сигналов*. Сложность не имеет значения, единственно, что внутри системы не допускается никаких *нелинейностей*.

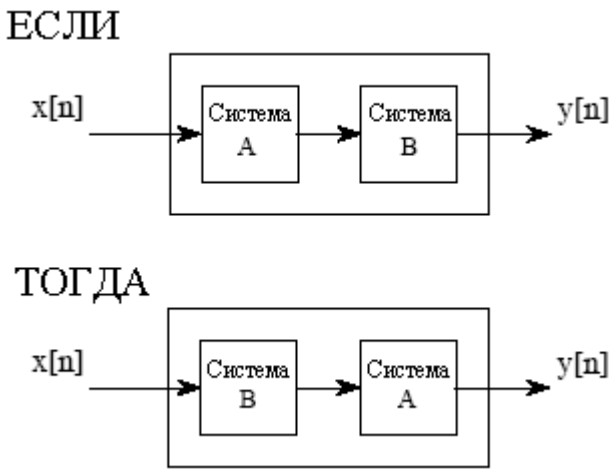


Рис. 5.7 Свойство коммутативности для линейных систем

Для того, что бы понять, что означает линейность для систем с несколькими входами и/или выходами, рассмотрим следующий мысленный эксперимент. Начнем с того, что подадим сигнал на один из входов, в то время как другие входы будем удерживать в нуле. Это вызовет появление на выходах нескольких образцов откликов сигналов. Затем повторим процедуру, поместив другой сигнал на другой вход. Так же как и раньше будем удерживать все остальные входы в нуле. Этот второй сигнал приведет к появлению другого образца сигналов на выходах. Для завершения эксперимента поместим оба сигнала на их соответствующие входы одновременно. Сигналы появившиеся на выходах будут просто *суперпозицией* (суммой) выходных сигналов, произведенных тогда, когда входные сигналы прикладывались раздельно.

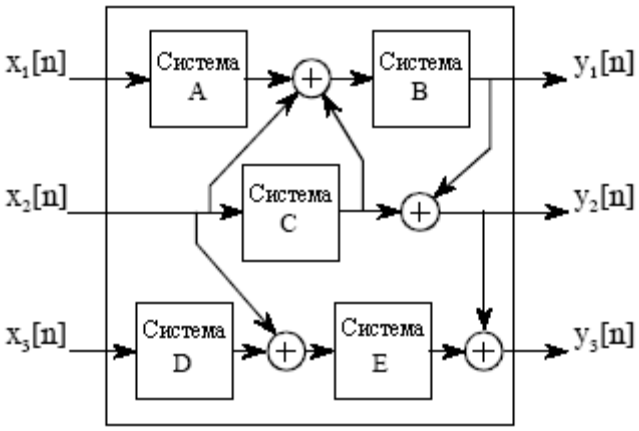
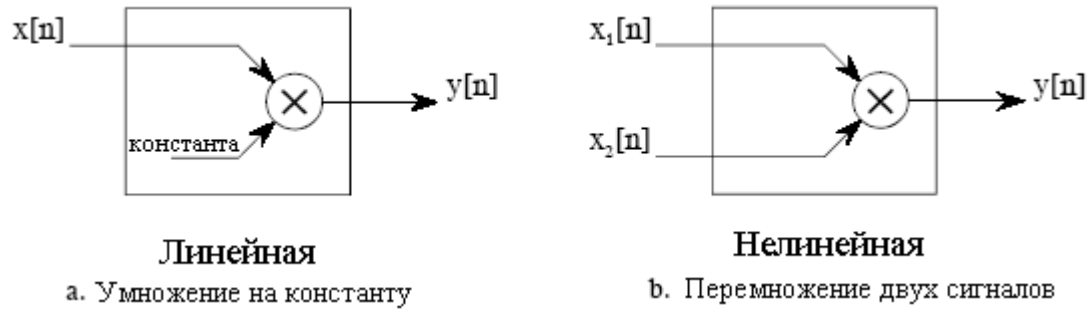


Рис. 5.8 Системы с несколькими входами и выходами

Использование умножения в линейных системах часто истолковывается неправильно. Это связано с тем, что умножение может быть как линейным, так и нелинейным в зависимости от того, с чем перемножается сигнал. На рис. 5.9 иллюстрируются оба случая. Система, которая умножает входной сигнал на *константу*, является линейной. Такими системами являются усилитель или аттенюатор в зависимости от того, соответственно, больше или меньше значение константы, чем единица. Напротив, умножение одного сигнала на *другой сигнал* является нелинейным. Представьте одну синусоиду, умноженную на вторую синусоиду с другой частотой, ясно, что результирующая форма волны будет несинусоидальной.



Линейная
a. Умножение на константу

Нелинейная
b. Перемножение двух сигналов

Рис. 5.9 Линейность умножения

Другая, обычно неправильно истолковываемая ситуация, касается добавляемых в электронике паразитных сигналов, наподобие смещения по постоянному току и теплового шума. Является ли линейным или нелинейным добавление таких посторонних сигналов? Ответ зависит от того, в каком месте, как источник, рассматриваются загрязняющие сигналы. Если они рассматриваются как возникающие *внутри* системы, процесс нелинеен. Это связано с тем, что синусоидальный входной сигнал не дает чистый синусоидальный сигнал на выходе. Напротив, посторонний сигнал может рассматриваться как входящий по отдельному входу сигнал *извне* в систему с несколькими входами. Это делает процесс линейным, так как в пределах системы требуется только сложение сигнала.

Суперпозиция: базис ЦОС

Когда мы имеем дело с линейными системами, единственным способом, с помощью которого могут быть объединены сигналы, является *масштабирование* (умножение сигналов на константу), сопровождаемое *сложением*. Например, сигнал не может быть умножен на другой сигнал. На рис. 5.10 показан пример сложения трех сигналов $x_0[n]$, $x_1[n]$ и $x_2[n]$ для формирования четвертого сигнала $x[n]$. Такой процесс объединения сигналов посредством масштабирования и сложения называется **синтезом**.

Декомпозиция является операцией обратной синтезу, в которой отдельный сигнал разделяется на две или более слагаемых компоненты. Декомпозиция является более емкой, чем синтез, поскольку для любого заданного сигнала существует бесконечное число возможных декомпозиций. Например, числа 15 и 25 могут быть синтезированы (сложены) только в число 40. Для сравнения, декомпозиция числа 40 может быть представлена как: 1+39 или 2+38 или –30,5+60+10,5 и т.д.

Ну а сейчас мы подходим к сердцу ЦОС, **суперпозиции,** всеобщей стратегии для понимания того, каким образом могут быть проанализированы сигналы и системы. Рассмотрим входной сигнал, обозначаемый как $x[n]$, проходящий через линейную систему и дающий на выходе сигнал $y[n]$. Как иллюстрируется на рис. 5.11 входной сигнал может быть разложен на группу более простых сигналов: $x_0[n]$, $x_1[n]$, $x_2[n]$ и т.д. Мы будем называть их **компонентами входного сигнала**. Затем, каждая компонента входного сигнала индивидуально пропускается через систему, в результате чего получается набор **компонент выходного сигнала**: $y_0[n]$, $y_1[n]$, $y_2[n]$ и т.д. После чего из этих компонент выходного сигнала синтезируется выходной сигнал $y[n]$.

А сейчас важная часть: выходной сигнал, полученный таким способом, *идентичен* сигналу, полученному с помощью непосредственного прохождения входного сигнала через систему. Это - очень мощная идея. Вместо того чтобы пытаться понять, как изменяются системой *сложные* сигналы, все, что нам нужно знать, так это то, как изменяются системой простые сигналы. На жаргоне обработки сигналов входные и выходные сигналы рассматриваются как *суперпозиция* (сумма) простейших форм волн. Это основа почти всех методов обработки сигналов.

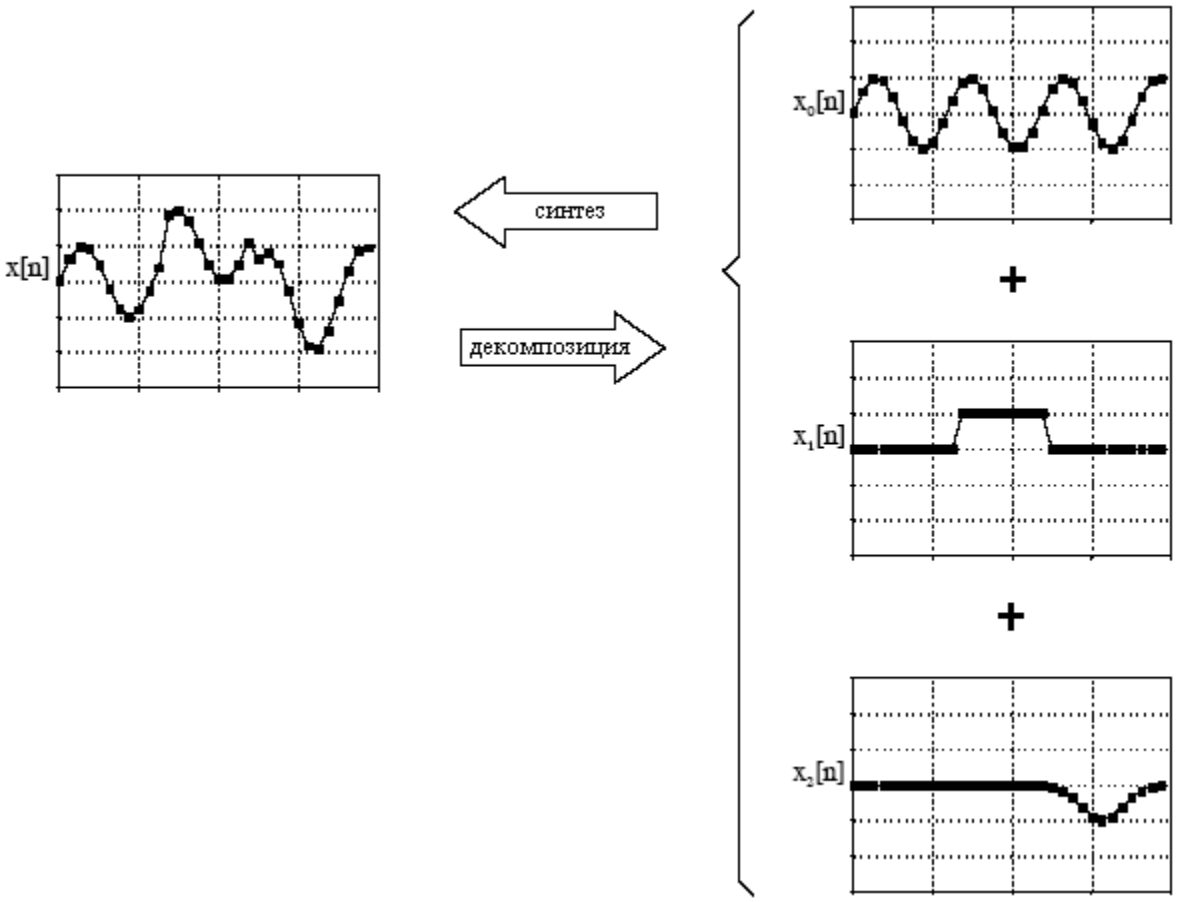


Рис. 5.10 Иллюстрация синтеза и разложения сигналов

В качестве простого примера использования суперпозиции умножьте число 2041 на число 4 в уме. Как Вы это сделали? Вы могли бы вообразить 2041 спичку, проплывающую в ваших мыслях, учетверить мысленный образ и начать считать. Гораздо более удобно, если Вы для упрощения задачи воспользуетесь суперпозицией. Для числа 2041 может быть осуществлена декомпозиция на: 2000+40+1. Каждая из этих компонент может быть умножена на 4 и затем, для нахождения окончательного ответа, произведен синтез, т.е. 8000+160+4=8164.

Обычная декомпозиция

Имейте в виду, что цель данного метода состоит в том, чтобы заменить сложную задачу несколькими более легкими. Если декомпозиция ни коим образом не упрощает ситуацию, тогда никакого выигрыша не будет. В обработке сигналов существует два основных способа декомпозиции сигнала: *импульсная декомпозиция* и *декомпозиция Фурье*. Они детально описываются в нескольких следующих главах. Кроме того, иногда используются несколько малозначительных декомпозиций. Здесь же вкратце описываются две основные декомпозиции вместе с тремя из малозначительных.

Импульсная декомпозиция

Как показано на рис. 5.12, импульсная декомпозиция разбивает сигнал, содержащий *N* отсчетов, на *N* компонент сигнала, каждая из которых содержит по *N* отсчетов. Каждая из компонент сигнала содержит одну точку из оригинального сигнала, при этом все остальные значения равны нулю. Единственная ненулевая точка в последовательности нулей называется *импульсом*. Импульсная декомпозиция очень

важна, поскольку она позволяет исследовать сигналы с точки зрения одного отсчета в данный момент времени. Подобным образом, системы характеризуются тем, как они отвечают на воздействие импульса. Зная, как система отвечает на импульс, можно вычислить выходной сигнал системы для любого заданного входного сигнала. Этот подход называется *сверткой* и является темой следующих двух глав.

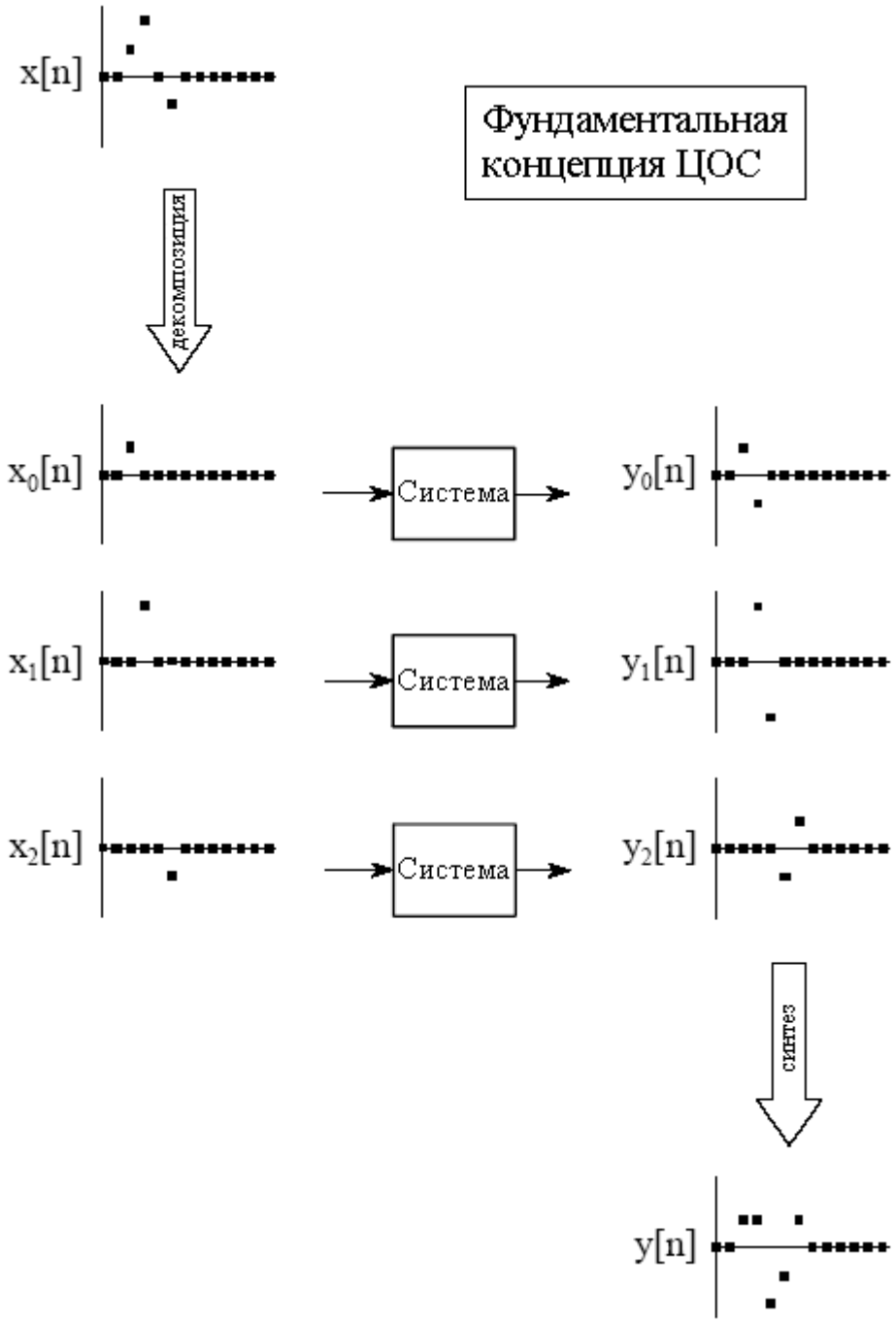


Рис. 5.11 Фундаментальная концепция ЦОС

Ступенчатая декомпозиция

Ступенчатая декомпозиция, показанная на рис. 5.13, также разбивает сигнал, содержащий N отсчетов, на N компонент сигнала, каждая из которых содержит по N отсчетов. Каждая компонента сигнала является *ступенькой*, т.е. первые отсчеты имеют значение нуля, в то время как последние отсчеты имеют некоторое постоянное значение. Рассмотрим декомпозицию N точечного сигнала $x[n]$ на компоненты: $x_0[n]$, $x_1[n]$, $x_2[n]$, …,

$x_{N-1}[n]$. k – я компонента сигнала $x_k[n]$ состоит из нулей от точки *0* по точку *k-1*, в то время как остальные точки имеют значение $x[k]$-$x[k-1]$. Например, 5-я компонента сигнала $x_5[n]$ состоит из нулей для точек с *0* по *4*, в то время как остальные отсчеты имеют значение: $x[5]$-$x[4]$ (разница между отсчетом 4 и 5 оригинального сигнала). Как частный случай все отсчеты $x_0[n]$ равны $x[0]$. Так же, как импульсная декомпозиция рассматривает сигналы с точки зрения одного отсчета в данный момент времени, ступенчатая декомпозиция характеризует сигналы *разностью* между соседними отсчетами. Аналогичным образом системы характеризуются тем, как они отвечают на *изменения* во входном сигнале.

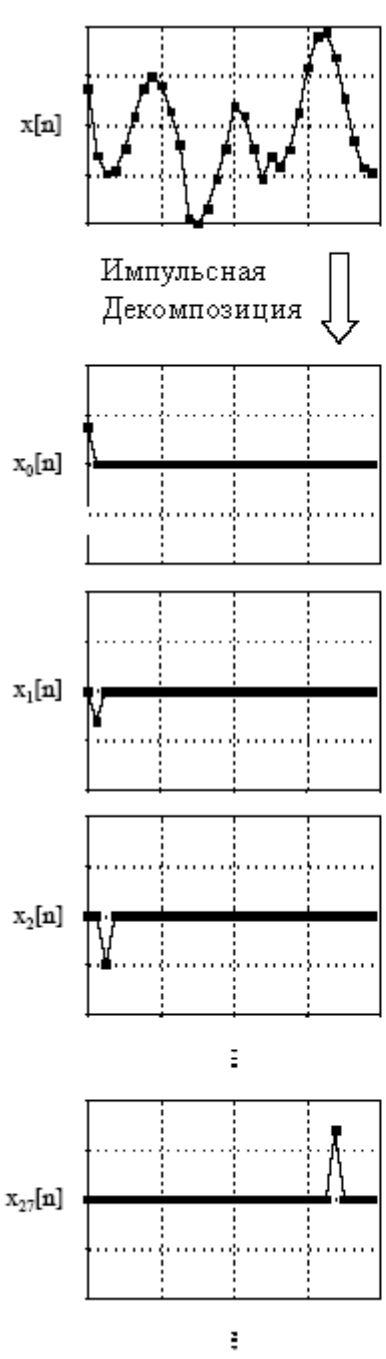


Рис. 5.12 Пример импульсной декомпозиции

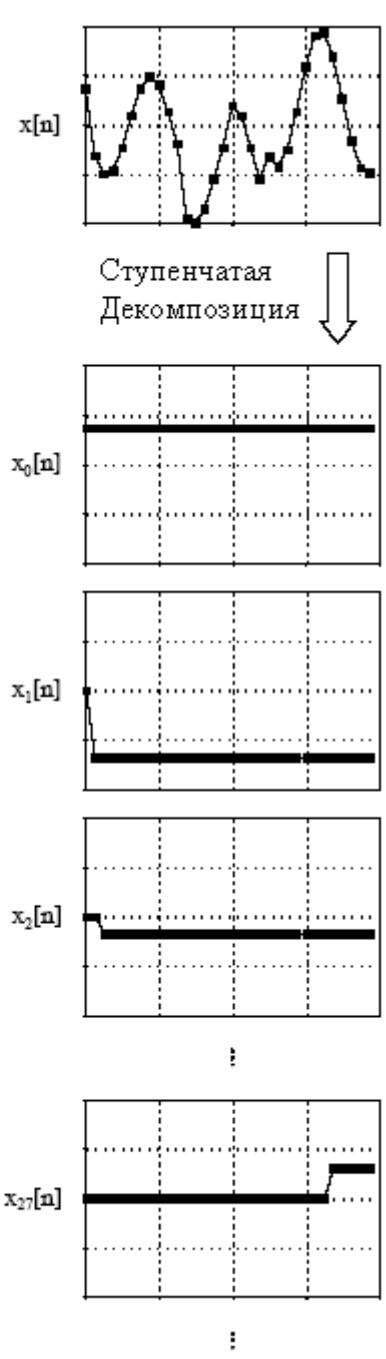


Рис. 5.13 Пример ступенчатой декомпозиции

Четная/нечетная декомпозиция

Четная/нечетная декомпозиция, показанная на рис. 5.14, разбивает сигнал на две компоненты, одна из них имеет **четную симметрию**, а другая имеет **нечетную симметрию**. Говорят, что *N* точечный сигнал обладает четной симметрией, если он зеркально отображается относительно точки *N/2*. То есть, отсчет *x*[*N/2 + 1*] должен быть равен отсчету *x*[*N/2 - 1*], отсчет *x*[*N/2 + 2*] должен быть равен отсчету *x*[*N/2 - 2*] и т.д. Подобным образом нечетная симметрия наблюдается тогда, когда соответствующие точки имеют равные значения, но противоположны по знаку, наподобие как: *x*[*N/2 + 1*] = - *x*[*N/2 - 1*], *x*[*N/2 + 2*] = - *x*[*N/2 - 2*] и т.д. Эти определения предполагают, что сигнал состоит из четного числа отсчетов, и что индексы изменяются от *0* до *N-1*. Декомпозиция вычисляется в соответствии с выражениями:

$$x_E[n] = \frac{x[n] + x[N-n]}{2}$$
$$x_O[n] = \frac{x[n] - x[N-n]}{2}. \qquad (5.1)$$

Такое определение симметрии слева – направо может показаться странным, поскольку точный центр сигнала это *N/2 – ½* (находится между двумя отсчетами), а не *N/2*. Таким образом, такая симметрия от центра означает, что нулевой отсчет требует специального с ним обращения. Для чего все это нужно?

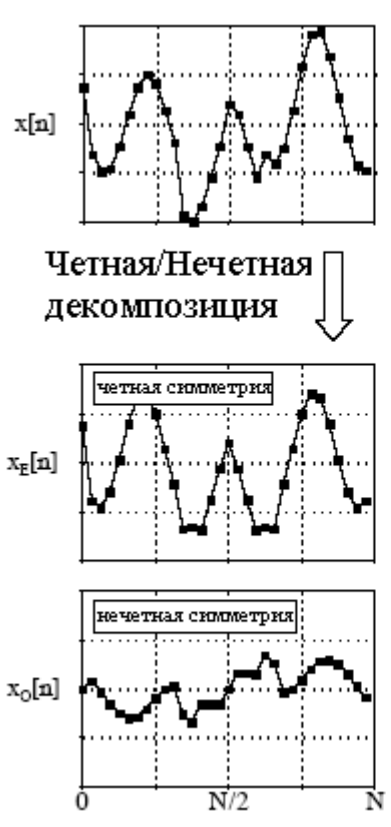


Рис. 5.14 Пример четной/нечетной декомпозиции

Рис. 5.15 Пример перекрестной декомпозиции

Эта декомпозиция часть важной концепции ЦОС, называемой **круговой симметрией**. В ее основе лежит подход, рассматривающий *конец* сигнала, как соединенный с *началом* сигнала. Так же, как точка $x[4]$ предшествует точке $x[5]$, точка $x[N-1]$ предшествует точке $x[0]$. Получается картина змеи кусающей свой собственный хвост. Когда в такой круговой манере рассматриваются четный и нечетный сигналы, фактически имеются *две* линии симметрии: одна в точке $x[N/2]$, другая в точке $x[0]$. Например, в четном сигнале такая симметрия вокруг $x[0]$ означает, что точка $x[1]$ равна точке $x[N-1]$, точка $x[2]$ равна точке $x[N-2]$ и т.д. В нечетном сигнале точка *0* и точка *N/2* всегда имеют нулевое значение. В четном сигнале точка *0* и точка *N/2* равны соответствующим точкам в исходном сигнале.

Какова мотивация для того, чтобы рассматривать последний отсчет в сигнале как отсчет, следующий за первым? В традиционном сборе данных для поддержания этого кругового понятия, ничего не существует. Фактически, первые и последние отсчеты, в общем случае в совокупности менее значимы, чем любые другие две точки в последовательности. Это следует из здравого смысла! В этих мудреных пазлах потерянной частью является техника ЦОС, называемая *Фурье анализом*. Математика Фурье анализа неотъемлемо рассматривает сигнал как круговой, хотя обычно в терминах того, откуда пришли данные, это не имеет физического смысла. Более детально мы рассмотрим это в Главе 10. А сейчас важно понять то, что уравнение (5.1) обеспечивает корректную декомпозицию просто потому, что для восстановления исходного сигнала четные и нечетные составляющие должны быть сложены вместе.

Перекрестная декомпозиция

Как показано на рис. 5.15, перекрестная декомпозиция разбивает сигнал на две составляющие сигнала: *четные отсчеты* сигнала и *нечетные отсчеты* сигнала (не путайте с четной и нечетной симметрией сигналов). Для нахождения четного отсчета сигнала начните с исходного сигнала и сделайте все его нечетные отсчеты равными нулю. Для нахождения нечетного отсчета сигнала начните с исходного сигнала и сделайте все его четные отсчеты равными нулю. Это очень просто.

На первый взгляд эта декомпозиция может показаться тривиальной и неинтересной. Но это ирония, поскольку перекрестная декомпозиция является основой для чрезвычайно важного алгоритма в ЦОС, алгоритма Быстрого Преобразования Фурье (БПФ). Процедура вычисления разложения Фурье известна несколько сотен лет. К сожалению, она катастрофически медленна, часто на существующих на сегодняшний день компьютерах для ее выполнения требуются минуты или часы. БПФ - это семейство алгоритмов, разработанных в 1960-х для снижения данного времени вычисления. Их стратегия является изящным примером ЦОС: довести сигнал до элементарных составляющих за счет повторного использования перекрестного преобразования; вычислить разложение Фурье для индивидуальных компонент; синтезировать результат в конечный ответ. В общем случае скорость увеличивается в *сотни* или *тысячи* раз - результат впечатляющий.

Декомпозиция Фурье

Декомпозиция Фурье вовсе не очевидная и чрезвычайно математическая. На рис. 5.16 показан пример такой техники. Любой N точечный сигнал может быть разложен на $N+2$ сигналов, половина из них являются синусоидальными волнами и половина косинусоидальными волнами. Самая нижняя частота косинусоидальной волны (названная на этой иллюстрации $x_{C0}[n]$) совершает за N отсчетов *ноль* полных колебаний, т.е. это постоянная составляющая. Следующие косинусные составляющие $x_{C1}[n]$, $x_{C2}[n]$ и $x_{C3}[n]$ совершают за N отсчетов, соответственно, 1, 2 и 3 полных колебания. Такие же образцы имеются и для остальных косинусоидальных волн, так же, как и для составляющих синусоидальных волн. Поскольку частота каждой составляющей является фиксированной, единственной вещью, которая изменяется для различных подвергаемых декомпозиции сигналов, является *амплитуда* каждой из синусоидальных и косинусоидальных волн.

Декомпозиция Фурье важна по трем причинам. Первое, широкое разнообразие сигналов создается непосредственно дополнительно добавленными синусоидами. Хорошим примером тому являются звуковые сигналы. Декомпозиция Фурье обеспечивает непосредственный анализ информации, содержащейся в этом типе сигналов. Второе, линейные системы отвечают на воздействие синусоиды единственным образом: синусоидальный сигнал на входе всегда дает синусоидальный сигнал на выходе. При таком подходе системы характеризуются тем, как они изменяют амплитуду и фазу проходящих через них синусоид. Поскольку входной сигнал может быть разложен на синусоиды, знание того, как система будет реагировать на синусоиды, позволяет найти выходной сигнал системы. Третье, декомпозиция Фурье является основой не только для широкой и мощной области математики, называемой *Фурье анализом*, но и даже для более превосходящих преобразований: *преобразования Лапласа* и *z – преобразования*. Большинство наиболее острых алгоритмов ЦОС основаны на некоторых аспектах этих методов.

При каких условиях можно разложить произвольный сигнал на синусную и косинусную волны? Как определяются амплитуды этих синусоид для конкретного сигнала? Какие типы систем могут быть разработаны с помощью этого метода? Ответы на эти вопросы будут даны в следующих главах. Ведь для представления в таком кратком обзоре не могут быть привлечены детали декомпозиции Фурье. А сейчас, важно понять идею, что, когда все синусоиды складываются вместе, происходит точная реконструкция исходного сигнала. Более подробно об этом в Главе 8.

Альтернативы линейности

Чтобы оценить важность линейных систем, предположите, что существует только *одна* основная стратегия для анализа нелинейных систем. Эта стратегия состоит в том, чтобы сделать нелинейную систему похожей на линейную систему. Существует три обычных способа сделать это.

Первый, игнорировать нелинейность. Если нелинейность достаточно мала, система может быть аппроксимирована как линейная. Ошибки, следующие из первоначального предположения, воспринимаются как шум или просто игнорируются.

Второй, использовать очень маленькие сигналы. Большое количество нелинейных систем проявляют линейность, если сигналы имеют очень маленькую амплитуду. Например, транзисторы являются очень нелинейными во всем их полном диапазоне работы, но обеспечивают очень точное линейное усиление, когда сигналы удерживаются около нескольких милливольт. Операционные усилители доводят эту идею до крайности. Используя очень высокое усиление в случае отсутствия обратной связи, совместно с отрицательной обратной связью входной сигнал на операционном усилителе (т.е. разница напряжений между инвертирующим и неинвертирующим входами) удерживается всего около нескольких микровольт. Этот крохотный входной сигнал приводит к достижению превосходной линейности от, иначе и не скажешь, ужасно нелинейной схемы.

Третий, применять лианеризующие преобразования. Например, рассмотрим два сигнала, которые перемножаются для получения третьего: $a[n] = b[n] \times c[n]$. Логарифмирование сигналов превращает нелинейный процесс умножения в линейный процесс сложения: $log(a[n]) = log(b[n]) + log(c[n])$. У этого подхода есть причудливое название *гомоморфная* (от греч. γομομορφη - равной формы. Гомоморфизм - это уподобление структуры, строения систем, множеств, процессов, конструкций. Не будучи симметричным отношением, гомоморфизм обосновывает перенос знаний лишь с гомоморфного образа на прообраз, но не наоборот – прим. перев.) обработка сигнала. Например, визуальное изображение может быть смоделировано, как отражающая способность сцены (двумерный сигнал) умноженная на окружающую освещенность

(другой двумерный сигнал). Гомоморфная техника дает возможность сделать сигнал освещенности более однородным, улучшая, таким образом, изображение.

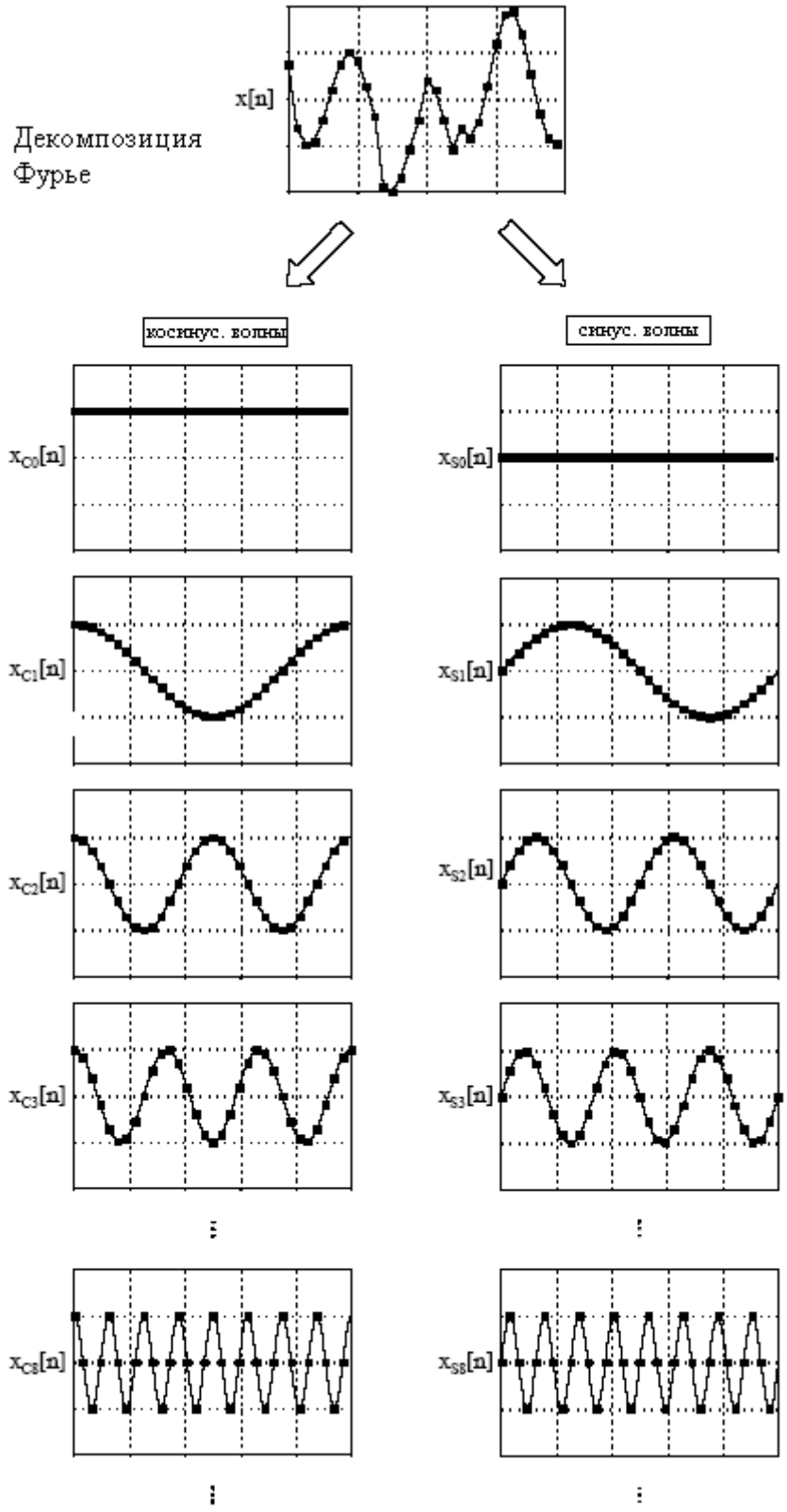


Рис. 5.16 Иллюстрация декомпозиции Фурье

В следующих главах мы исследуем два основных метода обработки сигнала: *свертку* и *Фурье анализ*. Оба они основаны на стратегии, представленной в этой главе: (1)

декомпозиция сигналов на простые аддитивные составляющие, (2) обработка составляющих некоторым полезным образом и (3) синтез составляющих в конечный результат. Это и есть ЦОС.

ГЛАВА

6

СВЕРТКА

Свертка – это математический способ объединения двух сигналов с целью получения третьего сигнала. Это отдельный наиболее важный метод цифровой обработки сигналов. Используя стратегию импульсной декомпозиции, системы описываются сигналом, называемым *импульсным откликом*. Свертка важна потому, что она связывает три интересующих сигнала: входной сигнал, выходной сигнал и импульсный отклик. Эта глава представляет свертку с двух различных точек зрения называемых алгоритмом входной стороны и алгоритмом выходной стороны. Свертка дает математическую основу для ЦОС, в этой книге нет ничего более важного.

Дельта функция и импульсный отклик

В предыдущей главе описывается то, каким образом сигнал может быть разложен на группу компонент называемых **импульсами**. Импульс – это сигнал, состоящий из всех нулей за исключением одной единственной ненулевой точки. В сущности, импульсная декомпозиция дает способ анализа сигналов по одному отсчету в конкретный момент времени. Предыдущая глава представляет также фундаментальную концепцию ЦОС: входной сигнал разлагается на простые аддитивные составляющие, затем каждая из этих составляющих пропускается через линейную систему, и получившиеся в результате выходные составляющие складываются (осуществляется синтез). Сигнал, получающийся в результате этой процедуры "разделяй и властвуй" идентичен тому, который получается от непосредственного прохождения исходного сигнал через систему. Хотя возможно большое число различных декомпозиций основу для обработки сигнала формируют две: импульсная декомпозиция и декомпозиция Фурье. При использовании импульсной декомпозиции процедура может быть описана математической операцией называемой **сверткой**. В этой главе (и в большинстве следующих) мы будем иметь дело только с *дискретными* сигналами. Свертка применяется также и к непрерывным сигналам, но в этом случае математика значительно усложняется. Как обрабатываются *непрерывные* сигналы, мы будем рассматривать в Главе 13.

Рис. 6.1 иллюстрирует два важных термина используемых в ЦОС. Первый термин - это **дельта функция**, обозначаемая греческой буквой $\delta[n]$. Дельта функция это *нормализованный* импульс, т.е. отсчет с номером ноль имеет единичное значение, тогда как все другие отсчеты имеют нулевое значение. По этой причине дельта функцию часто называют **единичным импульсом**.

Второй термин, иллюстрируемый на рис. 6.1, это **импульсный отклик**. Как предполагает название, импульсный отклик это сигнал, который выходит из системы, когда на вход системы поступает дельта функция (единичный импульс). Если две системы отличаются каким–либо образом, они будут иметь различный импульсный отклик. Так как входной и выходной сигналы часто обозначают $x[n]$ и $y[n]$, импульсный отклик обычно обозначают символом $\boldsymbol{h[n]}$. Конечно, он может быть заменен на более описательное имя, если оно доступно, например, для обозначения импульсного отклика *фильтра* может быть использовано $f[n]$.

Любой импульс может быть представлен как *сдвинутая* и *смасштабированная* дельта функция. Рассмотрим сигнал *a*[*n*], состоящий из всех нулей за исключением отсчета с номером восемь, который имеет значение *–3*. Это то же самое, что и дельта функция сдвинутая вправо на восемь отсчетов и умноженная на *–3*. В форме уравнения это будет выглядеть как: $a[n] = -3\delta[n-8]$. Убедитесь, что Вы понимаете эту систему обозначений, она используется почти во всех уравнениях ЦОС.

Каким же будет сигнал системы на выходе, если на вход системы поступает импульс, такой как $-3\delta[n-8]$? Здесь как раз то место, где следует применить свойства однородности и инвариантности сдвига. Масштабирование и сдвиг входного сигнала дают идентичные масштабирование и сдвиг выходного сигнала. Если $\delta[n]$ вызывает отклик $h[n]$, то $-3\delta[n-8]$ вызывает отклик $-3h[n-8]$. Выражаясь словами, сигнал на выходе является вариантом импульсного отклика системы *сдвинутого* и *смасштабированного* на ту же самую величину, что и дельта функция на входе. Таким образом, если Вам известен импульсный отклик системы, Вам немедленно известна ее реакция на *любой* импульс.

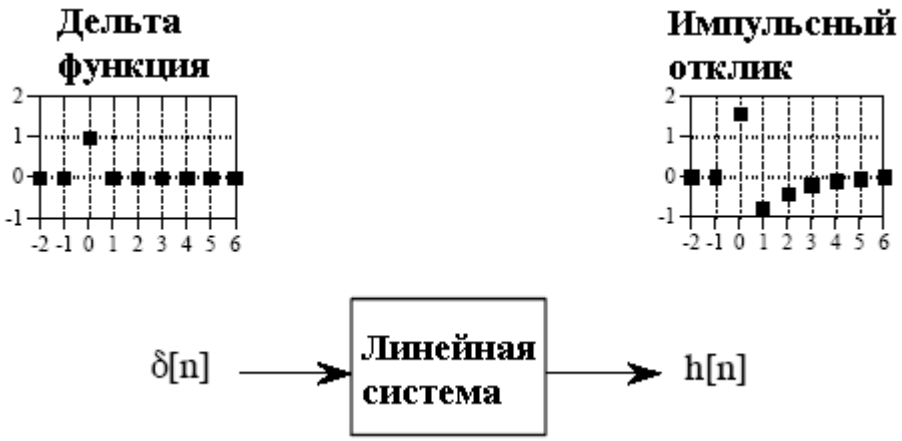


Рис.6.1 Определение дельта функции и импульсного отклика

Свертка

Обобщим путь преобразования системой входного сигнала в выходной. Прежде всего, входной сигнал должен быть представлен набором импульсов, каждый из которых может рассматриваться как смасштабированная и сдвинутая дельта функция. Затем для каждого входного импульса находится соответствующий ему смасштабированный и сдвинутый импульсный отклик системы. И, наконец, выходной сигнал находится как сумма этих смасштабированных и сдвинутых импульсных откликов. Другими словами, если мы знаем импульсный отклик системы, мы можем вычислить выходной сигнал для любого возможного входного воздействия. Это означает, что мы знаем о системе *все*, и никаких характеристик линейной системы, которые следует изучать, больше нет. (Однако в последующих главах мы покажем, что эта информация может быть представлена в различных формах.)

В различных приложениях импульсный отклик называют по-разному. Так если рассматриваемая система является *фильтром*, то импульсный отклик называют, **ядром фильтра** (**импульсной характеристикой** – прим. перев.), **ядром свертки** или просто **ядром**. В обработке изображений импульсный отклик называют **функцией размыва точки**. Несмотря на то, что эти термины используются в слегка различных назначениях, все они означают одно и тоже, сигнал, воспроизводимый системой тогда, когда на ее вход поступает дельта функция.

Свертка является формальной математической операцией такой же, как умножение, сложение и интегрирование. Сложение берет два *числа* и получает третье *число*, в то время как свертка берет два *сигнала* и получает третий *сигнал*. Свертка используется во многих областях математики, таких как теория вероятности и математическая статистика. В линейных системах свертка используется для описания взаимосвязи между тремя интересующими сигналами: входным сигналом, импульсным откликом и выходным сигналом.

Рис.6.2 Как свертка используется в ЦОС

На рис.6.2 показано используемое для линейной системы обозначение свертки. Входной сигнал $x[n]$ поступает на линейную систему с импульсным откликом $h[n]$, преобразуясь в результате в выходной сигнал $y[n]$. В форме уравнения это будет выглядеть как: $x[n]*h[n]=y[n]$. Выражаясь словами, свертка входного сигнала с импульсным откликом равна выходному сигналу. Так же как сложение обозначается знаком +, а умножение знаком ×, свертка обозначается знаком *. К сожалению, в большинстве языков программирования знак звездочки используется для обозначения операции умножения. Поэтому в компьютерной программе звездочка означает умножение, тогда как в уравнении звездочка обозначает свертку.

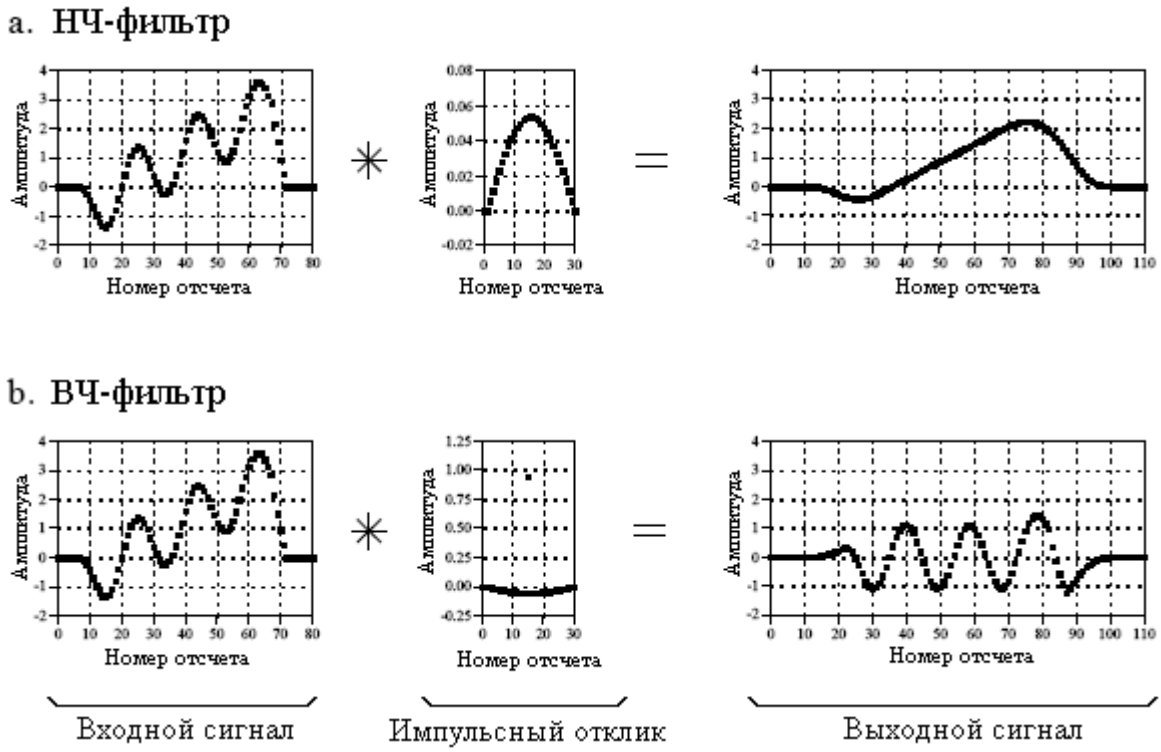


Рис.6.3 Примеры высокочастотной и низкочастотной фильтрации использующей свертку

Рис. 6.3 показывает, как свертка используется для низкочастотной и высокочастотной фильтрации. В этом примере входной сигнал является суммой двух

компонент: трех периодов синусоидальной волны (представляет высокую частоту) плюс медленно изменяющийся линейно-нарастающий участок (состоящий из нижних частот). На рис. 6.3a импульсный отклик для низкочастотного фильтра представляет собой гладкую арку, в результате на выход проходит только медленно изменяющаяся волна в форме линейно-нарастающего участка. Точно так же высокочастотный фильтр (см. рис. 6.3b) позволяет пройти только более быстро изменяющейся синусоиде.

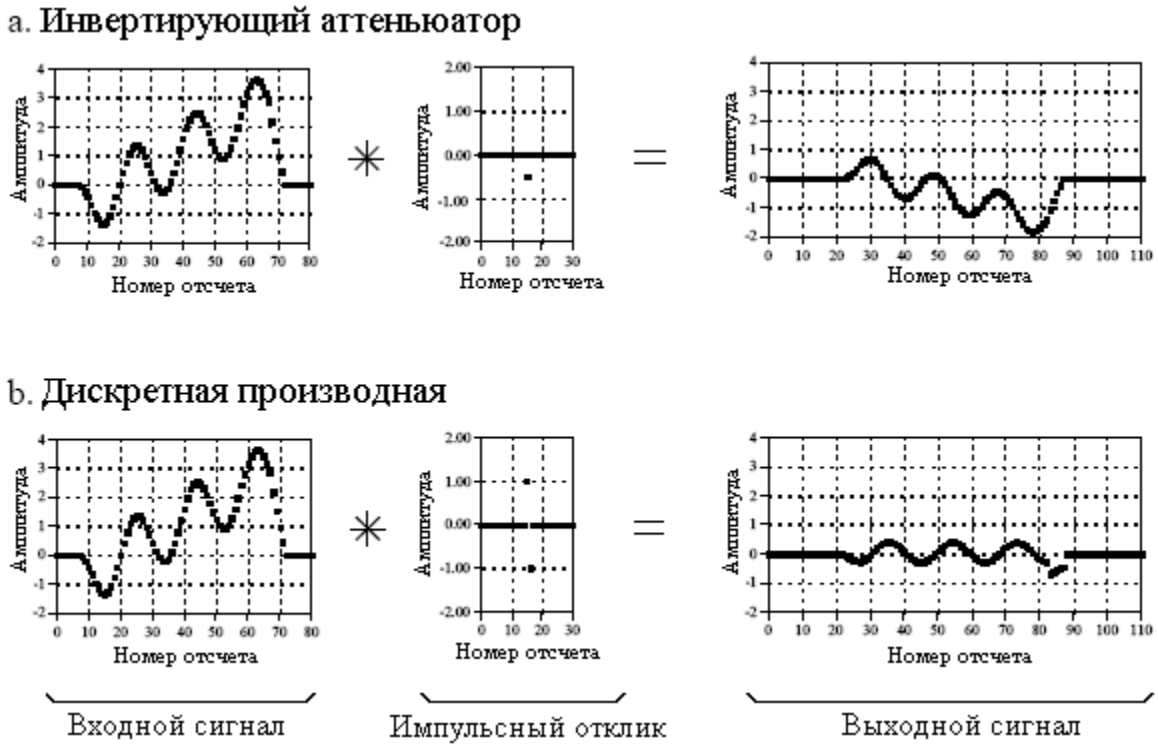


Рис.6.4. Примеры сигналов обрабатываемых с помощью свертки

Рисунок 6.4 иллюстрирует два дополнительных примера, показывающих, как свертка используется для обработки сигналов. Инвертирующий аттенюатор (рис. 6.4a) переворачивает сигнал сверху вниз и уменьшает его амплитуду. Дискретная производная (также называемая первой разностью), показанная на рис. 6.4b, исключает из выходного сигнала имеющийся во входном сигнале *наклон*.

Обратите внимание на длину сигналов, показанных на рис. 6.3 и рис. 6.4. Входной сигнал имеет длину 81 отсчет, тогда как импульсный отклик содержит 31 отсчет. В большинстве приложений ЦОС входной сигнал имеет длину в сотни, тысячи и даже миллионы отсчетов. Импульсный отклик значительно короче, скажем, в несколько точек или несколько сотен точек. Математика, на которой базируется свертка, не ограничивает длину этих сигналов. Однако она определяет длину выходного сигнала. Длина выходного сигнала равна длине входного сигнала плюс длина импульсного отклика минус единица. Для сигналов на рис.6.3 и рис. 6.4. длина каждого из выходных сигналов равна: 81+31-1=111 отсчетов. Входной сигнал определяется отсчетами с 0 по 80, импульсный отклик отсчетами с 0 по 30, и выходной сигнал определяется отсчетами с 0 по 110.

Теперь перейдем к более подробному рассмотрению математики, на которой базируется свертка. В цифровой обработке сигналов свертку рассматривают с двух различных сторон. Первый взгляд на свертку **с точки зрения входного сигнала**. Он позволяет проанализировать, какой *вклад* вносит каждый отдельный отсчет входного сигнала во множество точек выходного сигнала. Второй подход позволяет посмотреть на свертку **с точки зрения выходного сигнала**. Он показывает, как *формируется* каждый отдельный отсчет выходного сигнала из множества точек входного сигнала.

Необходимо помнить, что эти два взгляда лишь разные способы осмысления одной и той же математической операции. Первый взгляд важен, поскольку он обеспечивает *концептуальное* понимание того, каким образом свертка относится к ЦОС. Второй взгляд описывает *математику* свертки. Оба они символизируют одну из наиболее сложных задач, с которыми Вы можете столкнуться в ЦОС: увязать Ваше концептуальное понимание с математическим ералашем, используемым для объединения идей.

Алгоритм входной стороны

Рис. 6.5 иллюстрирует простую задачу свертки: входной сигнал $x[n]$, содержащий 9 точек, проходит через систему с импульсным откликом $h[n]$, состоящим из четырех точек, результатом является выходной сигнал $y[n]$ из 9+4-1=12 точек. На математическом языке $y[n]$ является результатом свертки $x[n]$ и $h[n]$. Данная первая точка зрения на свертку базируется на фундаментальной концепции ЦОС: декомпозиция входного сигнала, прохождение компонент через систему и синтез выходного сигнала. В этом примере каждый из девяти отсчетов входного сигнала добавляет в выходной сигнал смасштабированный и сдвинутый вариант импульсного отклика системы. Эти девять сигналов показаны на рис. 6.6. Сложение этих девяти сигналов дает выходной сигнал $y[n]$.

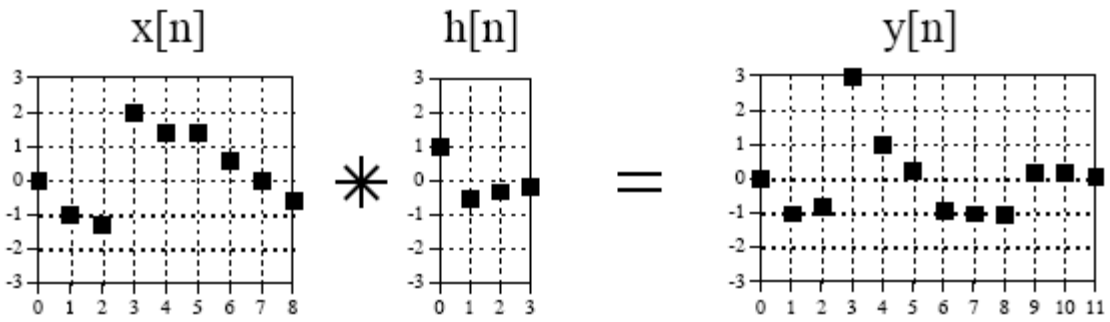


Рис. 6.5 Пример проблемы свертки

Взглянем на некоторые из этих девяти сигналов подробнее. Начнем с отсчета номер четыре в выходном сигнале, т.е. $x[4]$. Этот отсчет с индексным номером четыре имеет величину 1,4. Когда производится декомпозиция сигнала, этот отсчет представляется импульсом как: $1{,}4\delta[n-4]$. После прохождения через систему результирующим компонентом на выходе будет $1{,}4h[n-4]$. Этот сигнал показан в центральном прямоугольнике на рис. 6.6. Обратите внимание, что это импульсный отклик $h[n]$ умноженный на 1,4 и сдвинутый на четыре отсчета вправо. В отсчетах с нулевого по третий и с восьмого по одиннадцатый в пустые места были записаны нули. Для большей ясности, на рис.6.6 для обозначения точек данных, полученных из сдвинутого и смасштабированного импульсного отклика, используются *квадратики*, для обозначения точек с записанными нулями используются *ромбики*.

Теперь рассмотрим отсчет $x[8]$, последнюю точку во входном сигнале. Этот отсчет с индексным номером восемь имеет величину –0,5. Как показано в правом нижнем прямоугольнике на рис. 6.6, $x[8]$ в результате дает импульсный отклик, который сдвинут вправо на восемь позиций и умножен на –0,5. Нули записаны в позиции с нулевой по седьмую. Наконец рассмотрим вклад отсчетов $x[0]$ и $x[7]$. Величина обоих этих отсчетов равна нулю и, следовательно, получаемые выходные компоненты состоят из всех нулей.

В этом примере $x[n]$ - девяти точечный сигнал и $h[n]$ - четырех точечный сигнал. В следующем примере показанном на рис. 6.7 ситуация изменена на обратную, теперь $x[n]$ - четырех точечный сигнал, а $h[n]$ - девяти точечный сигнал. Используются те же самые две формы волны, их просто поменяли местами. Как видно из составляющих выходного

сигнала четыре отсчета в $x[n]$ преобразуются в четыре сдвинутых и смасштабированных варианта девяти точечных импульсных откликов. Так же, как и в предыдущем случае спереди и сзади добавлены нули.

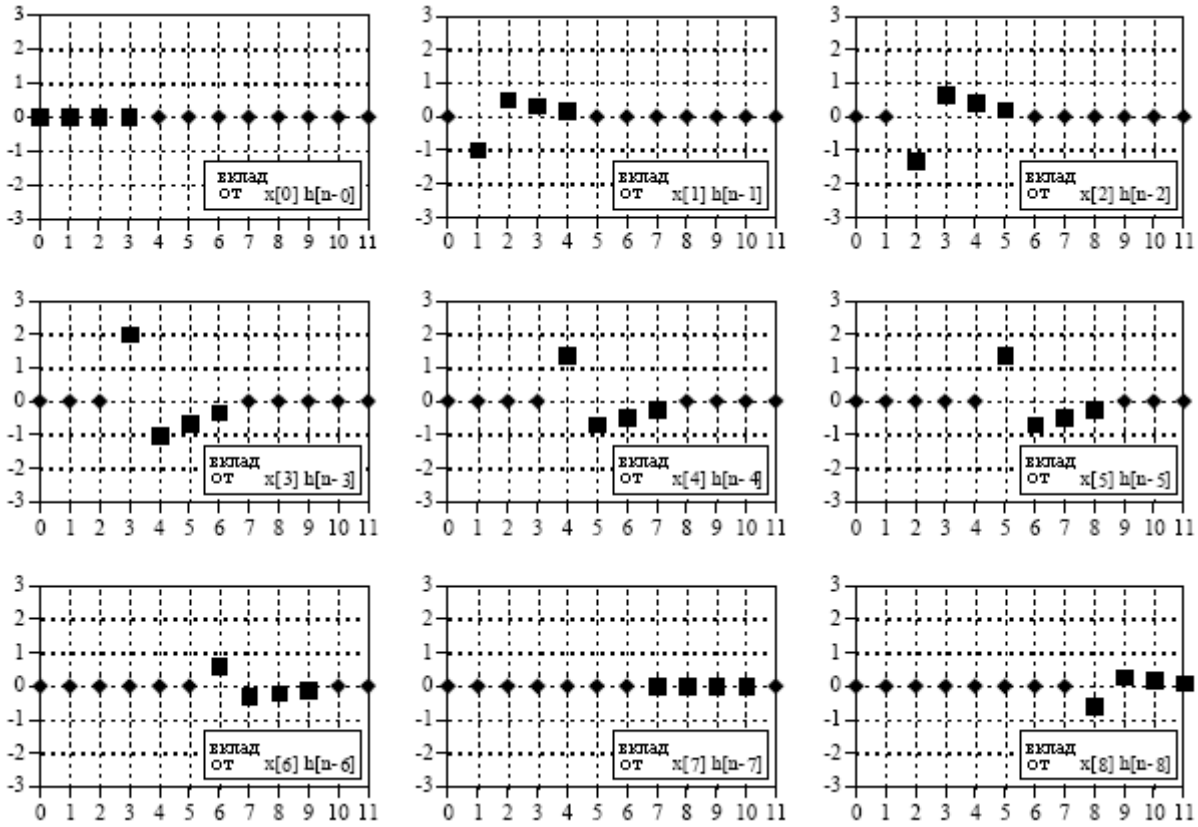


Рис.6.6 Составляющие выходного сигнала для свертки на рис. 6.5

Но подождите! Выходной сигнал на рис. 6.7 *идентичен* сигналу на рис. 6.5. Это подчеркивает важное свойство свертки. Свертка обладает свойством *коммутативности*: $a[n]*b[n]=b[n]*a[n]$. С точки зрения математики нет никакой разницы, какой из сигналов является входным, а какой импульсным откликом, имеет значение только то, что *два этих сигнала образуют свертку друг с другом*. Хотя математика и допускает взаимную замену двух сигналов, физического смысла в теории систем такая замена не имеет. Входной сигнал и импульсный отклик две совершенно разные вещи и взаимная замена их бессмысленна. Единственное, что дает свойство коммутативности так это *математический инструмент* для манипулирования уравнениями с целью достижения всевозможных результатов.

Программа для вычисления свертки, использующая алгоритм входной стороны (с точки зрения входа), приведена в таблице 6.1. Не забывайте, что программы в этой книге предполагают передачу *алгоритмов* в самой простой форме, даже ценой хорошего стиля программирования. Например, ввод и вывод данных осуществляется **мифическими** подпрограммами (строки 160 и 280) и каким образом должны выполняться эти операции не определяется. Вместе с тем ключевые моменты, необходимые для понимания, приведены в деталях.

Программа осуществляет свертку входного сигнала состоящего из 81 точки, помещенных в массив X[ ], с 31 точечным импульсным откликом, помещенным в массив H[ ], результатом работы программы является выходной сигнал из 111 точек, помещенных в массив Y[ ]. Все они имеют ту же длину, что и на рис.6.3 и рис. 6.4. Заметим, что для обозначения массивов используются заглавные буквы. Это отличие в обозначении от

первоначально принятого связано с тем, что прописные буквы зарезервированы для случая представления сигналов в частотной области. Кроме того, используемый здесь обычный Бейсик не допускает использования прописных букв для обозначения переменных. Заметим также, что в строке 240 звездочка используется как знак *умножения*. Напомним, что звездочка в программе означает умножение, в то время как звездочка в уравнении означает свертку. Звездочка в тексте (таком как документация или комментарий к программе) может означать и то, и другое.

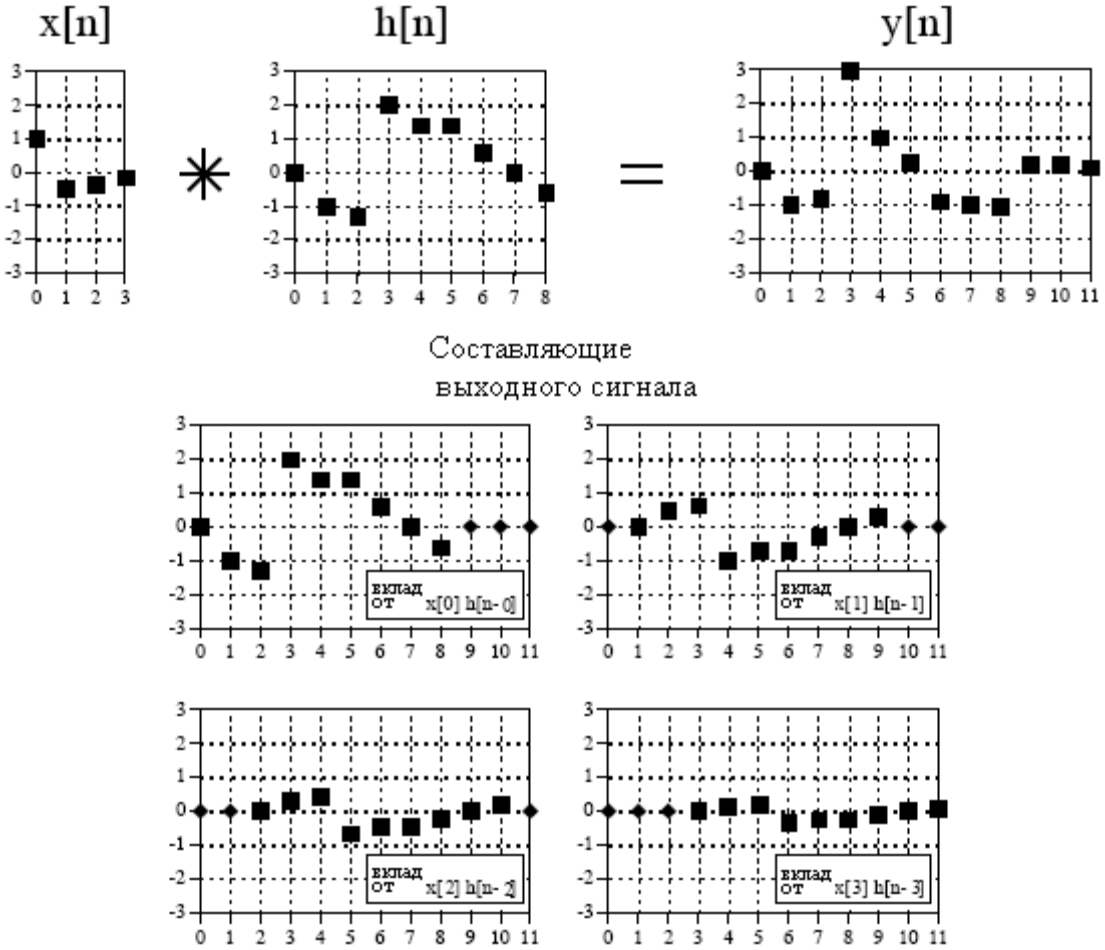


Рис. 6.7 Второй пример свертки

Мифическая подпрограмма в строке 160 помещает входной сигнал в X[ ] и импульсный отклик в H[ ]. Строки 180-200 обнуляют значения всех величин в Y[ ]. Это необходимо, поскольку Y[ ] используется в качестве аккумулятора для суммирования выходных составляющих по мере их вычисления. Строки с 220 по 260 - это сердце программы. Оператор FOR в строке 220 управляет циклом, который выполняется для каждой точки входного сигнала X[ ]. Для каждого отсчета входного сигнала выполняется внутренний цикл (строки 230-250), вычисляющий смасштабированный и сдвинутый вариант импульсного отклика, который суммируется с вектором Y[ ], аккумулирующим выходной сигнал. Такая структура с вложенными циклами (один цикл в пределах другого

Таблица 6.1

```
100 'CONVOLUTION USING THE INPUT SIDE ALGORITHM
110 '
120 DIM X[80]                    'The input signal, 81 points
```

```
130 DIM H[30]                  'The impulse response, 31 points
140 DIM Y[110]                 'The output signal, 111 points
150 '
160 GOSUB XXXX                 'Mythical subroutine to load X[ ] and H[ ]
170 '
180 FOR I% = 0 TO 110          'Zero the output array
190 Y(I%) = 0
200 NEXT I%
210 '
220 FOR I% = 0 TO 80           'Loop for each point in X[ ]
230    FOR J% = 0 TO 30        'Loop for each point in H[ ]
240      Y[I%+J%] = Y[I%+J%] + X[I%]*H[J%]
250    NEXT J%
260 NEXT I%                    '(remember, * is multiplication in programs!)
270 '
280 GOSUB XXXX                 'Mythical subroutine to store Y[ ]
290 '
300 END
```

цикла) является ключевой для программ осуществляющих свертку сигналов, познакомьтесь с нею поближе.

Вас может сбить с толку присутствие индексации непосредственно в строке 240! Предположим, что программа находится на половине пути своего прогона, что она только что приступила к обработке отсчета X[40], т.е. I%=40. Внутренний цикл проходит через каждую точку импульсного отклика, осуществляя три вещи. Первое, импульсный отклик *масштабируется* за счет умножения его на величину входного отсчета. Если бы это было единственным действием, производимым во внутреннем цикле, строка 240 была бы записана как Y[J%] = X[40]*H[J%]. Второе, смасштабированный импульс *сдвигается* на 40 отсчетов вправо, за счет добавления этого числа к индексу, используемому в выходном сигнале. Это второе действие меняет строку 240 до: Y[40+J%] = X[40]*H[J%]. Третье, Y[ ] должен аккумулировать (*синтезировать*) все результирующие сигналы от каждого отсчета во входном сигнале. Следовательно, новая информация должна быть прибавлена к информации, которая уже находилась в массиве. Это приводит к конечному виду команды: Y[40+J%] = Y[40+J%] + X[40]*H[J%]. Тщательно изучите это, это очень *запутанно*, но очень *важно*.

Алгоритм выходной стороны

Если первая точка зрения позволяет проанализировать, какой вклад вносит каждый отдельный отсчет *входного сигнала* во множество точек в выходном сигнале, то во второй точке зрения все меняется местами, и каждый отдельный отсчет в *выходном сигнале* рассматривается как результат, сформированный из вкладов множества точек во входном сигнале. Такой подход очень важен как с математической, так и с практической точек зрения. Предположим, что нам известен импульсный отклик и входной сигнал, и требуется найти их свертку. Наиболее прогрессивным методом решения этой задачи будет написание программы вычисляющей *выходной сигнал* в цикле, где за один цикл вычисляется один отсчет. Что-то вроде уравнения в виде: $y[n]$ = *некоторая комбинация других переменных*. Это означает, что отсчет n в выходном сигнале образуется из некоторой комбинации множества значений входного сигнала и импульсного отклика. Такой подход требует знания того, как каждый выходной отсчет может быть вычислен в

независимости от всех других отсчетов в выходном сигнале. Такую информацию дает алгоритм выходной стороны (с точки зрения выхода).

Рассмотрим пример того, как одна точка в выходном сигнале связана с несколькими точками во входном сигнале. В качестве примера будем использовать точку $y[6]$ на рис. 6.5. Эта точка равна сумме всех шестых точек в девяти выходных составляющих показанных на рис. 6.6. Остановимся детальнее на этих девяти составляющих и посмотрим, как они связаны с $y[6]$. Другими словами, найдем, какие из этих девяти сигналов содержат ненулевые отсчеты в шестой позиции. Пять из выходных составляющих имеют в шестой позиции *добавленные* нули (маркеры в виде ромбиков) и, следовательно, могут быть игнорированы. Только четыре из выходных составляющих имеют ненулевые значения в шестой позиции. Ими являются выходные составляющие, генерируемые из входных отсчетов: $x[3]$, $x[4]$, $x[5]$ и $x[6]$. Суммируя шестые отсчеты всех этих составляющих, $y[6]$ определится как: $y[6] = x[3]h[3] + x[4]h[2] + x[5]h[1] + x[6]h[0]$. Таким образом, четыре отсчета входного сигнала умножаются на четыре отсчета импульсного отклика и произведения суммируются.

Рис. 6.8 иллюстрирует алгоритм выходной стороны как **машину свертки**, а также структурную схему выполнения свертки. Представьте, что входной сигнал $x[n]$ и выходной сигнал $y[n]$ на странице зафиксированы, а машина свертки - все то, что находится внутри пунктирного прямоугольника, при необходимости может свободно перемещаться влево и вправо. Машина свертки располагается таким образом, чтобы ее выход был напротив позиции вычисляемого выходного отсчета. Четыре отсчета входного сигнала попадают на вход машины свертки. Эти величины умножаются на соответствующие отсчеты в импульсном отклике, после чего результаты произведений складываются. Полученное таким образом значение отсчета выходного сигнала помещается в соответствующее место. Например, как показано на рис. 6.8, $y[6]$ вычислено из четырех входных отсчетов: $x[3]$, $x[4]$, $x[5]$ и $x[6]$.

Чтобы вычислить $y[7]$, машина свертки смещается на один отсчет вправо. Теперь результат получается из других четырех отсчетов попадающих на вход машины свертки с $x[4]$ по $x[7]$, и значение $y[7]$ помещается в соответствующее место. Этот процесс повторяется для всех вычисляемых точек выходного сигнала.

Положение импульсного отклика *внутри* машины свертки очень важно. Импульсный отклик *перевернут слева направо*. Таким образом, нулевой отсчет располагается справа, а увеличивающиеся положительные номера отсчетов бегут влево. Чтобы понять геометрию этого переворота сравните этот импульсный отклик с нормальным импульсным откликом, показанным на рис. 6.5. Зачем нужен такой переворот? Он просто следует из математики (Этот переворот становится понятен, если обратить внимание на то, что изображения сигнала во времени и в пространстве выглядят перевернутыми слева направо друг относительно друга – прим. перев.). Импульсный отклик показывает, как каждая точка во входном сигнале воздействует на выходной сигнал. Результат в каждой точке в выходном сигнале получается за счет воздействия точек входного сигнала, взвешенных в соответствии с *перевернутым* импульсным откликом.

Рис.6.9 показывает, как машина свертки вычисляет некоторые отсчеты в выходном сигнале. Этот рисунок иллюстрирует также некоторые недоразумения свертки. На рис. 6.9 (а) машина свертки находится полностью слева, своим выходом нацеленным на $y[0]$. В этой позиции она пытается получить на вход отсчеты $x[-3]$, $x[-2]$, $x[-1]$ и $x[0]$. Проблема заключается в том, что трех из этих отсчетов: $x[-3]$, $x[-2]$ и $x[-1]$ не существует. Такая же дилемма возникает на рис. 6.9d, где машина свертки пытается получить отсчеты справа от заданных отсчетов входного сигнала, точки $x[9]$, $x[10]$ и $x[11]$.

Единственный способ решения этой проблемы – *выдумать* несуществующие отсчеты. Это осуществляется добавлением отсчетов в конец входного сигнала, причем каждый из добавленных отсчетов имеет значение *нуля*. Это называется **дополнить** сигнал

нулями. Вместо того чтобы пытаться получить доступ к несуществующим значениям, машина свертки получает отсчеты с нулевыми значениями. Поскольку эти нули сводят на нет результат умножения, конечный результат математически точно такой же, как если бы несуществующие входы *игнорировались*.

Важным является то, что находящиеся далеко слева и далеко справа отсчеты в выходном сигнале основаны на *неполной* информации. На жаргоне ЦОС **импульсный отклик не полностью растворяется во входном сигнале.** Если импульсный отклик имеет длину *M* точек, то первый *и* последний *M-1* отсчеты в выходном сигнале основываются на меньшей информации, чем отсчеты между ними. Это аналогично переходным процессам в электронной схеме, которой требуется некоторое время для стабилизации после включения питания. Разница заключается только в том, что этот переходный процесс легко игнорируется в электронике, но очень заметен в ЦОС.

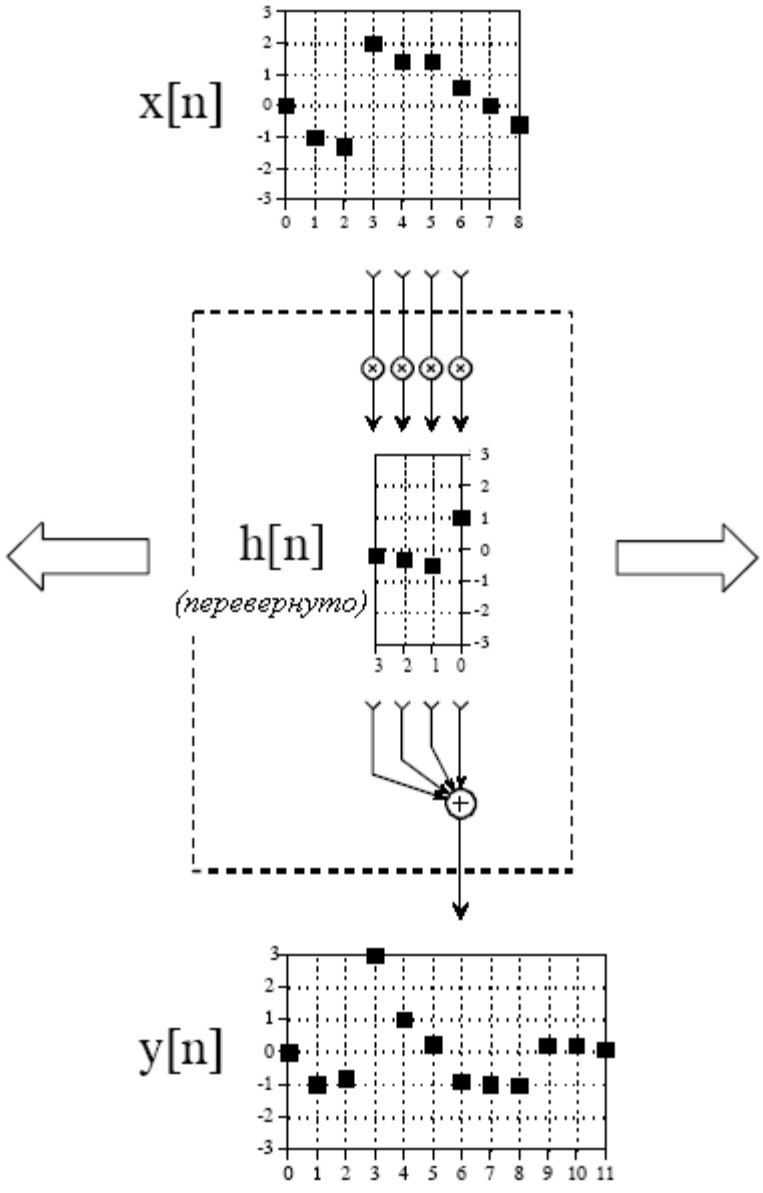


Рис. 6.8 Машина свертки

На рис. 6.10 приведен пример, показывающий, какие проблемы могут вызывать краевые эффекты. Входной сигнал является суммой постоянной составляющей и синусоидальной волны. Требуется удалить постоянную составляющую сигнала, оставив синусоидальную волну нетронутой. Это может быть сделано при помощи

высокочастотного фильтра с импульсным откликом, таким, как показано на рисунке. Проблема заключается в том, что первые и последние 30 точек являются сумбурными. Форма краевых областей может быть уяснена, если представить что входной сигнал дополнен 30 нулями слева, отсчеты с *x*[*-1*] по *x*[*-30*], и 30 нулями справа, отсчеты с *x*[*81*] по *x*[*110*]. Тогда выходной сигнал может быть рассмотрен, как отфильтрованная версия этой более длинной формы волны. Такие проблемы “краевых эффектов” часто встречаются в ЦОС. Как общее правило ожидайте, что начальные и конечные отсчеты в обработанном сигнале будут полностью бесполезными.

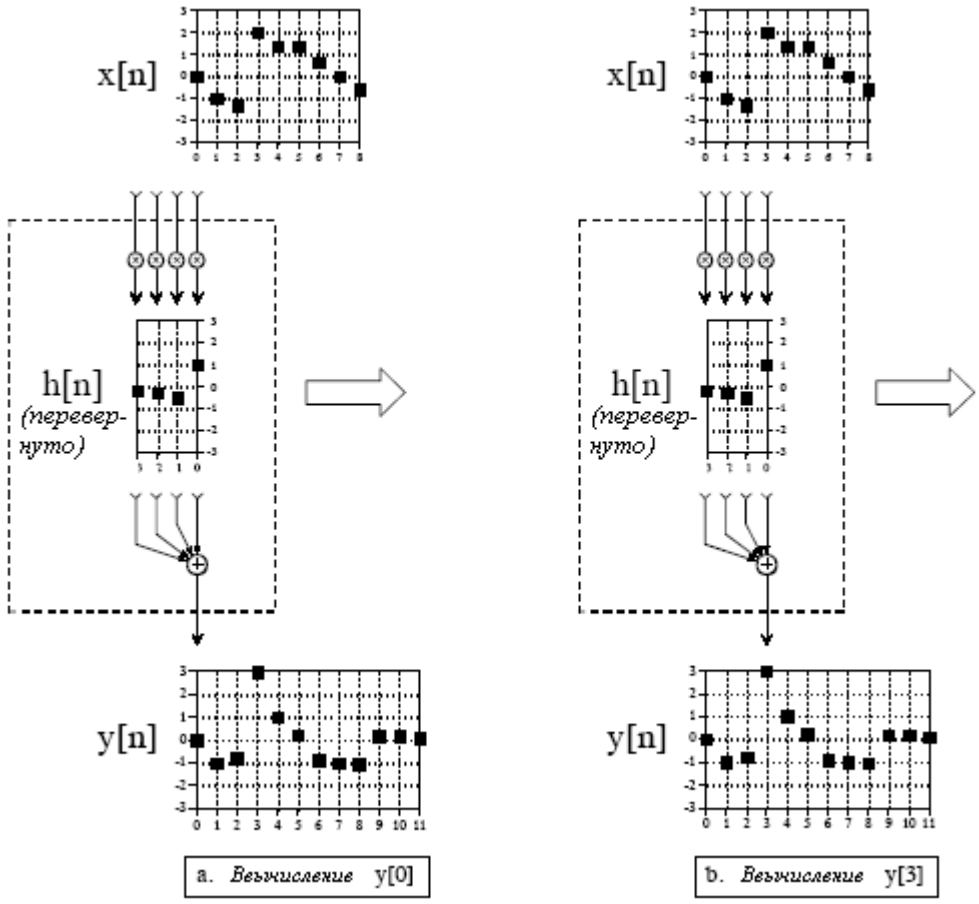


Рис. 6.9 Машина свертки в действии

Ну а сейчас математика. Используя машину свертки как директиву, мы можем записать стандартное уравнение для свертки. Если *x*[*n*] является *N* точечным сигналом, содержащим точки с *0* по *N-1*, а *h*[*n*] является *M* точечным сигналом, содержащим точки с *0* по *M-1*, свертка обоих: $y[n] = x[n] * h[n]$, это *N+M-1* точечный сигнал, содержащий точки с *0* по *N+M-1*, задаваемый как:

$$y[i] = \sum_{j=0}^{M-1} h[j]x[i-j]. \qquad (6.1)$$

Это уравнение называется **сверточной суммой**. Оно позволяет вычислить каждую точку в выходном сигнале независимо от всех других точек в выходном сигнале. Индекс *i* определяет какой отсчет в выходном сигнале будет вычисляться, и следовательно, соответствует позиционированию машины свертки слева – направо. В компьютерных

программах, выполняющих свертку, цикл заставляет пробежать этот индекс через каждый отсчет в выходном сигнале. Для вычисления одного из выходных отсчетов *внутри* машины свертки используется индекс *j*. По мере того как *j* пробегает значения с *0* по *M-1*, каждый отсчет в импульсном отклике *h*[*j*] умножается на соответствующий отсчет входного сигнала *x*[*i-j*]. Все эти произведения складываются, чтобы получить вычисляемый выходной отсчет. Изучение уравнения (6.1) позволяет полностью понять работу машины свертки. Многое в ЦОС базируется на этом уравнении. (Пусть у Вас не вызывает конфуз *n* в $y[n] = x[n] * h[n]$. Это просто держатель места необходимый для индикации того, что *некоторая* переменная является индексом внутри массива. Иногда уравнение записывается как: $y[\ ] = x[\ ] * h[\ ]$, просто для того, чтобы избежать ввода бессмысленного символа.)

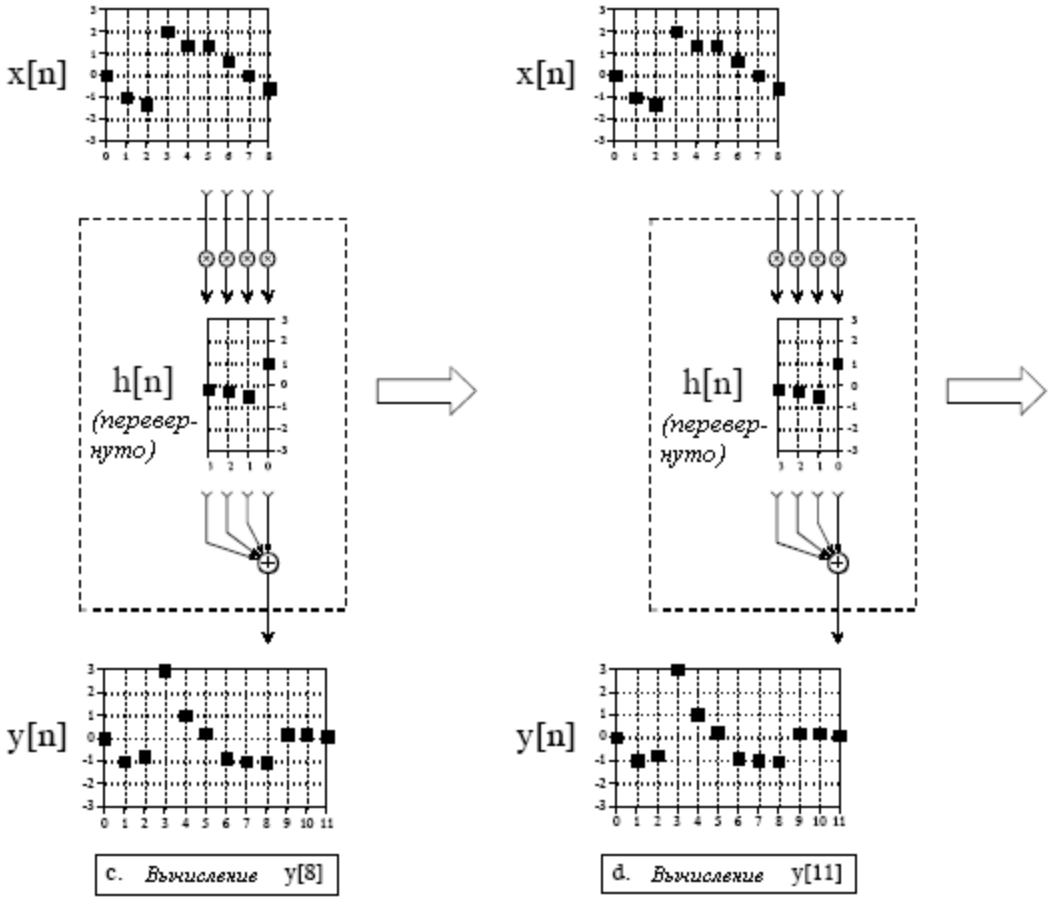


Рис. 6.9 (продолжение)

В таблице 6.2 показана программа для выполнения операции свертки, использующая *алгоритм выходной стороны*, т.е. непосредственно использующая уравнение (6.1). Эта программа вычисляет такой же выходной сигнал, что и приведенная ранее в таблице 6.1 программа, использующая *алгоритм входной стороны*. Обратите внимание, что главное отличие между этими двумя программами в следующем: в программе, использующей алгоритм входной стороны повторение цикла происходит после обработки каждого отсчета во *входном сигнале* (строка 220 в таблице 6.1), в то время как в программе использующей алгоритм выходной стороны повторение цикла происходит после обработки каждого отсчета в *выходном сигнале* (строка 180 в таблице 6.2).

Приведем здесь детальное описание работы программы. Цикл FOR-NEXT в строках со 180 по 250 проходит через каждый отсчет в выходном сигнале, используя I%

как индекс. Для каждого из этих значений внутренний цикл, составленный из строк с 200 по 230, вычисляет значения выходных отсчетов Y[I%]. Значение Y[I%] в строке 190 устанавливается в ноль, позволяя накапливать произведения внутри машины свертки. Цикл FOR-NEXT в строках с 200 по 240 обеспечивает непосредственное применение уравнения 6.1. Индекс J% проходит через каждый отсчет в импульсном отклике. Строка 230 обеспечивает перемножение каждого отсчета в импульсном отклике H[J%] с соответствующим отсчетом во входном сигнале X[I%-J%] и добавляет результат в аккумулятор.

Рис. 6.10 Краевые эффекты в свертке

В строке 230 отсчетом, выбираемым из входного сигнала, является X[I%-J%]. Строки 210и 220 предотвращают выход за пределы заданного массива с X[0] по X[80]. Другими словами, эта программа обращается с неопределенными отсчетами во входном сигнале, *игнорируя* их. Другой альтернативой может быть задание массива входного сигнала от X[-30] до X[110], позволяя дополнить истинные данные тридцатью нулями с каждой стороны. И как третья альтернатива, цикл FOR-NEXT в 180 строке может быть изменен так, чтобы осуществлять проход от 30 до 80 вместо прохода от 0 до 110. В этом случае программа будет вычислять только те отсчеты выходного сигнала, для которых импульсный отклик *полностью растворен* во входном сигнале. Важно чтобы Вы использовали одну из этих трех техник. Если Вы этого не сделаете программа потерпит крах, когда попытается прочесть данные за пределами границ.

Таблица 6.2

```
100 'CONVOLUTION USING THE OUTPUT SIDE ALGORITHM
110 '
120 DIM X[80]                   'The input signal, 81 points
130 DIM H[30]                   'The impulse response, 31 points
140 DIM Y[110]                  'The output signal, 111 points
150 '
160 GOSUB XXXX                  'Mythical subroutine to load X[ ] and H[ ]
170 '
180 FOR I% = 0 TO 110           'Loop for each point in Y[ ]
190   Y[I%] = 0                 'Zero the sample in the output array
200   FOR J% = 0 TO 30          'Loop for each point in H[ ]
210     IF (I%-J% < 0) THEN GOTO 240
220     IF (I%-J% > 80) THEN GOTO 240
230     Y(I%) = Y(I%) + H(J%) * X(I%-J%)
240   NEXT J%
250 NEXT I%
260 '
270 GOSUB XXXX                  'Mythical subroutine to store Y[ ]
```

```
280 '
290 END
```

Сумма взвешенных входов

Характеристика линейной системы полностью описывается ее импульсным откликом. Это является базисом алгоритма входной стороны: каждая точка входного сигнала добавляет смасштабированный и сдвинутый импульсный отклик в выходной сигнал. Математическое следование этому приводит к алгоритму выходной стороны: каждая точка в выходном сигнале получается как результат вклада множества точек входного сигнала, умноженных на *перевернутый* слева направо импульсный отклик. Несмотря на то, что все это является справедливым, это не исчерпывает полностью рассказ о том, почему свертка является такой важной в обработке сигналов.

Оглянемся назад и посмотрим на машину свертки на рис. 6.8, проигнорируем при этом то, что сигнал внутри пунктирного прямоугольника является *импульсным откликом*. Будем думать о нем как о наборе **весовых коэффициентов,** которые, так случилось, вложены в структурную схему. При таком взгляде каждый отсчет в выходном сигнале равен *сумме взвешенных входов*. Каждый отсчет на выходе определяется группой отсчетов входного сигнала в зависимости от того, какие весовые коэффициенты выбраны. Например, предположим, что есть десять весовых коэффициентов, значение каждого из которых одна десятая. Тогда каждый отсчет выходного сигнала будет *средним* десяти отсчетов входного сигнала.

Разовьем эту мысль дальше, нет необходимости ограничивать размещение весовых коэффициентов только *слева* от вычисляемого выходного отсчета. Например, рис. 6.8 показывает, что $y[6]$ вычисляется из: $x[3]$, $x[4]$, $x[5]$ и $x[6]$. Рассматривая машину свертки как сумматор со взвешенными входами, весовые коэффициенты могут быть выбраны *симметрично* вокруг выходного отсчета. Например, $y[6]$ может быть результатом вклада $x[4]$, $x[5]$, $x[6]$, $x[7]$ и $x[8]$. Используя те же обозначения индексов, что и на рис. 6.8 весовые коэффициенты для этих пяти входов будут находиться в: $h[2]$, $h[1]$, $h[0]$, $h[-1]$ и $h[-2]$. Другими словами, импульсный отклик, который соответствует нашему выбору, т.е. симметрии весовых коэффициентов, требует использования *отрицательных индексов*. Мы вернемся к этому в следующей главе.

С точки зрения математики, здесь есть только одна концепция: свертка определена уравнением (6.1). Однако научные и инженерные задачи подходят к этой единственной концепции с двух различных направлений. Иногда Вам может понадобиться рассматривать систему в терминах выглядящих в виде ее импульсного отклика. В другой раз вы будете понимать систему, как набор весовых коэффициентов. Поэтому Вам нужно быть знакомым с обоими взглядами и с методами перехода от одного к другому.

ГЛАВА
7

СВОЙСТВА СВЕРТКИ

Как следует из математики свертки, характеристики линейной системы полностью определены ее импульсным откликом. Этот факт положен в основу многочисленных методов обработки сигнала. Например, за счет *разработки* соответствующего импульсного отклика создаются цифровые фильтры. Вражеские самолеты обнаруживаются с помощью радара благодаря *анализу* и измерению импульсного отклика. Подавление эха при междугородних телефонных переговорах осуществляется с помощью создания импульсного отклика, который *противодействует* импульсному отклику отражений. Этот перечень можно еще продолжать и продолжать. Эта глава подробно останавливается на свойствах свертки и использовании ее в некоторых областях. Во-первых, обсуждаются несколько типичных импульсных откликов. Во-вторых, представлены методы имеющие отношение к последовательно и параллельно соединенным комбинациям линейных систем. В-третьих, представлен метод *корреляции*. В-четвертых, исследуется неприятная, связанная со сверткой, проблема, а именно, при использовании обычных алгоритмов и обычных компьютеров время вычисления может быть неприемлемо большим.

Обычный импульсный отклик

Дельта функция

Простейший импульсный отклик - это ничто иное, как дельта функция, показанная на рис. 7.1a. То есть, импульс на входе, вырабатывающий идентичный импульс на выходе. Последнее означает, что *все* сигналы проходят через систему *без изменений*. Свертка любого сигнала с дельта функцией дает точно такой же сигнал. Математически это записывается как:

$$x[n] * \delta[n] = x[n]. \qquad (7.1)$$

Это свойство делает дельта функцию **тождественной** свертке. Это аналог *нуля*, являющегося тождественным сложению ($a + 0 = a$), и *единицы*, являющейся тождественной умножению ($a \times 1 = a$). На первый взгляд, этот тип системы может показаться тривиальным и неинтересным. Но это не так! Такие системы являются идеальными для хранения данных, осуществления связи и измерений. Многое в ЦОС связано с прохождением через систему информации без ее изменения или ухудшения.

Рисунок 7.1b показывает небольшую модификацию импульсного отклика дельта функции. Если дельта функция становится больше или меньше по амплитуде, получающаяся в результате система является соответственно **усилителем** или **аттенюатором**. В форме уравнения, если *k больше чем единица*, получается усиление, и если *k меньше чем единица* получается ослабление.

$$x[n] * k\delta[n] = kx[n] \qquad (7.2)$$

Импульсный отклик на рис. 7.1c является дельта функцией со **сдвигом**. В системе это приводит к тому, что между входным и выходным сигналом вводится идентичный сдвиг. В зависимости от направления сдвига, это может быть описано как **запаздывание** или **опережение** сигнала. Если сдвиг будет представлен с помощью параметра *s*, то в виде уравнения это может быть записано как:

$$x[n] * \delta[n+s] = x[n+s] \tag{7.3}$$

Наука и инженерная практика изобилуют случаями, когда один сигнал является сдвинутой версией другого сигнала. Например, рассмотрим радиосигнал, переданный удаленным космическим зондом, и соответствующий сигнал, полученный на Земле. Время, требуемое радиосигналу для преодоления расстояния, приводит к запаздыванию между переданным и полученным сигналами. В биологии, электрические сигналы в смежных нервных клетках являются сдвинутыми версиями друг друга, что определено временем, необходимым потенциалу действия, для того чтобы пересечь соединяющий их синоптический переход.

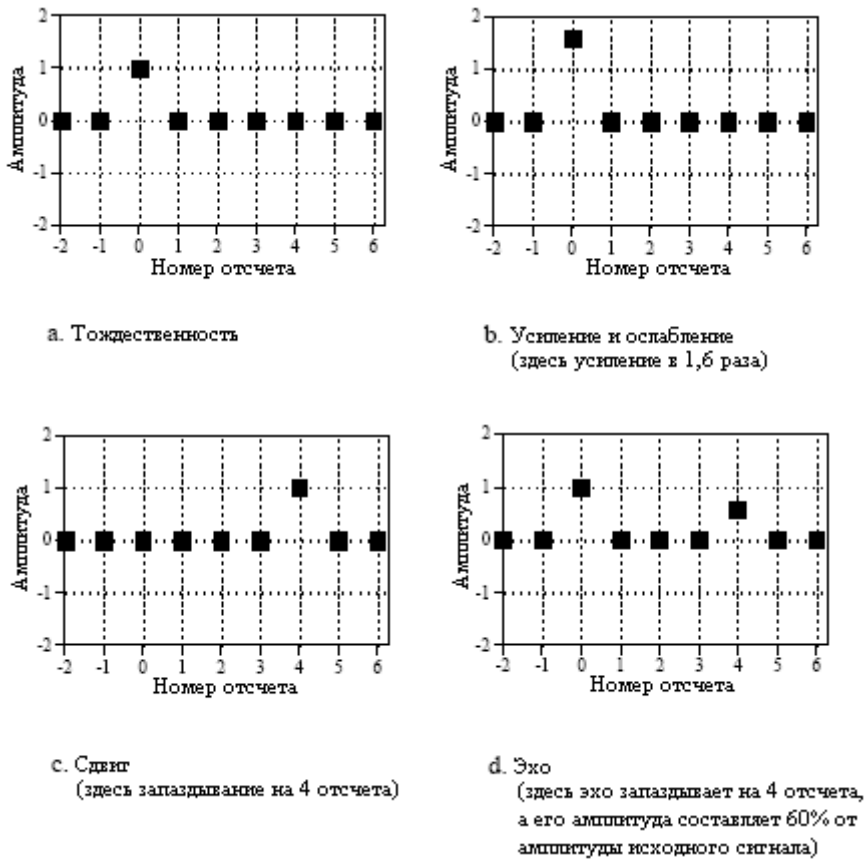


Рис. 7.1 Простейшие импульсные отклики использующие сдвинутую и смасштабированную дельта функцию

На рис. 7.1d показан импульсный отклик, состоящий из дельта функции, плюс сдвинутой и смасштабированной дельта функции. В соответствии с принципом суперпозиции выход этой системы есть входной сигнал плюс сдвинутый входной сигнал или *эхо*. Эхо очень важно для большого числа приложений ЦОС. Добавление эха - ключевая часть в создании звукозаписей с естественным и приятным звучанием. Радары и сонары анализируют эхо для обнаружения самолетов и подводных лодок. Геофизики

используют эхо для поиска нефти. Эхо также очень важно в телефонных сетях, поскольку Вы хотите его *устранить*.

Операции подобные исчислениям

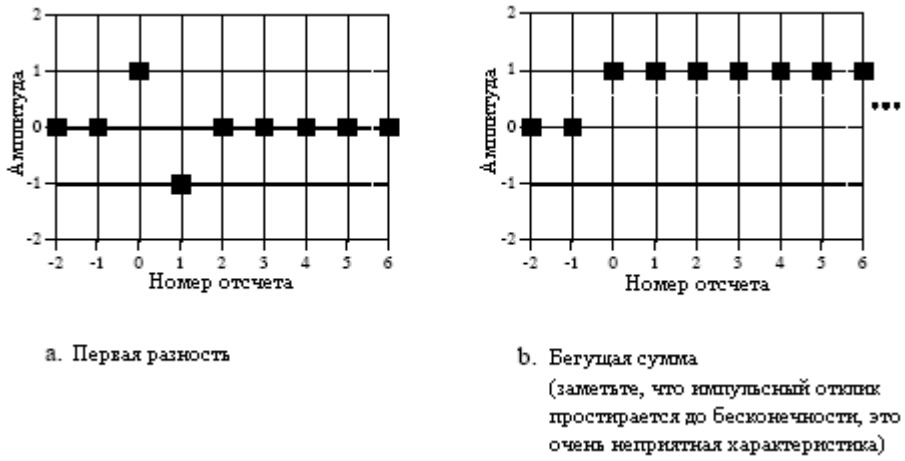
Свертка может изменять дискретные сигналы методами, которые походят на интегрирование и дифференцирование. Поскольку термины “производная” и “интеграл” специфически относятся к операциям над *непрерывными* сигналами, их дискретным дубликатам даются другие названия. Дискретная операция, которая имитирует *первую производную*, называется **первой разностью**. Аналогично, дискретная форма *интеграла* называется **бегущей суммой**. Обычно можно также услышать, что эти операции называют **дискретной производной** и **дискретным интегралом**, хотя математики хмурятся, когда слышат эти неофициально используемые термины.

На рис. 7.2 показаны импульсные отклики, которые реализуют первую разность и бегущую сумму. На рис. 7.3 показан пример использования этих операций. На рис. 7.3a исходный сигнал состоит из нескольких участков с изменяющимся наклоном. Свертка этого сигнала с импульсным откликом первой разности даст сигнал, приведенный на рис. 7.3b. Так же, как и в случае первой производной, амплитуда каждой точки в сигнале первой разности равна *наклону* в соответствующей точке исходного сигнала. Бегущая сумма является инверсной операцией по отношению к операции первой разности. То есть, свертка сигнала, показанного на рис. 7.3b, с импульсным откликом бегущей суммы даст сигнал, приведенный на рис. 7.3a.

Такие импульсные отклики достаточно просты, так что обычно нет необходимости целиком применять программу свертки. Достаточно подумать о них в альтернативном ключе: каждый отсчет в выходном сигнале является *суммой взвешенных отсчетов входного сигнала*. Например, первая разность может быть вычислена как:

$$y[n] = x[n] - x[n-1]. \qquad (7.4)$$

То есть, каждый отсчет в выходном сигнале равен разнице между двумя смежными отсчетами во входном сигнале. Например, $y[40] = x[40] - x[39]$. Следует заметить, что это не единственный путь нахождения *дискретной производной*. Другим обычно используемым подходом является определение наклона точек, симметрично расположенных вокруг исследуемой точки, подобно: $y[n] = (x[n+1] - x[n-1])/2$.



a. Первая разность

b. Бегущая сумма
(заметьте, что импульсный отклик простирается до бесконечности, это очень неприятная характеристика)

Рис. 7.2 Импульсные отклики имитирующие операции исчисления

Используя точно такой же подход, каждый отсчет в бегущей сумме может быть вычислен с помощью суммирования всех точек исходного сигнала *слева* от исследуемого

отсчета. Например, если $y[n]$ является бегущей суммой $x[n]$, тогда отсчет $y[40]$ находится суммированием отсчетов от $x[0]$ до $x[40]$. Аналогично отсчет $y[41]$ находится суммированием отсчетов от $x[0]$ до $x[41]$. Конечно же, вычисление бегущей суммы подобным образом будет очень неэффективно. Например, если $y[40]$ уже вычислено, $y[41]$ может быть вычислено с помощью только одного единственного суммирования: $y[41] = x[41] + y[40]$. В форме уравнения это будет записано как:

$$y[n] = x[n] + y[n-1]. \qquad (7.5)$$

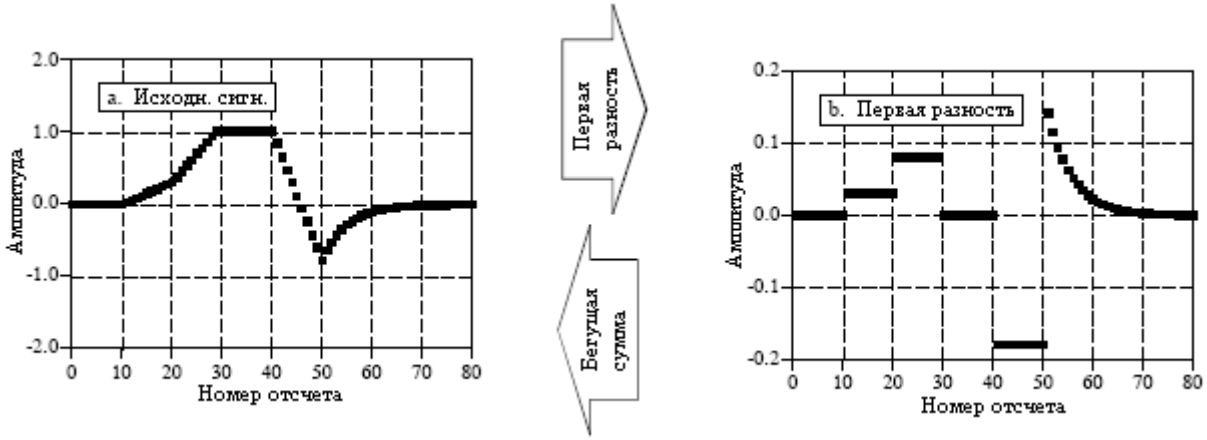


Рис. 7.3 Пример исчислений подобных операциям

Соотношения такого типа называются **уравнениями рекурсии** или **уравнениями в конечных разностях**. Мы еще раз вернемся к ним в Главе 19. А сейчас, наиболее важной идеей, которую следует понять, является то, что эти соотношения *идентичны* свертке, использующей импульсный отклик, представленный на рис. 7.2. Таблица 7.1 содержит компьютерные программы, которые реализуют эти операции подобные исчислениям.

Таблица 7.1.

```
100 'Calculation of the First Difference
110 Y[0] = 0
120 FOR I% = 1 TO N%-1
130 Y[I%] = X[I%] - X[I%-1]
140 NEXT I%
```

```
100 'Calculation of the running sum
110 Y[0] = X[0]
120 FOR I% = 1 TO N%-1
130 Y[I%] = Y[I%-1] + X[I%]
140 NEXT I%
```

Примечание: Исходный сигнал содержится в X[ ], а обработанный сигнал в Y[ ].

Низкочастотный и высокочастотный фильтры

Проектирование цифровых фильтров подробно рассматривается в более поздних главах. Пока же удовлетворимся тем, что попытаемся понять общую форму *ядра* (другое название импульсного отклика фильтра) низкочастотного и высокочастотного *фильтров*. На рис. 7.4 показано несколько типичных ядер низкочастотного фильтра. В общем случае, ядра низкочастотного фильтра состоят из группы *смежных положительных точек*. Это приводит к тому, что каждый отсчет в выходном сигнале является взвешенным средним значением большого числа смежных точек входного сигнала. Такое усреднение *сглаживает* сигнал, удаляя таким образом его высокочастотные составляющие. Как показано на примере синк функции на рис. 7.4, некоторые ядра низкочастотных фильтров включают в себя несколько находящихся в хвостах отсчетов с отрицательными

значениями. Точно так же, как и в аналоговой электронике, цифровые фильтры нижних частот используются для снижения шума, разделения сигналов, формирования формы волны, и т.п.

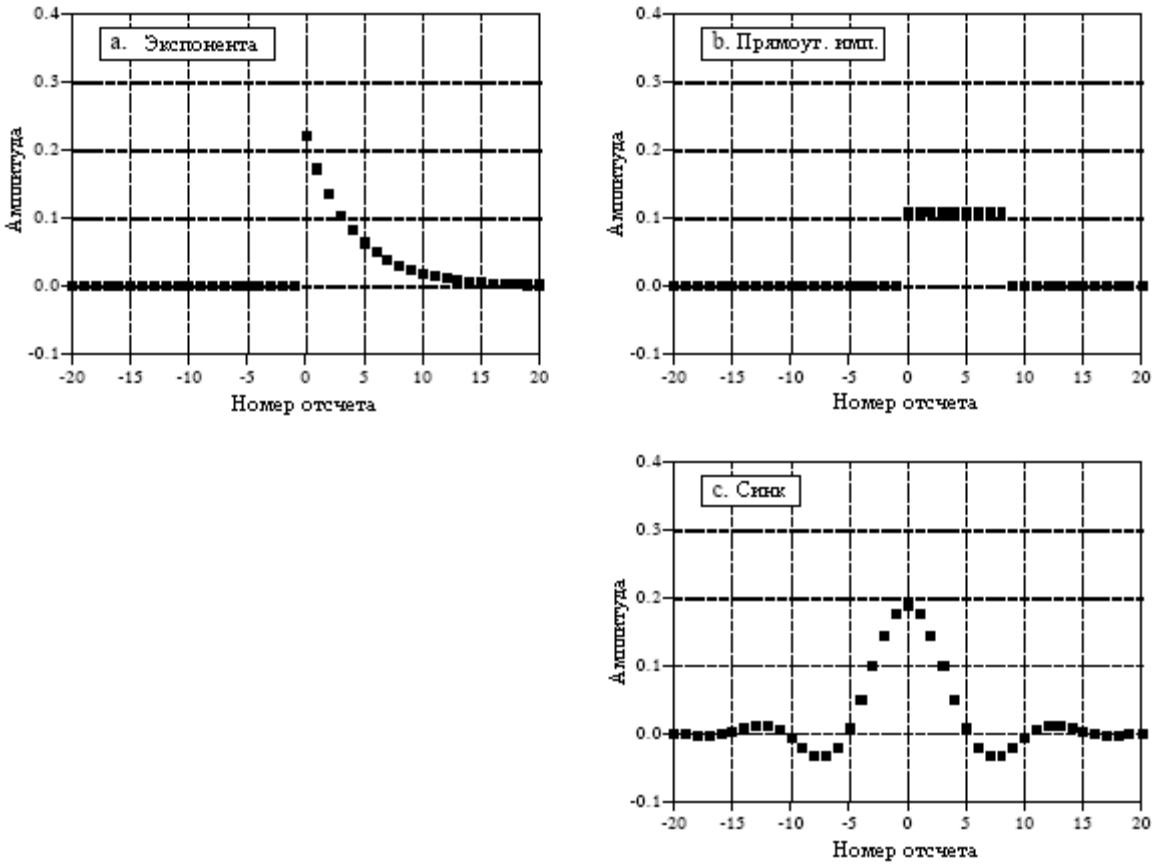


Рис. 7.4 Типичные ядра низкочастотных фильтров

Делая ядро фильтра более широким или более узким, изменяется частота среза фильтра. Если усиление фильтра нижних частот по постоянной составляющей (нулевая частота) равно *единице*, то сумма всех точек в импульсном отклике так же должна быть равна *единице*. Как иллюстрируется на рис. 7.4a и рис. 7.4c, некоторые ядра фильтра *теоретически* простираются до бесконечности, не достигая значения нуля. В реальной практике после некоторого количества отсчетов хвосты отсекаются, давая возможность представить ядра конечным числом точек. А как иначе можно было бы хранить их в компьютере?

Рис. 7.5 показывает три типичных ядра высокочастотного фильтра, полученных из соответствующих показанных на рис. 7.4 ядер фильтра нижних частот. В проектировании фильтров это обычная стратегия, сначала разрабатывают фильтр нижних частот, а затем преобразуют его к тому, что Вам нужно: фильтру верхних частот, полосно-пропускающему фильтру, полосно-подавляющему фильтру и т.д. Чтобы понять преобразование низкочастотного фильтра в высокочастотный, вспомните, что импульсный отклик в виде дельта функции пропускает весь сигнал, в то время как импульсный отклик низкочастотного фильтра пропускает только низкочастотные составляющие. Благодаря суперпозиции ядро фильтра, состоящего из дельта функции за минусом ядра фильтра низкой частоты, будет пропускать весь сигнал за минусом низкочастотных составляющих. Вот так рождается высокочастотный фильтр! Как показано на рис. 7.5, дельта функция обычно добавляется в центр симметрии ядра или, если ядро фильтра не симметрично, в отсчет с номером ноль. По постоянной

составляющей (нулевая частота), фильтры высоких частот имеют нулевое усиление, достигаемое равенством *нулю* суммы всех точек в импульсном отклике.

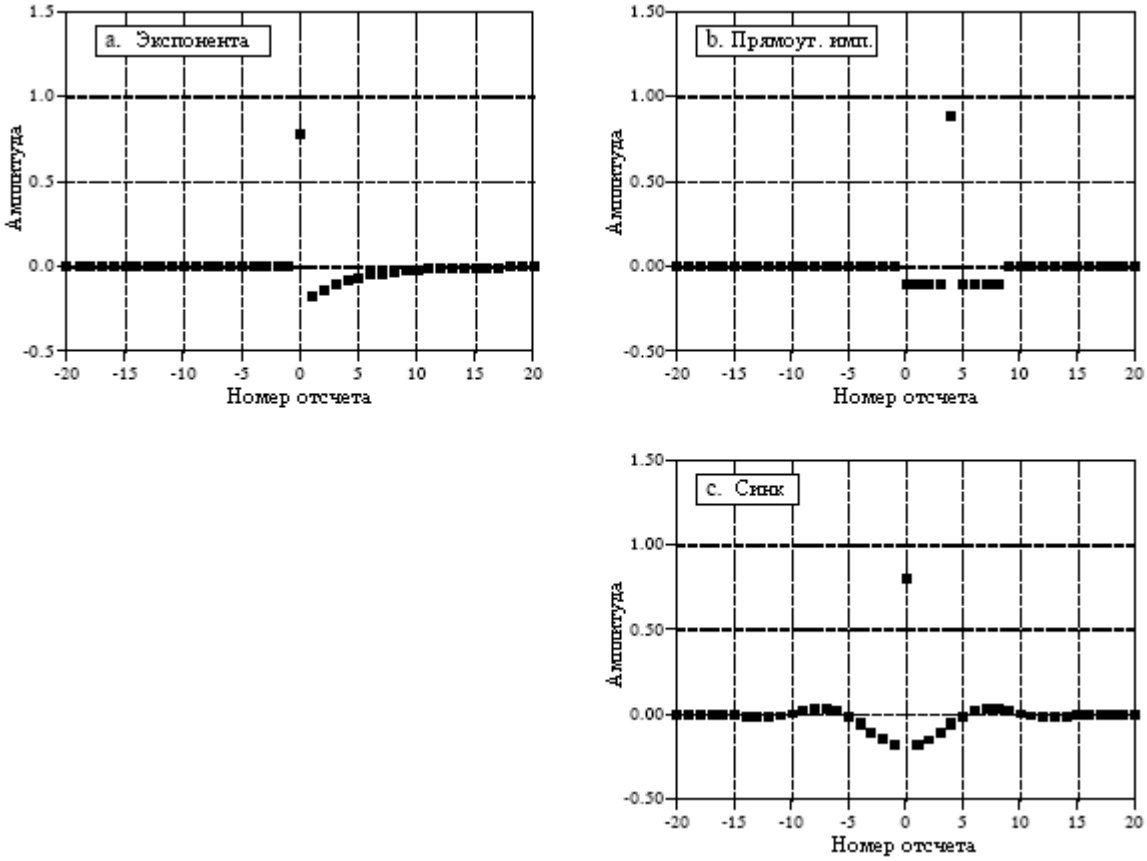


Рис. 7.5 Типичные ядра высокочастотных фильтров

Причинные и беспричинные сигналы

Представьте простую аналоговую электронную схему. Если вы подадите на вход короткий импульс, Вы увидите отклик на выходе. Это природа причины и следствия, на которых основана наша вселенная. Единственная вещь, которая нам определенно известна: *любое событие происходит по какой-нибудь причине*. Это основная характеристика того, что мы называем *временем*. Теперь сравните сказанное с системой ЦОС преобразующей входной сигнал в выходной, каждый из которых хранится в компьютере в виде массива. Если она имитирует систему реального мира, она должна следовать тем же принципам причинности, которым следует реальный мир. Например, значение отсчета с номером восемь во входном сигнале, в выходном сигнале может оказать воздействие только на отсчет с номером восемь или больше. Говорят, что системы, функционирующие в такой манере, являются **причинными**. Конечно же, цифровая обработка не обязательно должна работать таким образом. Поскольку и входные и выходные сигналы представляют собой массивы чисел, хранящихся в компьютере, любое значение входного сигнала может воздействовать на любое значение выходного сигнала.

Как показано в примере на рис. 7.6, импульсный отклик причинных систем должен иметь нулевое значение для всех отсчетов с *отрицательными номерами*. Поразмышляйте об этом, взглянув на свертку со стороны входа. Чтобы быть причинным, импульс в отсчете с номером n во входном сигнале должен воздействовать только на точки в выходном сигнале с номером отсчета n или больше. При обычном использовании термин причинный применяется к любому сигналу, в котором все отсчеты с отрицательными

номерами имеют значение нуля, в независимости от того является ли это импульсным откликом или нет.

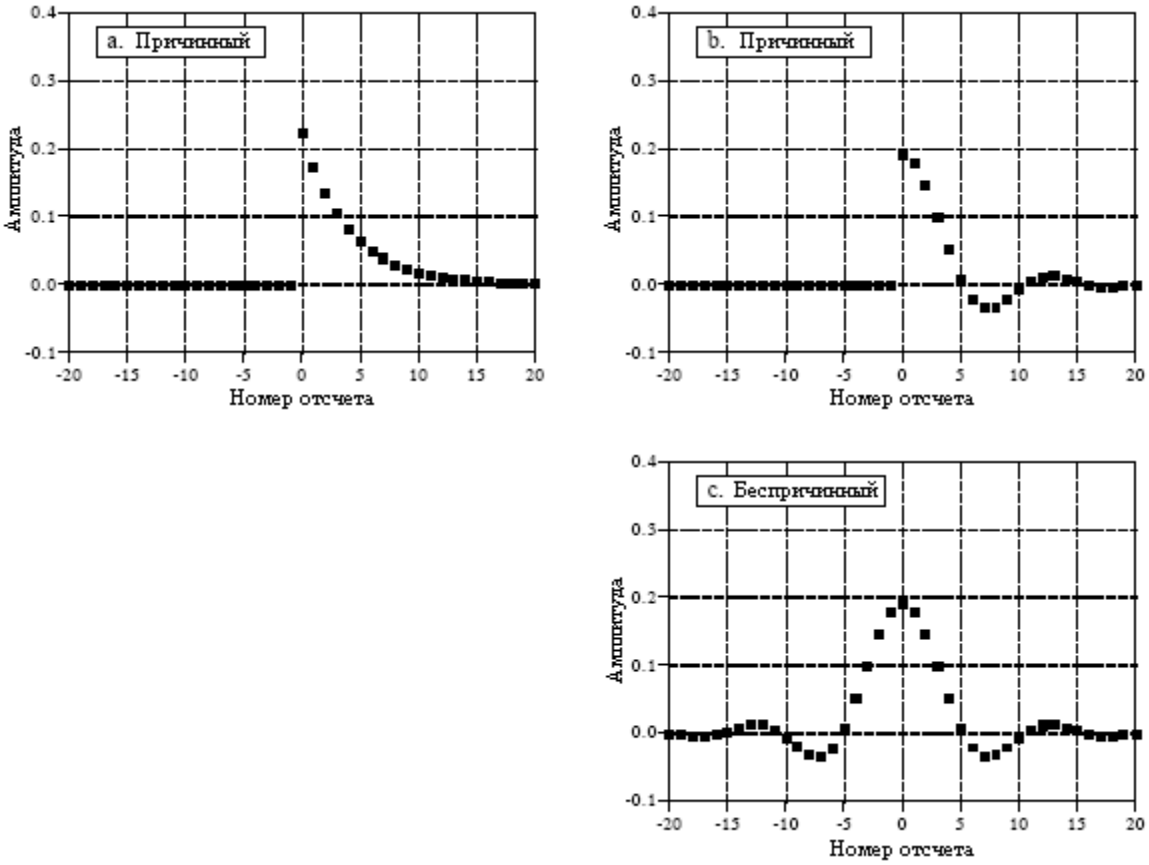


Рис. 7.6 Примеры причинных сигналов

Нулевая фаза, линейная фаза и нелинейная фаза

Как показано на рис. 7.7, говорят, что сигнал имеет **нулевую фазу**, если он обладает симметрией слева направо относительно отсчета с номером ноль. Говорят, что сигнал имеет **линейную фазу**, если он обладает симметрией слева направо, но относительно отсчета с номером отличным от нуля. Это означает, что любой сигнал с линейной фазой может быть приведен к сигналу с нулевой фазой простым сдвигом влево или вправо. Наконец, говорят, что сигнал имеет **нелинейную фазу**, если он не обладает симметрией слева направо.

Вы, вероятно, думаете, что эти названия, как кажется на первый взгляд, не следуют из их определений. Что может *фаза* сделать с *симметрией*? Ответ находится в частотном спектре и более детально будет обсужден в поздних главах. Вкратце, частотный спектр любого сигнала состоит из двух частей амплитуды и фазы. Частотный спектр сигнала симметричного вокруг нуля имеет нулевую фазу. Аналогично, частотный спектр сигнала, симметричного вокруг некоторой ненулевой точки, имеет фазу, представляющую собой прямую линию, т.е. линейную фазу. Наконец, частотный спектр несимметричного сигнала имеет фазу, не являющуюся прямой линией, т.е. он имеет нелинейную фазу.

Специальное замечание относительно потенциально запутывающих терминов: *линейная* и *нелинейная фаза*. Как они должны повлиять на концепцию линейности системы, обсужденной в предыдущих главах? Абсолютно никак! Линейность систем является широкой концепцией, на которой базируется почти вся ЦОС (суперпозиция, однородность, аддитивность, и т.д.). *Линейная* и *нелинейная фаза* означает только то, что фаза является или не является прямой линией. Фактически, система должна быть линейна даже тогда, когда фаза является нулевой, линейной или нелинейной.

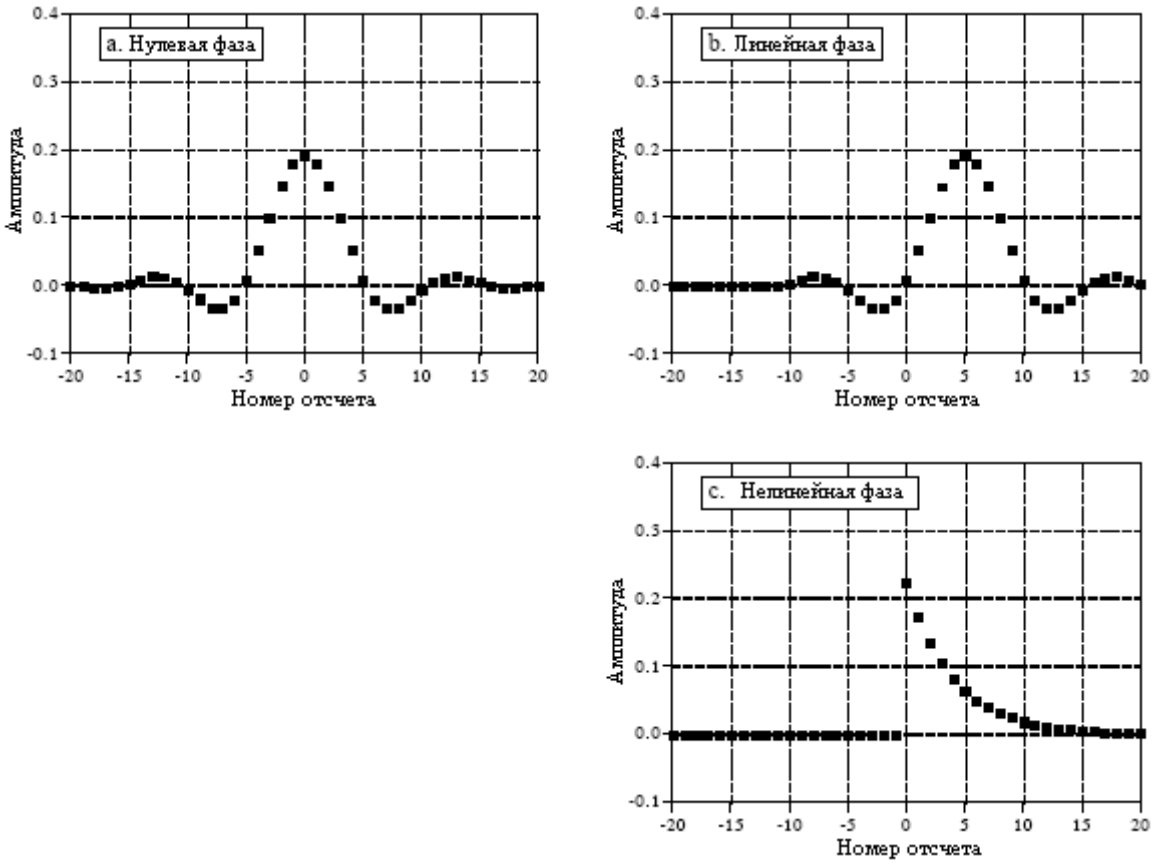


Рис. 7.7 Примеры линейности фазы

Математические свойства

Свойство коммутативности

В математической форме свойство коммутативности для свертки выражается как:

$$a[n] * b[n] = b[n] * a[n]. \qquad (7.6)$$

Выражаясь словами, порядок, в котором находятся сигналы, подвергаемые свертке, не имеет ни какой разницы, результат в любом случае идентичен. Как показано на рис. 7.8, этот факт имеет странное значение для теории систем. В любой линейной системе входной сигнал и импульсный отклик системы могут быть *поменяны* местами без изменения выходного сигнала. Это интересно, но обычно не имеет никакого физического смысла. Входной сигнал и импульсный отклик - очень разные вещи. И только то, что математика *позволяет* Вам проделывать что-либо, еще не означает, что есть смысл это делать. Например, предположим, что Вы зарабатываете: 10 долл./час × 2000 час./год = 20000 долл./год. Свойство коммутативности, которым обладает операция умножения, позволяет Вам заработать такую же годовую зарплату, работая всего 10 часов в году за 2000 долл./час. Давайте посмотрим, как Вы убедите своего босса, что это одно и тоже! Вопреки этому свойство коммутативности позволяет увидеть больше возможностей в ЦОС для манипулирования уравнениями точно так же, как и в обычной алгебре.

Свойство ассоциативности

Можно ли осуществить свертку трех и более сигналов? Ответом будет – да, и свойство ассоциативности описывает как: сделайте свертку двух сигналов, чтобы

получить промежуточный сигнал, а затем сделайте свертку промежуточного сигнала с третьим сигналом. Свойство ассоциативности гарантирует, что последовательность выполнения сверток не имеет значения. В виде уравнения это выглядит как:

$$(a[n] * b[n]) * c[n] = a[n] * (b[n] * c[n]). \qquad (7.7)$$

Свойство ассоциативности используется в теории систем для описания того, как ведут себя **системы при последовательном соединении**. Как показано на рис. 7.9, считается, что две или более системы соединены *последовательно*, если выход одной системы используется как вход для следующей системы. Из свойства ассоциативности вытекает, что последовательность расположения систем может быть перестроена без изменения общего отклика каскада. И далее, любое количество систем соединенных последовательно может быть заменено одной *единственной* системой. Импульсный отклик, заменяющей системы находится сверткой импульсных откликов всех исходных систем.

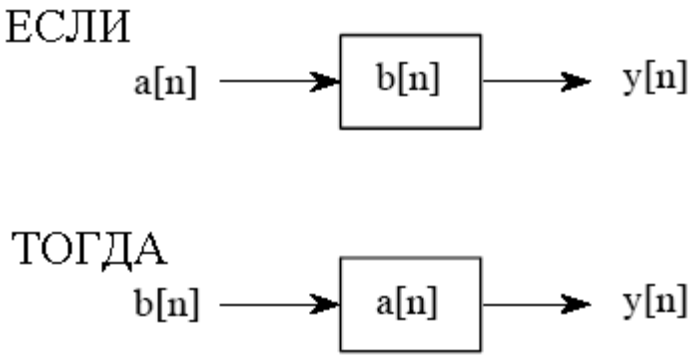


Рис. 7.8 Свойство коммутативности в теории систем

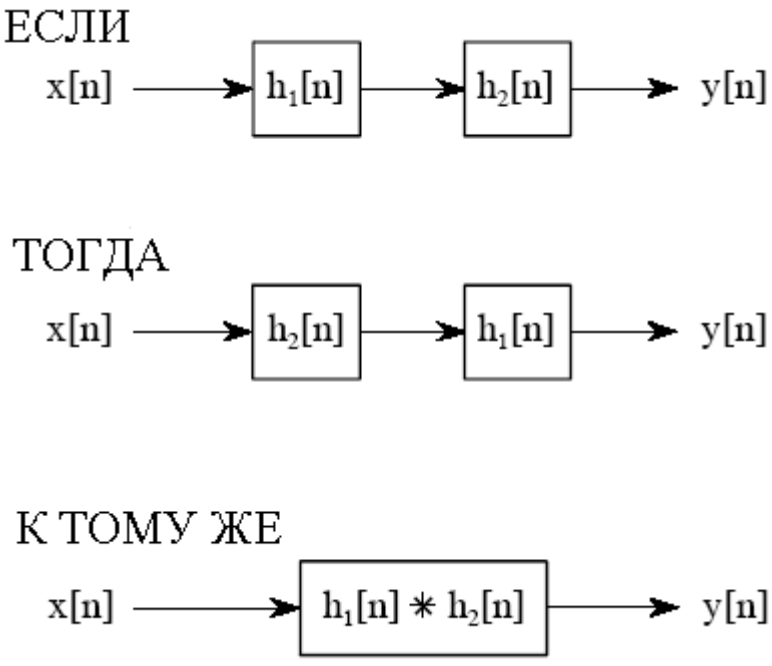


Рис. 7.9 Свойство ассоциативности в теории систем

Свойство дистрибутивности

Свойство дистрибутивности записывается в форме уравнения как:

$$a[n]*b[n]+a[n]*c[n]=a[n]*(b[n]+c[n]). \qquad (7.8)$$

Свойство дистрибутивности описывает работу **параллельно соединенных систем с суммированием выходов**. Как показано на рис. 7.10, две или более систем могут совместно иметь один и тот же вход $x[n]$, при этом их выходы суммируются для получения сигнала $y[n]$. Свойство дистрибутивности позволяет заменять такие комбинации систем одной единственной системой с импульсным откликом равным *сумме* импульсных откликов исходных систем.

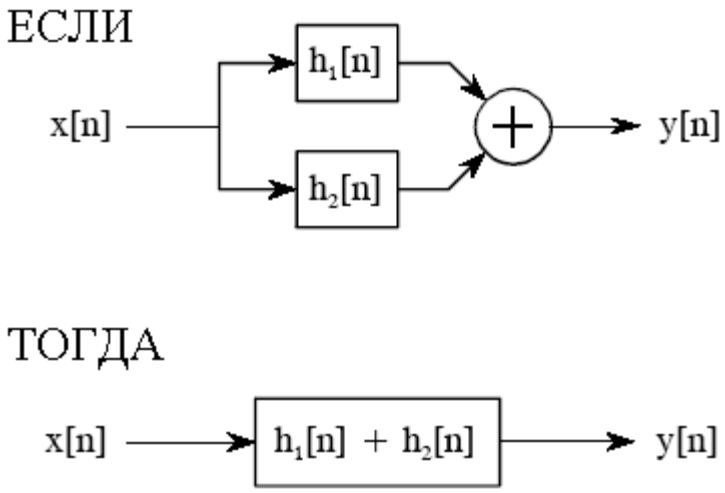


Рис. 7.10 Свойство дистрибутивности в теории систем

Перенесение с входа на выход

Это скорее способ мышления относительно общей ситуации в обработке сигналов, чем формальное математическое свойство. Как иллюстрируется на рис. 7.11, вообразите линейную систему, получающую входной сигнал $x[n]$ и генерирующую выходной сигнал $y[n]$. А сейчас предположим, что входной сигнал изменяется некоторым линейным способом, приводя к новому входному сигналу, который мы назовем $x'[n]$. В свою очередь это приведет к новому выходному сигналу $y'[n]$. Вопрос заключается в следующем, как изменения во входном сигнале соотносятся с изменениями в выходном сигнале? Ответом является следующее: *выходной сигнал изменяется точно таким же линейным способом, каким происходило изменение входного сигнала*. Например, если входной сигнал увеличился по амплитуде в два раза, выходной сигнал будет также увеличиваться в два раза. Если взята производная входного сигнала, будет также взята производная выходного сигнала. Если входной сигнал был некоторым способом подвергнут фильтрации, выходной сигнал будет подвергнут фильтрации в точно такой же манере. Это может быть легко доказано с помощью свойства ассоциативности.

Центральная предельная теорема

Центральная предельная теорема важный инструмент в теории вероятности, поскольку она математически объясняет, почему Гауссово распределение вероятности так часто наблюдается в природе. Например: амплитуда теплового шума в электронных схемах подчиняется Гауссову распределению; поперечная интенсивность лазерного луча является Гауссианом; даже дырки от попаданий вокруг глаза быка, нарисованного на доске для метания дротиков, подчиняются Гауссову закону. В своей наиболее простой форме центральная предельная теорема утверждает, что Гауссово распределение

наблюдается тогда, когда наблюдаемая переменная является суммой большого числа случайных процессов. Даже если составляющие процессы не имеют Гауссова распределения, их сумма будет его иметь.

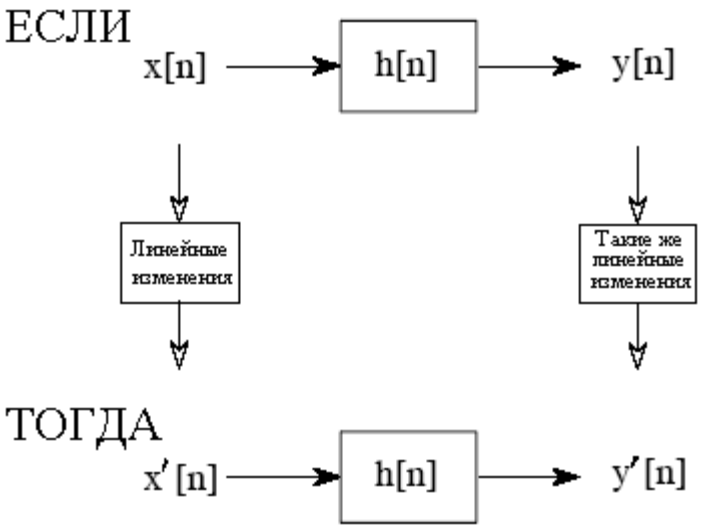


Рис. 7.11 Перенесение с входа на выход

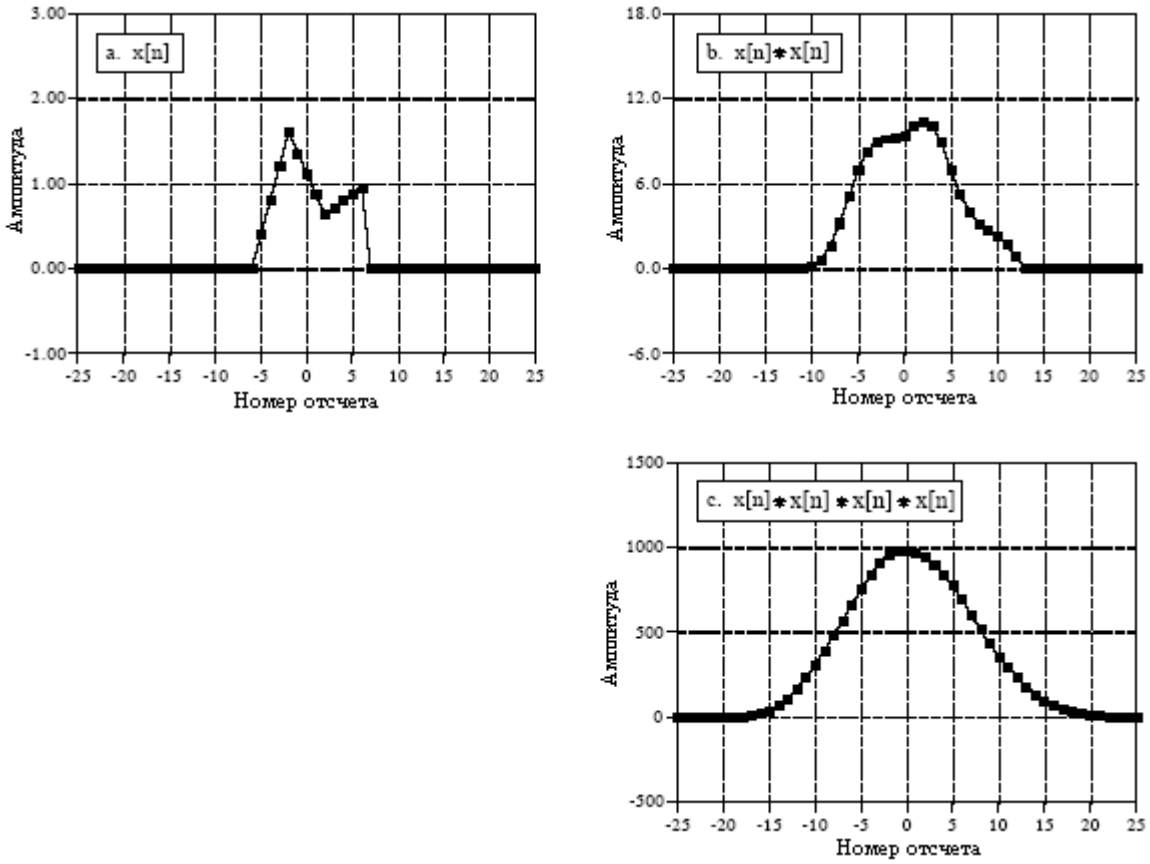


Рис. 7.12 Пример свертки импульса с самим собой

Центральная предельная теорема имеет интересную связь со сверткой. Если многократно производить свертку сигнала подобного импульсному *с самим собой*, то в результате получается Гауссиан. Рис. 7.12 показывает пример этого. Сигнал на рис. 7.12a

представляет собой импульс неправильной формы, преднамеренно выбранный так, чтобы очень сильно отличаться от Гауссиана. Рис. 7.12b показывает результат свертки этого сигнала с самим собой *один* раз. Рис. 7.12c показывает результат свертки этого сигнала с самим собой *три* раза. Даже всего только после трех операций свертки форма волны выглядит очень похожей на Гауссиан. На математическом жаргоне процедура *очень быстро сходится* к Гауссиану. Ширина результирующего Гауссиана (т.е. σ в уравнении (2.7) или (2.8)) равна ширине исходного импульса (выраженного через σ в уравнении (2.7)) умноженной на корень квадратный из количества операций свертки.

Корреляция

Концепция корреляции лучше всего может быть представлена на примере. Рис. 7.13 показывает ключевые элементы радиолокационной системы. Специально сконструированная антенна передает короткий всплеск энергии радиоволн в выбранном направлении. Если распространяющаяся волна ударяется об объект, такой как вертолет на этой иллюстрации, небольшая часть энергии отражается в обратном направлении к радиоприемнику, расположенному около передатчика. Переданный импульс имеет специфическую, выбранную нами форму, подобную показанной в этом примере форме треугольника. Принятый сигнал будет состоять из двух частей: (1) сдвинутой и смасштабированной версии переданного импульса и (2) случайного шума, полученного в результате интерференции радиоволн, теплового шума в электронике и т.д. Поскольку скорость, с которой путешествуют радиосигналы, известна, это - скорость света, сдвиг между переданным и полученным импульсами является непосредственным измерением расстояния до обнаруживаемого объекта. Ну а вот и проблема: имея сигнал некоторой известной формы, каким является наилучший способ определения, где находится (или есть ли) этот сигнал в *другом* сигнале. Ответ дает корреляция.

Корреляция это математическая операция очень похожая на свертку. Так же, как и в случае со сверткой, корреляция использует два сигнала для того, чтобы получить третий. Этот третий сигнал называется **взаимной корреляцией** двух входных сигналов. Если сигнал коррелирует *с самим собой*, то вместо этого результирующий сигнал называют **автокорреляцией**. Машина свертки, представленная в предыдущей главе, показывает, как выполняется свертка. Рис. 7.14 является подобной иллюстрацией **корреляционной машины**. Принятый сигнал $x[n]$ и сигнал взаимной корреляции $y[n]$ на странице зафиксированы. Форма волны, которую мы ищем $t[n]$, обычно называемая сигналом **цели**, находится *в пределах* корреляционной машины. Каждый отсчет в $y[n]$ вычисляется перемещением корреляционной машины влево или вправо до тех пор, пока она не укажет на отсчет, который будет обрабатываться. Затем, обозначенные отсчеты из принятого сигнала попадают в корреляционную машину и перемножаются с соответствующими точками в сигнале цели. Сумма этих произведений заносится затем в соответствующий отсчет сигнала взаимной корреляции.

Амплитуда каждого отсчета в сигнале взаимной корреляции является мерой того, насколько *в этом месте* принятый сигнал *походит* на сигнал цели. Это означает, что для каждого сигнала цели, присутствующего в принятом сигнале, появится максимум в сигнале взаимной корреляции. Другими словами, значение взаимной корреляции становится максимальным тогда, когда сигнал цели *сравняется* с теми же самыми особенностями в принятом сигнале.

А что если сигнал цели содержит отсчеты с отрицательными значениями? Ничего не меняется. Представьте, что корреляционная машина позиционируется таким образом, что сигнал цели совершенно сравнялся с соответствующей формой волны в принятом сигнале. Как только отсчеты из принятого сигнала попадут в корреляционную машину, они будут перемножены с их соответствующими отсчетами в сигнале цели. Пренебрегая шумом, положительный отсчет будет умножен сам на себя, давая положительное число.

Аналогично, отрицательный отсчет будет умножен сам на себя, также давая положительное число. Даже если сигнал цели является полностью отрицательным, пик во взаимной корреляции будет оставаться положительным.

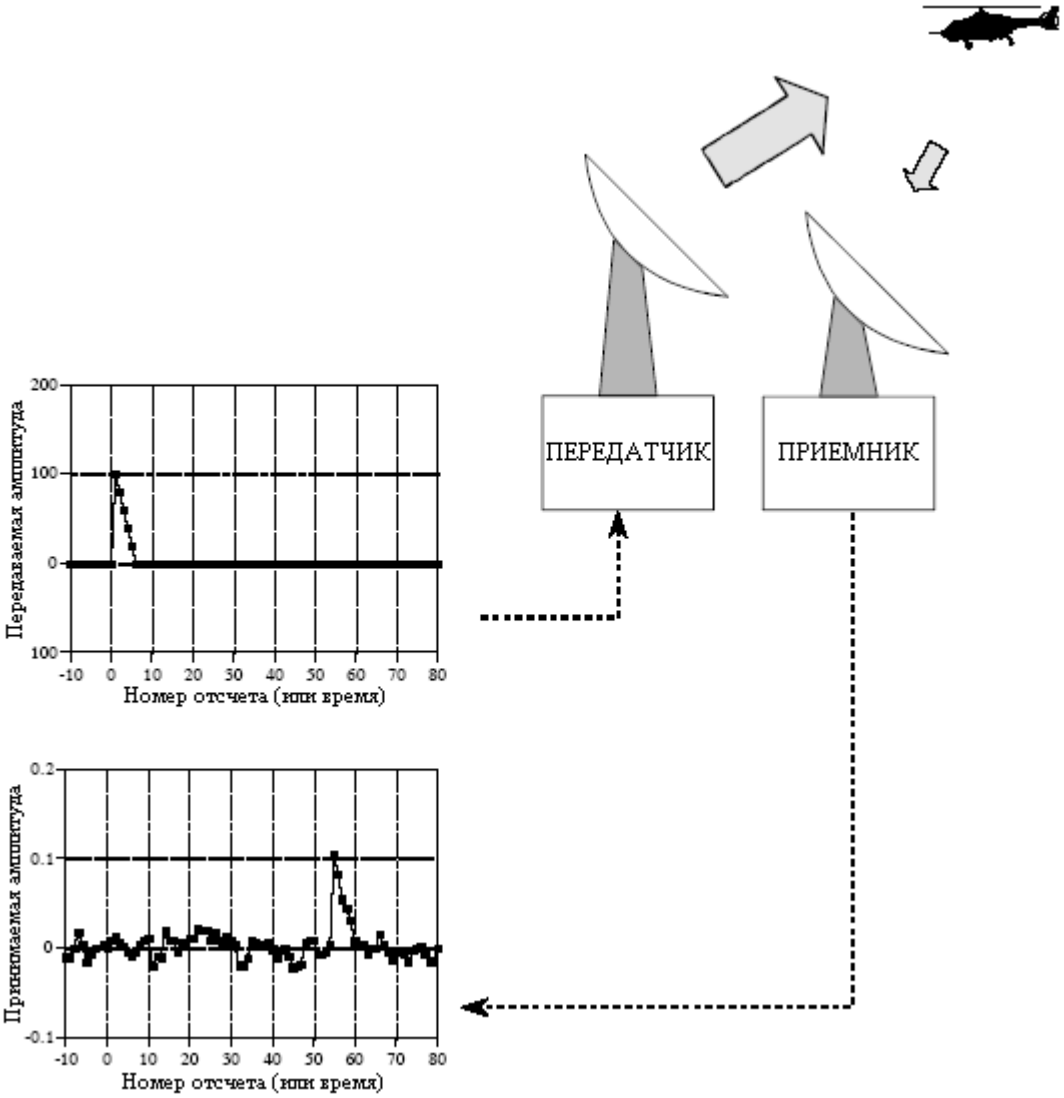


Рис. 7.13 Ключевые элементы радиолокационной системы

Если в принятом сигнале присутствует шум, он будет также присутствовать и в сигнале взаимной корреляции. Неизбежным фактом является то, что случайный шум выглядит до некоторой степени как любой, какой бы Вы не выбрали сигнал цели. Шум, наложенный на сигнал взаимной корреляции, является просто мерой такого подобия. Исключительно из-за этого шума левая и правая части пика, полученного в сигнале взаимной корреляции, являются симметричными. Это справедливо, даже если сигнал цели не является симметричным. Кроме того, ширина пика вдвое шире сигнала цели. Помните, что взаимная корреляция пытается *обнаружить* сигнал цели, а не *восстановить* его. Поэтому нет никаких причин ждать, что пик даже напомнит сигнал цели.

Корреляция является *оптимальным* методом для обнаружения волн с известной формой на фоне случайного шума. То есть, использование корреляции дает самый высокий пик над шумом, чем любые другие линейные системы. (Чтобы быть совершенно точным, она оптимальна только для *случайного белого шума*.) Использование корреляции для обнаружения известных форм волн часто называется **согласованной фильтрацией**. Более подробно об этом в главе 17.

Корреляционная машина и машина свертки одинаковы за исключением одной маленькой разницы. Как обсуждалось в прошлой главе, сигнал внутри машины свертки *перевернут* слева направо. Это означает, что номера отсчетов: 1, 2, 3, … изменяются в сторону увеличения справа налево. В корреляционной машине такого переворота нет, и номера отсчетов изменяются в сторону увеличения в нормальном направлении.

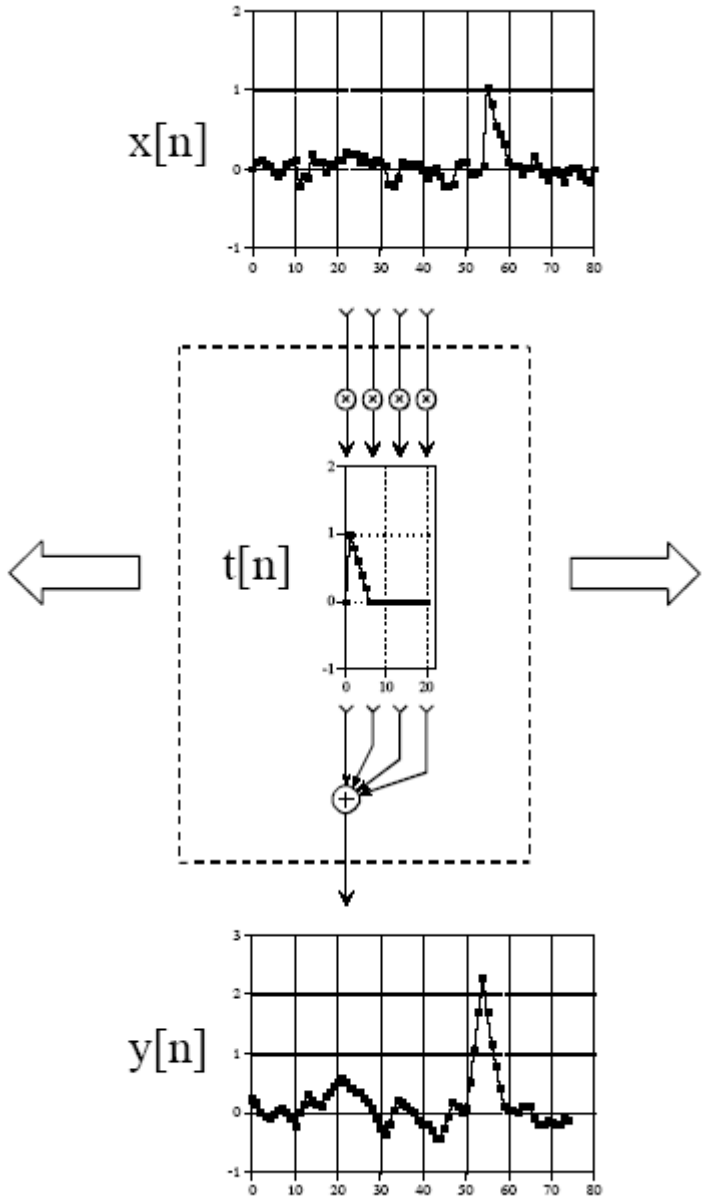


Рис. 7.14 Корреляционная машина

Поскольку лишь один этот перевернутый сигнал является единственной разницей между двумя операциями, становится возможным представлять *корреляцию*, используя ту же математику, что и для *свертки*. Это требует *предварительного переворота* одного из двух сигналов, для которых определяется корреляция, таким образом, чтобы свойственный свертке переворот слева направо отменял предварительный переворот. Например, когда $a[n]$ и $b[n]$ подвергаются свертке для получения $c[n]$, уравнение записывается как: $a[n] * b[n] = c[n]$. Для сравнения, взаимная корреляция $a[n]$ и $b[n]$ может быть записана как: $a[n] * b[-n] = c[n]$. То есть, переворот $b[n]$ слева направо достигается изменением знака индексов, т.е. $b[-n]$.

Не позволяйте математическому подобию между сверткой и корреляцией дурачить Вас; они представляют очень разные процедуры ЦОС. Свертка это соотношение между входным, выходным сигналами системы и ее импульсным откликом. Корреляция это метод обнаружения волны с известной формой на фоне шума. Похожая математика это только удачное совпадение.

Скорость

Написание программы для осуществления свертки одного сигнала с другим задача простая, требующая всего нескольких строк программного кода. Более болезненным может быть выполнение программы. Проблемой является огромное число требуемых алгоритмом сложений и умножений, выражающееся в большом времени выполнения. Как показано на примере программ в предыдущей главе, операции, отнимающее много времени, состоят из умножения двух чисел и добавления результата в аккумулятор. Остальные части алгоритма, такие как индексация массивов, являются очень быстрыми. Умножение и добавление результата в аккумулятор являются основными кирпичиками в ЦОС, и мы увидим, что они повторяются в нескольких других важных алгоритмах. В действительности скорость компьютеров ЦОС очень часто *определяется* тем, сколько времени требуется для выполнения операции умножения и аккумулирования.

Если осуществляется свертка сигнала, состоящего из N отсчетов, с сигналом, состоящим из M отсчетов, то должно быть выполнено $N \times M$ операций умножения и аккумулирования. Это можно увидеть из программ предыдущей главы. Персональные компьютеры середины 1990-х требуют около одной микросекунды на одно умножение и аккумулирование (Pentium 100 МГц, использующий числа с плавающей запятой одинарной точности см. табл. 4.6). Следовательно, свертка сигнала, содержащего 10000 отсчетов, с сигналом, содержащим 100 отсчетов, требует около одной секунды. Для обработки сигнала, содержащего один миллион точек, совместно с импульсным откликом, содержащим 3000 точек, требуется около одного часа. Десятилетием раньше (80286 12 МГц) эти вычисления потребовали бы три дня!

Проблема чрезмерного времени выполнения обычно обрабатывается одним из трех способов. Первый, просто удерживайте сигналы короткими настолько, насколько это возможно и используйте целые числа вместо чисел с плавающей запятой. Вероятно, это будет лучший компромисс между временем выполнения программы и усилиями, затраченными на программирование, если конечно Вам требуется запустить свертку всего несколько раз. Второй, Используйте компьютеры разработанные специально для ЦОС. Доступные микропроцессоры ЦОС имеют время выполнения операции умножения и аккумулирования всего в несколько десятков наносекунд. Это путь, по которому следует идти, если Вы планируете многократно выполнять свертку, так как это делается при разработке коммерческих продуктов.

Третьим решением является использование наилучших алгоритмов для осуществления свертки. Глава 17 описывает очень искусный алгоритм, называемый *БПФ сверткой*. БПФ свертка дает точно такой же результат, как и алгоритмы свертки, представленные в предыдущей главе; однако время выполнения впечатляюще снижено. Для сигналов содержащих тысячи отсчетов БПФ свертка может быть в сотни раз быстрее. Недостатком является сложность программы. Даже если Вы знакомы с этой техникой программирования, ожидайте, что Вы потратите несколько часов, пока заставите программу работать.

ГЛАВА
8

ДИСКРЕТНОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

Анализ Фурье это семейство математических методов, основанных на разложении сигналов на синусоиды. Дискретное преобразование Фурье (ДПФ) это член семейства, используемый с *оцифрованными* сигналами. Это первая из четырех глав по **действительному ДПФ**, версии дискретного преобразования Фурье, использующей для представления входного и выходного сигналов действительные числа. **Комплексное ДПФ** - более прогрессивная техника, использующая комплексные числа, будет обсуждаться в Главе 31. В этой главе мы рассмотрим математику и алгоритмы разложения Фурье, сердца ДПФ.

Семейство преобразований Фурье

Анализ Фурье назван по имени французского математика и физика Жана Батиста Жозефа Фурье (1768-1830). Хотя в эту область внесли свой вклад многие, удостоился чести Фурье за свои математические открытия и тщательное изучение возможностей практического использования этих методов. Фурье интересовался вопросами теплопереноса и в 1807 году представил доклад в Институт де Франс по применению синусоид для представления распределения температуры. Доклад содержал спорное утверждение о том, что любой непрерывный периодический сигнал мог бы быть представлен как сумма должным образом выбранных синусоидальных волн. Среди рецензентов были двое из наиболее известных математиков в истории, Жозеф Луи Лагранж (1736-1813), и Пьер Симон де Лаплас (1749-1827).

Хотя Лаплас и другие рецензенты проголосовали за опубликование доклада, Лагранж непоколебимо был против. Почти 50 лет Лагранж упорно настаивал, что такой подход не может быть использован для представления сигналов с *углами*, т.е. наклонами с разрывами, такими же, как в прямоугольных волнах. Институт де Франс преклонялся перед авторитетом Лагранжа и отклонил работу Фурье. И только через 15 лет после смерти Лагранжа доклад наконец-то был опубликован. К счастью, Фурье было чем заняться, политическая деятельность, экспедиции в Египет с Наполеоном, и попытка избежать гильотины в период после Французской Революции (буквально!).

Кто же был прав? Это двойственное решение. Лагранж был прав в его утверждении, что суммирование синусоид не может сформировать сигнал с углом. Однако Вы можете подойти к нему *очень* близко. Настолько близко, что разница между этими двумя обладает *нулевой энергией*. В этом смысле, был прав Фурье, хотя наука 18-ого столетия мало знала о концепции энергии. Это явление известно сейчас под названием *эффект Гиббса* и будет обсуждаться в Главе 11.

Рис. 8.1 иллюстрирует то, как сигнал может быть разложен на синусоидальные и косинусоидальные волны. На рис. 8.1a показан пример сигнала длиной 16 точек пробегающего значения от отсчета с номером 0 по номер 15. На рис. 8.1b показано разложение Фурье этого сигнала на девять косинусоидальных волн и девять синусоидальных волн с различной частотой и амплитудой. Хотя это и далеко от очевидного, но сумма этих 18 синусоид дает форму волны, показанную на рис. 8.1a.

Следует заметить, что возражение, сделанное Лагранжем относится только к *непрерывным* сигналам. Для *дискретных* сигналов это разложение является математически точным. Нет никакой разницы между сигналом на рис. 8.1a и суммой сигналов на рис. 8.1b, точно так же, как нет разницы между *7* и *3+4*.

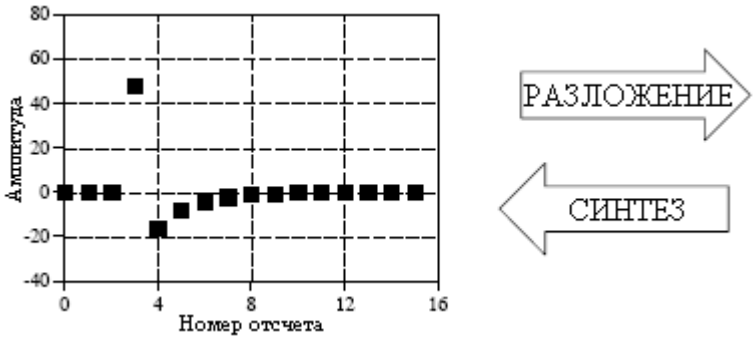


Рис. 8.1a Пример разложения Фурье

Почему используются синусоиды вместо, например, прямоугольных или треугольных волн? Запомните, существует бесконечное число способов, которыми может быть разложен сигнал. Цель разложения состоит в том, чтобы в конечном итоге иметь дело с чем-то более *простым*, чем с исходным сигналом. Например, импульсная декомпозиция позволяет в данный момент времени исследовать одну точку сигнала, что приводит к мощной технике свертки. Синусоидальные и косинусоидальные составляющие волны являются более простыми, чем исходный сигнал потому, что они обладают свойством, которого исходный сигнал не имеет: *синусоидальное качество*. Как обсуждалось в Главе 5, синусоида на входе линейной системы гарантированно дает синусоиду на ее выходе. Изменяться могут лишь амплитуда и фаза сигнала, частота и форма сигнала должны оставаться такими же. Синусоида является единственной формой волны, обладающей этим полезным свойством. Хотя *возможна* декомпозиция на прямоугольные и треугольные волны, общей причины, по которой они были бы *полезны*, не существует.

А сейчас основной тезис: как результат четырех основных типов сигналов, которые могут встречаться, *преобразование Фурье* может быть разбито на четыре категории. Сигналы могут быть либо *непрерывными* или *дискретными*, либо *периодическими* или *апериодическими*. Комбинации этих характеристик дают четыре, описываемых ниже и иллюстрируемых на рис. 8.2, категории.

Апериодические непрерывные

Включают, например, затухающие экспоненты и Гауссову кривую. Эти сигналы простираются в обе стороны положительной и отрицательной бесконечности *без* периодического повторения какого-либо образца. Преобразование Фурье для такого типа сигнала называется просто **преобразование Фурье**.

Периодические непрерывные

Здесь примером служат: синусоидальные сигналы, прямоугольные сигналы и любые другие формы волн которые регулярным образом повторяются от отрицательной до положительной бесконечности. Версия преобразования Фурье для этих сигналов называется **ряд Фурье**.

Апериодические дискретные

Такие сигналы определяются только дискретными точками между положительной и отрицательной бесконечностью и периодическим образом не повторяются. Тип преобразования Фурье для таких сигналов называется **дискретно-временным преобразованием Фурье**.

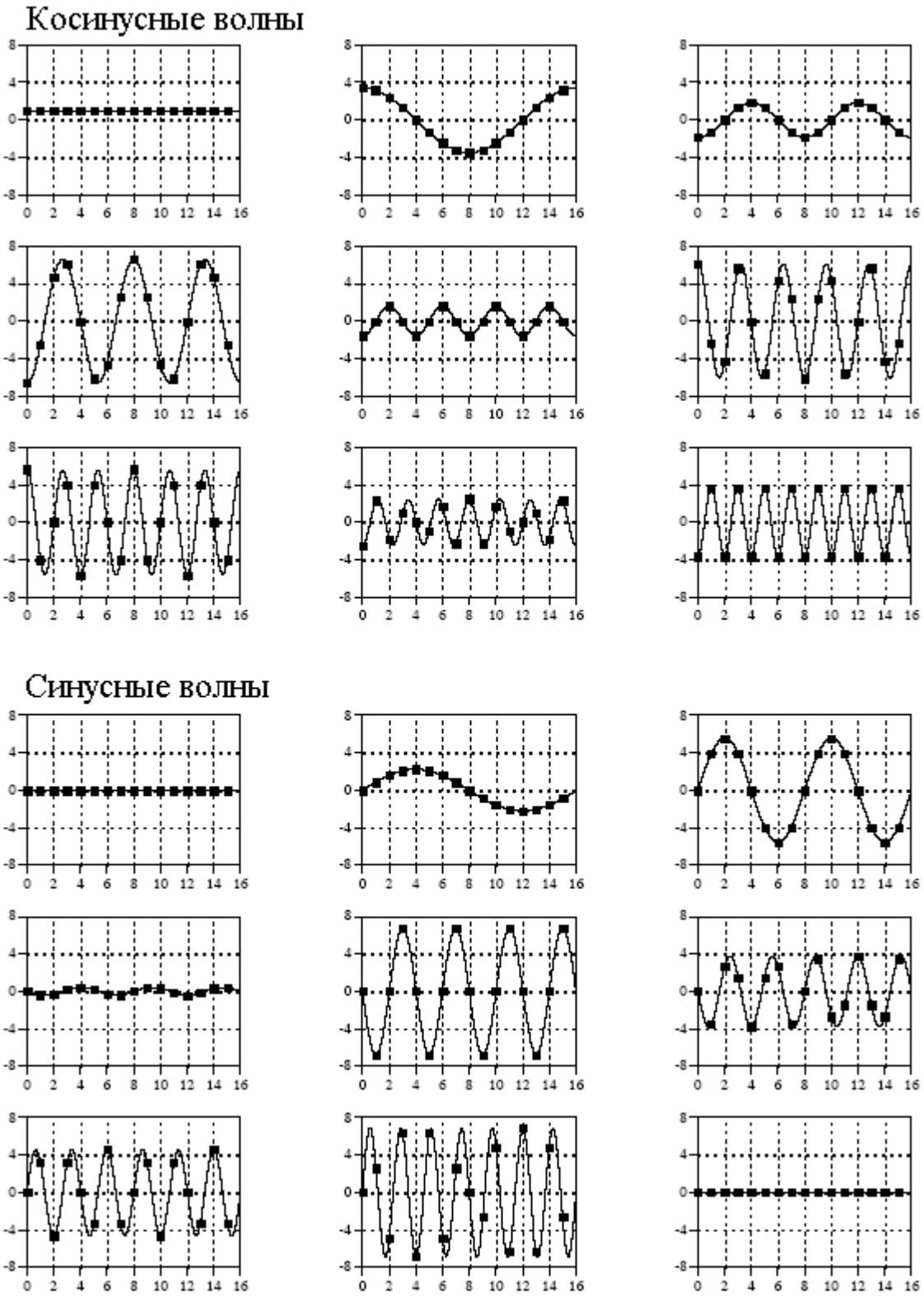


Рис. 8.1b (продолжение)

Периодические дискретные

Данными сигналами являются дискретные сигналы, повторяющиеся периодическим образом от отрицательной до положительной бесконечности. Преобразование Фурье для такого типа сигналов иногда называется дискретным рядом Фурье, но наиболее часто **дискретным преобразованием Фурье**.

Возможно, Вы думаете, что названия, данные этим четырем типам преобразований Фурье, плохо продуманы и приводят к путанице. Вы правы, названия формировались довольно случайно за более чем 200 лет. Вам ничего не остается кроме как запомнить их и двигаться дальше.

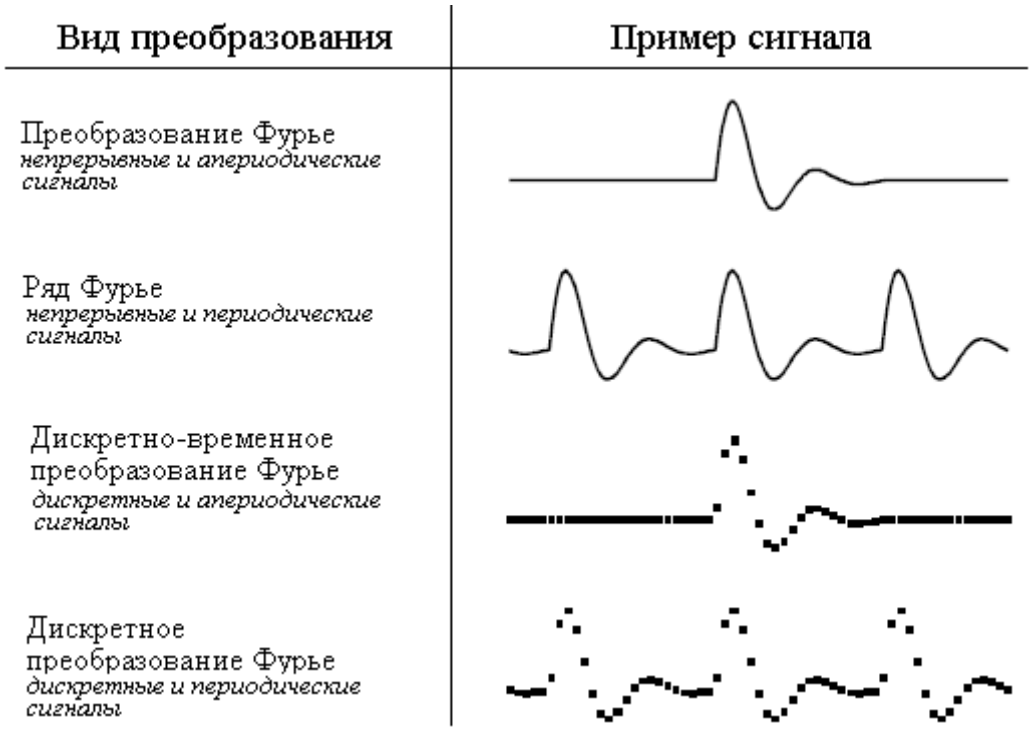


Рис. 8.2 Иллюстрация четырех преобразований Фурье

Все эти четыре класса сигналов простираются к положительной и отрицательной *бесконечности*. Постойте, скажите Вы! А что если у Вас в компьютере хранится конечное число отсчетов, скажем сигнал, сформированный из 1024 точек. Существует ли версия преобразования Фурье, использующая сигнал конечной длины? Нет, не существует. Синусные и косинусные волны *определены* как простирающиеся от отрицательной до положительной бесконечности. Вы не можете использовать группу сигналов бесконечной длины для того, что бы синтезировать что-то, имеющее конечную длину. Способ обойти эту дилемму – сделать данные конечной длины *выглядящими* как сигнал бесконечной длины. Это можно сделать, представив, что сигнал имеет бесконечное число отсчетов слева и справа от действительных точек. Если все эти “воображаемые” отсчеты имеют значение нуля, сигнал выглядит *дискретным* и *апериодическим*, и тогда, здесь используется дискретно-временное преобразование Фурье. В качестве альтернативы воображаемые отсчеты могут быть копией действительных 1024 точек. В этом случае сигналы выглядят *дискретными* и *периодическими* с периодом 1024 отсчета. Это требует применения дискретного преобразования Фурье. Как оказывается, для синтеза *апериодического* сигнала требуется *бесконечное* число синусоид. Это делает невозможным вычисление дискретно-временного преобразования Фурье с помощью компьютерного алгоритма. Таким образом, методом исключения, единственным типом преобразования Фурье, который может быть использован в ЦОС, является ДПФ. Другими словами, цифровые компьютеры могут работать только с *дискретной* информацией *конечной* длины. Когда Вы воюете с теоретическими проблемами, сражаетесь с задачами домашней работы и размышляете над математическими загадками, Вы можете пользоваться первыми тремя членами семейства преобразований Фурье. Когда же Вы садитесь за свой компьютер, Вы можете пользоваться только ДПФ. В будущих главах мы

будем кратко останавливаться и на этих других преобразованиях Фурье. А сейчас сконцентрируемся на понимании дискретного преобразования Фурье.

Вернемся назад к примеру разложения с помощью ДПФ, показанному на рис. 8.1. Судя по внешнему виду, налицо 16 точечный сигнал, разложенный на 18 синусоид, каждая из которых состоит из 16 точек. В более формальных выражениях, 16 точечный сигнал, показанный на рис. 8.1a, должен рассматриваться как отдельный период бесконечно длинного периодического сигнала. Подобным образом каждая из 18синусоид, показанных на рис. 8.1b, представляет 16 точечный сегмент из бесконечно длинной синусоиды. Действительно ли это имеет значение, рассматриваем ли мы это как 16 точечный сигнал, синтезируемый из 16 точечных синусоид, или как бесконечно длинный периодический сигнал, синтезируемый из бесконечно длинных синусоид? Ответ следующий: *обычно нет*, *но иногда да*. В предстоящих главах мы столкнемся со свойствами ДПФ, которые покажутся непреодолимыми, если сигналы рассматриваются как конечные, но становятся устранимыми, когда рассматривается периодический характер сигналов. Ключевым вопросом, который нужно понять является то, что, эта периодичность призвана для того, чтобы использовать *математический инструментарий*, т.е. ДПФ. В терминах того, где возник сигнал или как он был получен, это, обычно, не имеет значения.

Каждое из четырех преобразований Фурье может быть подразделено на **действительную** и **комплексную** версии. Действительная версия является наиболее простой, использующей обычные числа и алгебру для синтеза и разложения. Так, на рис. 8.1 показан пример **действительного ДПФ**. Комплексные версии четырех преобразований Фурье являются чрезвычайно более сложными, требующими использования *комплексных чисел*. Комплексные числа это числа подобные *3+4j*, где *j* равно $\sqrt{-1}$ (инженеры-электрики используют переменную *j*, в то время как математики используют переменную *i*). Комплексная математика очень быстро может стать довлеющей даже на тех, кто специализируется в ЦОС. Фактически, первостепенная цель этой книги состоит в том, чтобы представить основные принципы ЦОС *без* использования комплексной математики, давая возможность понять материал более широкому диапазону ученых и инженеров. Комплексное преобразование Фурье - это царство тех, кто специализируется в ЦОС и желает погрузиться в математику с головой. Если Вы склонны к этому, Главы 30-33 поведут Вас туда.

Математический термин **преобразование** широко используется в цифровой обработке сигналов, например, в преобразовании Фурье, преобразовании Лапласа, *z*-преобразовании, преобразовании Гильберта, дискретном косинус преобразовании и т.д. Так все же, что представляет собой преобразование? Чтобы ответить на этот вопрос вспомните, что представляет собой *функция*. Функция представляет собой алгоритм или процедуру, которая изменяет одно значение в другое значение. Например, $y=2x+1$ является функцией. Вы выбираете некоторое значение для *x*, подставляете его в уравнение и получаете значение для *y*. Функции могут также изменять *несколько* значений в одно значение, например: $y=2a+3b+4c$, где *a*, *b*, и *c* изменяются в *y*.

Преобразования являются прямым расширением этого и допускают возможность принимать *множественные* значения и входным и выходным величинам. Предположим у Вас есть сигнал, состоящий из 100 отсчетов. Если Вы составляете некоторое уравнение, алгоритм или процедуру для изменения этих100 отсчетов в другие 100 отсчетов, то у Вас получится преобразование. Если Вы считаете, что оно достаточно полезно, то Вы имеете полное право присвоить ему свою фамилию и разъяснить его достоинства своим коллегам. (Лучше всего это получается тогда, когда Вы являетесь выдающимся математиком 18-го столетия.) Преобразования не ограничиваются каким-то определенным видом или количеством данных. Например, Вы можете иметь 100 отсчетов дискретных данных на входе и 200 отсчетов дискретных данных на выходе. Подобным же образом Вы можете иметь непрерывный сигнал на входе и непрерывный сигнал на выходе.

Допускаются также и смешанные сигналы, дискретные на входе и непрерывные на выходе, и наоборот. Короче говоря, преобразование - это заданная процедура, изменяющая один блок данных в другой блок данных. Давайте посмотрим, как это все прилагается к рассматриваемой сейчас теме: дискретному преобразованию Фурье.

Обозначения и формат действительного ДПФ

Как показано на рис. 8.3 дискретное преобразование Фурье преобразует *N* точечный входной сигнал в два *N/2+1* точечных выходных сигнала. Входной сигнал содержит разлагаемый сигнал, в то время как два выходных сигнала содержат *амплитуды* составляющих синусных и косинусных волн (смасштабированные эдаким методом, который мы вскоре обсудим). Входной сигнал, как говорят, находится во **временной области**. Так говорят потому, что наиболее типичный вид сигнала, попадающий на вход ДПФ, состоит из отсчетов, взятых через одинаковые интервалы *времени*. Конечно, в ДПФ может быть введен любой вид дискретных данных, вне зависимости от того, как они были получены. Когда Вы видите в Фурье анализе выражение “временная область”, то оно может относиться действительно к отсчетам взятым в течение некоторого времени или оно может быть общей ссылкой на любой дискретный сигнал, подвергающийся декомпозиции. Выражение **частотная область** используется для описания амплитуд синусных и косинусных волн (включая масштабирование спектра, которое мы обещали объяснить).

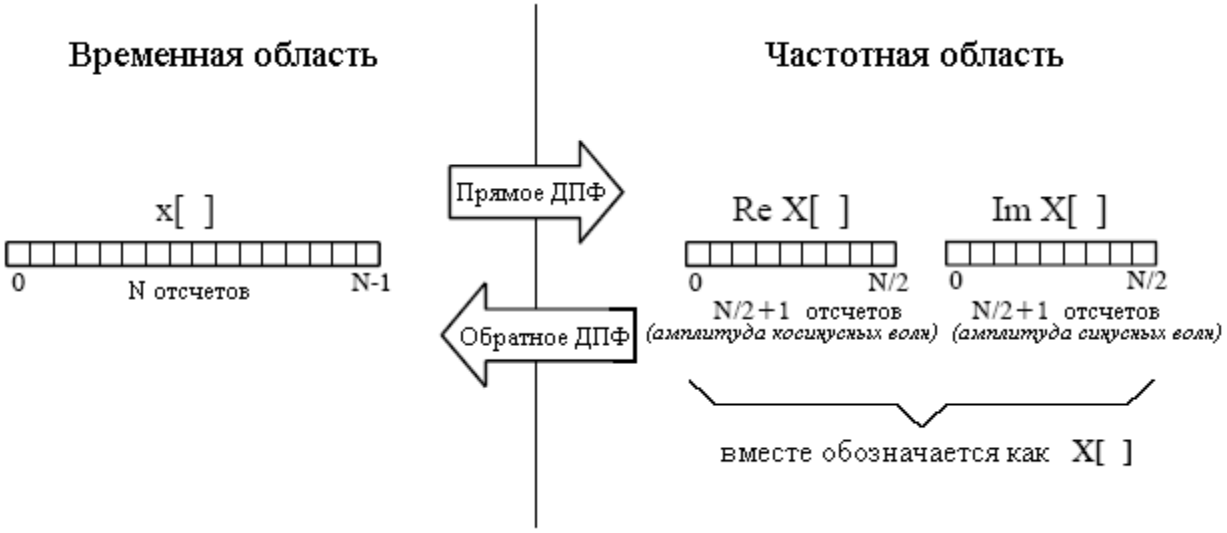


Рис. 8.3 Терминология ДПФ

Частотная область содержит точно такую же информацию, как и временная область, но просто в другой форме. Если Вам известна одна область, Вы можете вычислить другую. При заданном сигнале временной области процесс вычисления частотной области называется **разложением** (**декомпозицией** – прим. перев.), **анализом**, **прямым ДПФ** или просто **ДПФ**. Если Вам известна частотная область, вычисление временной области называется **синтезом** или **обратным ДПФ**. Как синтез, так и анализ могут быть представлены в форме уравнения и компьютерного алгоритма.

Число отсчетов во временной области обычно обозначается **переменной *N***. Хотя *N* может быть любым положительным целым числом, обычно выбираются числа кратные степени числа два, т.е. 128, 256, 512, 1024 и т.д. Этому есть две причины. Во-первых, для хранения цифровых данных используется двоичная адресация, при которой естественной длиной сигнала становятся величины кратные степени двойки. Во-вторых, наиболее

эффективный алгоритм для вычисления ДПФ, быстрое преобразование Фурье (БПФ), работает с *N* кратным степени двойки. Обычно *N* выбирается между *32* и *4096*. В большинстве случаев номера отсчетов изменяются от *0* до *N-1*, а не от *1* до *N*.

Стандартным обозначением в ЦОС для представления сигналов временной области, является использование **строчных букв** таких, как *x*[ ], *y*[ ] и *z*[ ]. Соответствующие **заглавные буквы** используются для представления частотных областей этих сигналов, т.е. *X*[ ], *Y*[ ] и *Z*[ ]. Для иллюстрации предположим, что в *x*[ ] находится *N* точечный сигнал временной области. Частотная область этого сигнала называется *X*[ ] и состоит из двух частей, каждая из которых представляет собой массив, состоящий из *N/2+1* отсчетов. Они называются **действительной частью *X*[ ]**, записывается как ***Re X[ ]***, и **мнимой частью *X*[ ]**, записывается как ***Im X[ ]***. Значения содержащиеся в *Re X[ ]* - это амплитуды косинусных волн, в то время как значения содержащиеся в *Im X[ ]* - это амплитуды синусных волн (не волнуйтесь в данный момент о масштабирующих коэффициентах). По мере изменения временной области от *x*[*0*] до *x*[*N-1*], сигнал частотной области изменяется от *Re X[0]* до *Re X[N/2]* и от *Im X[0]* до *Im X[N/2]*. Внимательно изучите эти обозначения, они крайне необходимы для понимания уравнений в ЦОС. К сожалению, некоторые языки программирования не проводят различия между заглавными и строчными буквами, оставляя название переменной на усмотрение программиста. В программах в этой книге для размещения сигнала временной области используется массив XX[ ] и массивы REX[ ] и IMX[ ] для размещения сигналов частотной области.

Названия *действительная часть* и *мнимая часть* происходят от комплексного ДПФ, где они используются для того, чтобы различать *действительные* и *мнимые* числа. Ничего такого сложного для действительного ДПФ не требуется. До тех пор пока Вы не дойдете до Главы 31, просто думайте, что “действительная часть” означает *амплитуды косинусных волн*, в то время как “мнимая часть” означает *амплитуды синусных волн*. Не допускайте, чтобы эти, наводящие на размышления названия увели Вас в сторону, все здесь использует обычные числа.

Аналогично, не допускайте того, чтобы Вас вводила в заблуждение *длина* сигналов частотной области. Часто в литературе по ЦОС можно увидеть утверждения подобные следующему: ”ДПФ преобразует *N* точечный сигнал временной области в *N* точечный сигнал частотной области”. Это относится к *комплексному* ДПФ, где каждая “точка” является комплексным числом (состоит из действительной и мнимой частей). А теперь, сосредоточьтесь на изучении действительного ДПФ, достаточно скоро подойдет трудная математика.

Независимая переменная частотной области

На рис. 8.4 показан пример ДПФ с *N=128*. Сигнал временной области находится в массиве: от *x*[*0*] до *x*[*127*]. Сигналы частотной области находятся в двух массивах: от *Re X[0]* до *Re X[64]* и от *Im X[0]* до *Im X[64]*. Заметим, что *128* точкам во временной области соответствуют *65* точек в каждом из сигналов частотной области с частотными индексами, изменяющимися от *0* до *64*. То есть, *N* точкам во временной области соответствуют *N/2+1* точка в частотной области (не *N/2* точек). Запамятовать об этой дополнительной точке - это типичная ошибка в программах ДПФ.

Горизонтальная ось частотной области может быть обозначена **четырьмя различными способами**, каждый из которых является обычным в ЦОС. В первом способе горизонтальная ось размечается от *0* до *64* в соответствии с множеством отсчетов от *0* до *N/2*. Когда используется такая разметка, то индекс частотной области является целым числом, например, *Re X[k]* и *Im X[k]*, где *k* изменяется от *0* до *N/2* с шагом равным единице. Программистам нравится этот метод, поскольку он похож на то, как они пишут

программный код, используя индекс для поучения доступа к элементам массива. Такое обозначение используется на рис 8.4b.

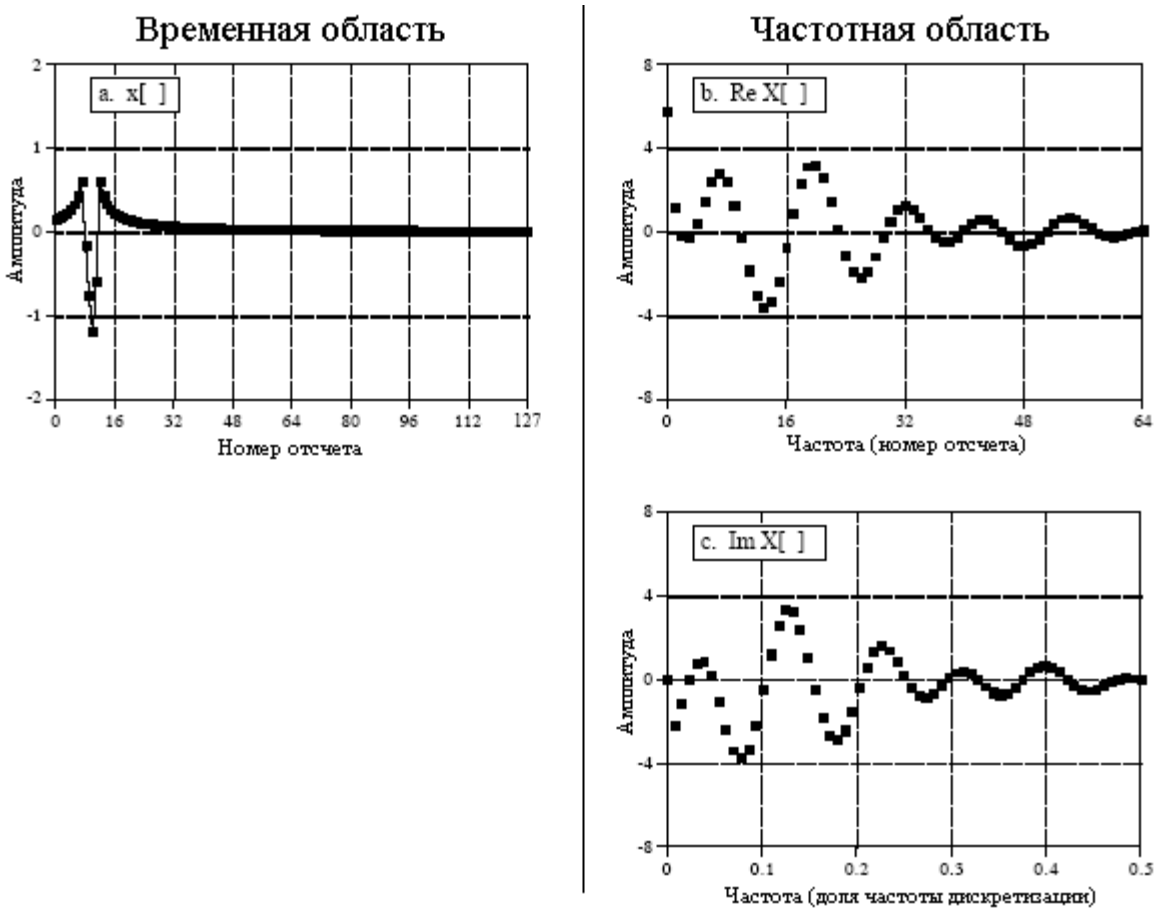


Рис. 8.4 Пример ДПФ

Во втором методе, используемом на рис. 8.4c, горизонтальная ось размечена в *долях частоты дискретизации*. Это означает, что значения вдоль горизонтальной оси всегда изменяются между *0* и *0,5*, поскольку дискретные данные могут содержать только частоты лежащие между постоянной составляющей и частотой равной половине частоты дискретизации. Индекс, используемый для частоты при таком обозначении - *f*. Действительная и мнимая части записываются как: *Re X[f]* и *Im X[f]*, где *f* принимает *N/2+1* равноотстоящих друг от друга значений находящихся между *0* и *0,5*. Для того чтобы преобразовать первый вид обозначения - *k* во второй вид обозначения – *f*, делят значения вдоль горизонтальной оси на *N*. То есть *f=k/N*. Большинство рисунков в этой книге использует этот второй метод, подчеркивая, что дискретные сигналы содержат только частоты, находящиеся между *0* и *0,5* от частоты дискретизации.

Третий стиль подобен второму за исключением того, что значения вдоль горизонтальной оси умножаются на 2π. Индекс, используемый при таком обозначении – ω, греческая строчная буква омега. При данном обозначении действительная и мнимая части записываются как: *Re X[ω]* и *Im X[ω]*, где ω принимает *N/2+1* равноотстоящих друг от друга значений, находящихся между *0* и π. Параметр ω называется **естественной частотой** и измеряется в **радианах**. Этот стиль базируется на мысли, что в окружности - 2π радиан. Математики любят этот метод, поскольку он делает уравнения более короткими. Например, рассмотрим, как косинусная волна записывается в каждом из этих трех обозначений: используя *k* – $c[n]=\cos(2\pi kn/N)$, используя *f* – $c[n]=\cos(2\pi fn)$ и используя ω – $c[n]=\cos(\omega n)$.

Четвертый способ - разметить горизонтальную ось в обозначениях аналоговых частот, используемых в *специфических* приложениях. Например, если исследуемая система имеет частоту дискретизации равную *10* кГц (т.е. *10000* отсчетов в секунду), график частотной области будет изменяться от *0* до *5* кГц. Этот метод обладает преимуществом представления частотных данных в терминах значений *реального мира*. Недостатком метода является то, что он привязан к определенной частоте дискретизации и, следовательно, не применим к разработке общего алгоритма ЦОС, такого как проектирование цифровых фильтров.

Все эти четыре системы обозначений применяются в ЦОС, и при переходе от одной системы к другой Вам нужно чувствовать себя уверенно. Это относится как к графикам, так и к математическим уравнениям. Для того чтобы определить какое из обозначений используется, посмотрите на независимую переменную и диапазон ее значений. Вы найдете одно из четырех обозначений: k (или некоторый другой индекс целого) изменяющееся от *0* до *N/2*; f изменяющееся от *0* до *0,5*; ω изменяющееся от *0* до π; или частоту, выраженную в герцах, изменяющуюся от постоянной составляющей до половины фактической частоты дискретизации.

Базисные функции ДПФ

Синусные и косинусные волны, используемые в ДПФ, обычно называют **базисными функциями** ДПФ. Если говорить другими словами, результатом ДПФ является набор чисел, представляющих собой амплитуды. Базисными же функциями является набор синусоидальных и косинусоидальных волн с *единичной* амплитудой. Если Вы назначите каждую из амплитуд (частотная область) соответствующей синусной или косинусной волне (базисные функции), то в результате получится набор *смасштабированных* синусных и косинусных волн, который при суммировании сформирует сигнал временной области.

Базисные функции ДПФ получаются из уравнений:

$$c_k[i] = \cos(2\pi ki / N);$$
$$s_k[i] = \sin(2\pi ki / N), \tag{8.1}$$

где $c_k[\ ]$ – косинусоидальная волна для амплитуд, находящихся в *Re X[k]* и $s_k[\ ]$ – синусоидальная волна для амплитуд, находящихся в *Im X[k]*. Например, на рис. 8.5 показаны некоторые из 17 синусных и 17 косинусных волн, используемых в точечном ДПФ с *N=32*. Поскольку эти синусоиды для формирования входного сигнала складываются, они должны иметь одинаковую *длину* с входным сигналом. В этом случае каждая из них содержит 32 точки, изменяющиеся от *i=0* до *31*. Параметр k устанавливает частоту каждой синусоиды. В частности $c_1[\ ]$ является косинусной волной, совершающей на N точках *один* полный цикл, $c_5[\ ]$ является косинусной волной, совершающей на N точках *пять* полных циклов и т.д. Это важная концепция в понимании базисных функций; частотный параметр k равен числу полных циклов совершаемых на N точках сигнала.

Давайте подробнее рассмотрим несколько этих базисных функций. На рис. 8.5a показана косинусная волна $c_0[\ ]$. Это косинусная волна с нулевой частотой и с постоянным значением равным единице. Это означает, что *Re X[0]* содержит среднее значение всех точек в сигнале временной области. В электронике говорят, что *Re X[0]* содержит **смещение по постоянной составляющей**. Синусная волна нулевой частоты $s_0[\ ]$, показанная на рис. 8.5b, это сигнал состоящий из всех *нулей*. Величина *Im X[0] к делу не относится*, поскольку не может повлиять на синтезируемый сигнал временной области и всегда устанавливается в ноль. Подробнее об этом вскоре.

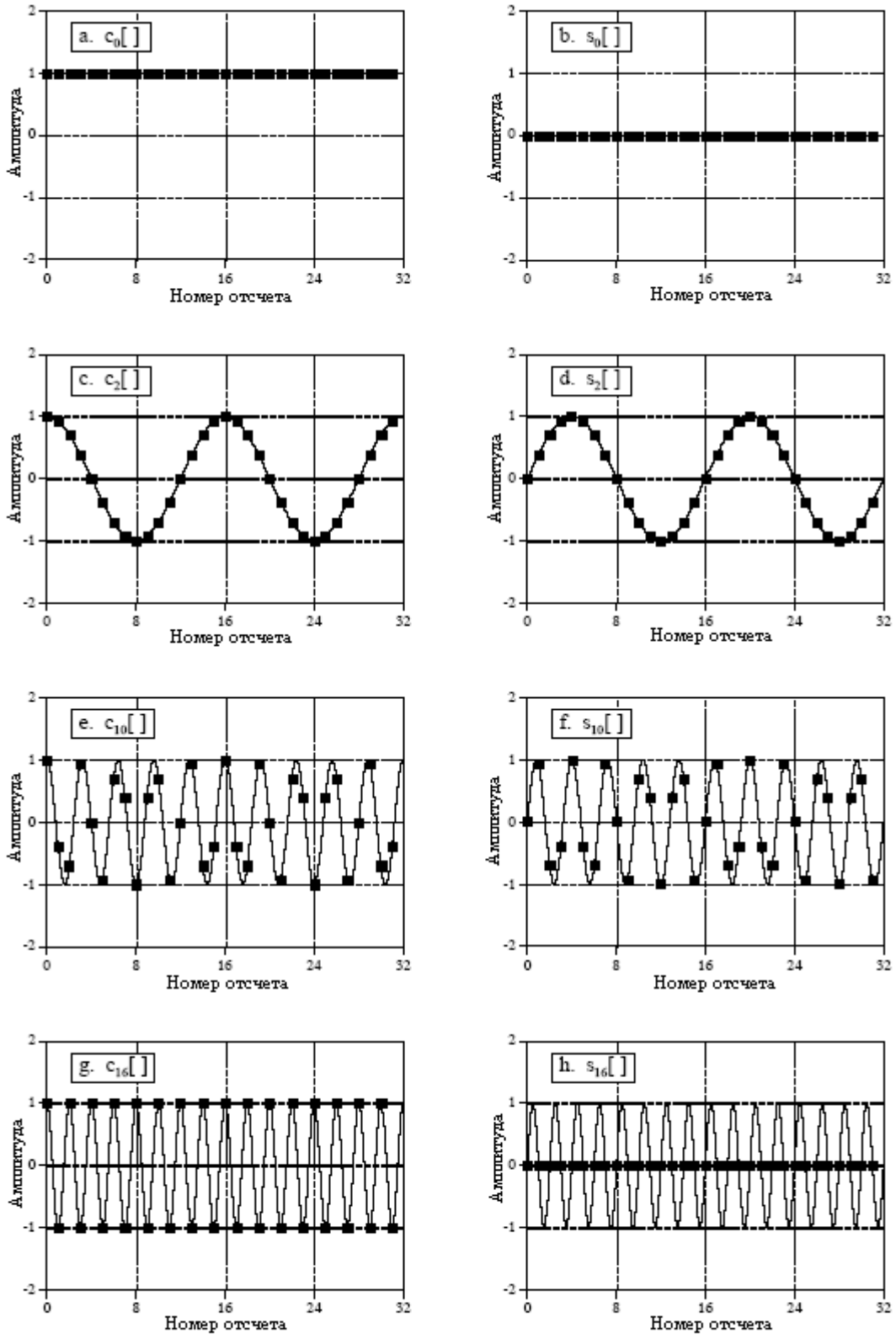


Рис. 8.5 Базисные функции ДПФ

На рис. 8.5c и рис. 8.5d показаны синусоиды $c_2[\ ]$ и $s_2[\ ]$, совершающие *два* полных цикла на N точках. Они соотносятся с *Re X[2]* и *Im X[2]*, соответственно. Подобным же образом на рис. 8.5e и рис. 8.5f показаны синусоиды $c_{10}[\ ]$ и $s_{10}[\ ]$ совершающие *десять*

полных циклов на *N* точках. Данные синусоиды соответствуют амплитудам, находящимся в множествах *Re X[10]* и *Im X[10]*. Проблема заключается в том, что отсчеты на рис. 8.5e и рис. 8.5f больше *не выглядят* похожими на синусные и косинусные волны. Если бы на этих рисунках небыли представлены непрерывные линии, Вам было бы очень сложно даже обнаружить конфигурацию формы волны. Это может Вас немного обеспокоить, но не волнуйтесь относительно этого. С математической точки зрения эти отсчеты формируют дискретную синусоиду, даже если Ваш взгляд не может проследить за конфигурацией образуемой отсчетами.

Самые высокие частоты в базисных функциях показаны на рис. 8.5g и рис. 8.5h. Это $c_{N/2}[\ ]$ и $s_{N/2}[\ ]$ или в данном примере $c_{16}[\ ]$ и $s_{16}[\ ]$. Дискретная косинусная волна периодически изменяется по значению от 1 до –1, что может быть интерпретировано как снятие отсчетов непрерывной синусоиды в моменты ее прохождения через *пики*. Дискретная синусная волна, напротив, содержит все нули, как результат снятия отсчетов в моменты *прохождения* ее *через ноль*. Это делает значение *Im X[N/2]* таким же, как *ImX[0]*, всегда равным нулю и не влияющим на синтез сигнала временной области.

А вот и загадка: Если на вход ДПФ поступает *N* отсчетов, а на выходе появляется *N+2* отсчета, то откуда берется дополнительная информация? Ответ: Два выходных отсчета не содержат *никакой* информации, что позволяет другим *N* отсчетам быть полностью независимыми. Как Вы вероятно уже догадались, точками, которые не несут никакой информации, являются *Im X[0]* и *Im X[N/2]*, отсчеты, которые всегда имеют значение равное нулю.

Синтез, вычисление обратного ДПФ

Подытоживая все сказанное ранее, мы можем записать **уравнение синтеза**:

$$x[i] = \sum_{k=0}^{N/2} \mathrm{Re}\,\overline{X}[k]\cos(2\pi ki/N) + \sum_{k=0}^{N/2} \mathrm{Im}\,\overline{X}[k]\sin(2\pi ki/N). \qquad (8.2)$$

Выражаясь словами, любой *N* точечный сигнал *x*[*i*] может быть получен с помощью сложения *N/2+1* косинусных волн и *N/2+1* синусных волн. Амплитуды косинусных и синусных волн содержатся в множествах $\mathrm{Im}\,\overline{X}[k]$ и $\mathrm{Re}\,\overline{X}[k]$, соответственно. Уравнение синтеза умножает эти амплитуды на базисные функции для создания набора смасштабированных синусных и косинусных волн. Суммирование смасштабированных синусных и косинусных волн дает сигнал временной области *x*[*i*].

В уравнении (8.2) вместо *Im X[k]* и *Re X[k]* множества называются $\mathrm{Im}\,\overline{X}[k]$ и $\mathrm{Re}\,\overline{X}[k]$. Это потому, что *амплитуды необходимые для синтеза* (называемые в данном обсуждении $\mathrm{Im}\,\overline{X}[k]$ и $\mathrm{Re}\,\overline{X}[k]$) немного отличаются от частотной области сигнала (обозначаемой с помощью *Im X[k]* и *Re X[k]*). Это связано с появлением масштабирующего коэффициента, о котором мы упоминали ранее. Хотя такое преобразование является только простой нормализацией, в компьютерных программах здесь часто случаются ошибки. Хорошенько просматривайте его! В форме уравнения преобразование между этими двумя множествами дается как:

$$\mathrm{Re}\,\overline{X}[k] = \frac{\mathrm{Re}\,X[k]}{N/2};$$

$$\operatorname{Im}\bar{X}[k] = -\frac{\operatorname{Im}X[k]}{N/2},$$

за исключением двух особых случаев: (8.3)

$$\operatorname{Re}\bar{X}[0] = \frac{\operatorname{Re}X[0]}{N};$$

$$\operatorname{Re}\bar{X}[N/2] = \frac{\operatorname{Re}X[N/2]}{N}.$$

Предположим, что Вам дано представление в частотной области и Вас просят синтезировать соответствующий сигнал временной области. Для того чтобы начать, Вы должны найти амплитуды синусных и косинусных волн. Другими словами, даны *Im X[k]* и *Re X[k]*, и Вы должны найти $\operatorname{Im}\bar{X}[k]$ и $\operatorname{Re}\bar{X}[k]$. Уравнение (8.3) показывает этот процесс в математической форме. Для того чтобы реализовать его, в компьютерной программе должны быть выполнены три действия. Первое, разделите все значения частотной области на *N/2*. Второе, измените знак у всех мнимых значений. Третье, разделите первый и последний отсчеты в действительной части *Re X[0]* и *Re X[N/2]* на два. Это даст амплитуды, требуемые для синтеза описываемого уравнением (8.2). Вместе взятые уравнения (8.2) и (8.3) *определяют* обратное ДПФ.

Полностью обратное ДПФ показано в распечатке компьютерной программы в таблице 8.1. Существует всего два способа, которыми может быть осуществлено программирование синтеза (уравнение (8.2)), и оба они показаны здесь. В первом способе, за один раз генерируется одна из смасштабированных синусоид и прибавляется к аккумулирующему массиву, который в конце становится сигналом временной области. Во втором способе, за один раз вычисляется каждый отсчет сигнала временной области, как сумма всех соответствующих отсчетов синусных и косинусных волн. Оба способа дают один и тот же результат. Различие между этими двумя программами весьма незначительно; во время синтеза внутренние и внешние циклы меняются местами.

Рис. 8.6 иллюстрирует работу обратного ДПФ и небольшие различия между частотной областью и амплитудами, требуемыми для синтеза. На рис. 8.6a показан пример сигнала который мы желаем синтезировать: некоторый импульс с амплитудой равной 32 в нулевом отсчете. На рис. 8.6b показано представление этого сигнала в частотной области. Действительная часть частотной области является постоянной величиной равной 32. Мнимая часть (не показана) состоит из нулей. Как обсуждается в следующей главе это важная *пара* ДПФ: импульс во временной области соответствует постоянной величине в частотной области. А пока, важным является то, что то, что изображено на рис. 8.6b является ДПФ того, что изображено на рис. 8.6a, а то, что изображено на рис. 8.6a, является обратным ДПФ того, что изображено на рис. 8.6b.

Уравнение (8.3) используется для преобразования сигнала частотной области (рис. 8.6b) в амплитуды косинусных волн (рис. 8.6). Как показано все косинусные волны имеют амплитуду равную *двум*, исключением являются отсчеты *0* и *16*, значения которых равны *единице*. Амплитуды синусных волн в этом примере не показаны, поскольку их значения равны нулю и, следовательно, не вносят никакого вклада. Для преобразования амплитуд косинусных волн (рис. 8.6b) в сигнал временной области (рис. 8.6a) используется, затем, уравнение (8.2), уравнение синтеза.

Таблица 8.1

```
100 'THE INVERSE DISCRETE FOURIER TRANSFORM
110 'The time domain signal, held in XX[ ], is calculated from the frequency domain signals,
120 'held in REX[ ] and IMX[ ].
130 '
140 DIM XX[511]                    'XX[ ] holds the time domain signal
150 DIM REX[256]                   'REX[ ] holds the real part of the frequency domain
160 DIM IMX[256]                   'IMX[ ] holds the imaginary part of the frequency domain
170 '
180 PI = 3.14159265                'Set the constant, PI
190 N% = 512                       'N% is the number of points in XX[ ]
200 '
210 GOSUB XXXX                     'Mythical subroutine to load data into REX[ ] and IMX[ ]
220 '
230
240 '                              'Find the cosine and sine wave amplitudes using Eq. 8-3
250 FOR K% = 0 TO 256
260 REX[K%] = REX[K%] / (N%/2)
270 IMX[K%] = -IMX[K%] / (N%/2)
280 NEXT K%
290 '
300 REX[0] = REX[0] / 2
310 REX[256] = REX[256] / 2
320 '
330 '
340 FOR I% = 0 TO 511              'Zero XX[ ] so it can be used as an accumulator
350 XX[I%] = 0
360 NEXT I%
370 '
380 '                              Eq. 8-2 SYNTHESIS METHOD #1. Loop through each
390 '                              frequency generating the entire length of the sine and cosine
400 '                              waves, and add them to the accumulator signal, XX[ ]
410 '
420 FOR K% = 0 TO 256              'K% loops through each sample in REX[ ] and IMX[ ]
430 FOR I% = 0 TO 511              'I% loops through each sample in XX[ ]
440 '
450 XX[I%] = XX[I%] + REX[K%] * COS(2*PI*K%*I%/N%)
460 XX[I%] = XX[I%] + IMX[K%] * SIN(2*PI*K%*I%/N%)
470 '
480 NEXT I%
490 NEXT K%
500 '
510 END
```

Альтернативные коды для строк с 380 по 510

```
380 '                              Eq. 8-2 SYNTHESIS METHOD #2. Loop through each
390 '                              sample in the time domain, and sum the corresponding
400 '                              samples from each cosine and sine wave
410 '
420 FOR I% = 0 TO 511              'I% loops through each sample in XX[ ]
430 FOR K% = 0 TO 256              'K% loops through each sample in REX[ ] and IMX[ ]
440 '
```

```
450 XX[I%] = XX[I%] + REX[K%] * COS(2*PI*K%*I%/N%)
460 XX[I%] = XX[I%] + IMX[K%] * SIN(2*PI*K%*I%/N%)
470 '
480 NEXT K%
490 NEXT I%
500 '
510 END
```

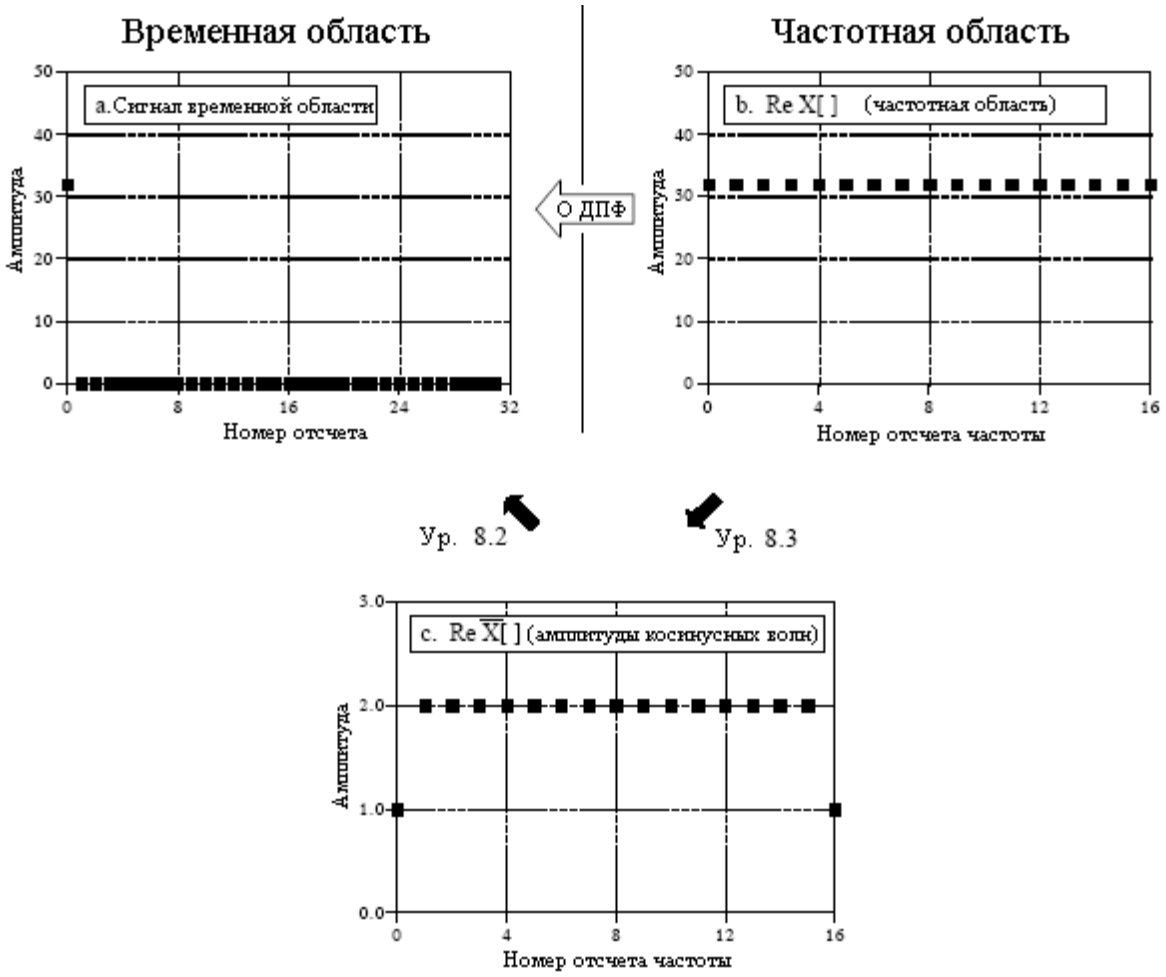


Рис. 8.6 Пример обратного ДПФ

Все выше сказанное описывает то, *как* частотная область отличается от синусоидальных амплитуд, но не объясняет, *почему* она отличается. Отличие получаются потому, что частотная область определена как **спектральная плотность**. Как все это работает, показано на рис. 8.7. Пример, показанный на этом рисунке представляет собой действительную часть частотной области *32* точечного сигнала. Как Вы, должно быть, и предполагали, изменяющиеся от *0* по *16* отсчеты представляют 17 равноудаленных друг от друга частот, находящихся между *0* и *1/2* частоты дискретизации. *Спектральная плотность* описывает количество сигнала (амплитуды) присутствующее *в единице полосы частот*. Для преобразования синусоидальных амплитуд в спектральную плотность разделите каждую амплитуду на полосу частот представленную каждой амплитудой. Последнее вызывает следующий вопрос: как мы определяем полосу частот каждой из дискретных частот в частотной области?

Как показано на рисунке, полоса частот может быть определена при помощи проведения разделительных линий между отсчетами. Например, отсчет номер *5* располагается в полосе частот между *4,5* и *5,5*; отсчет номер *6* в полосе частот между *5,5* и *6,5* и т.д. Выраженная в виде доли от общей полосы частот (т.е. *N/2*), полоса частот каждого отсчета есть *2/N*. Исключением из этого являются отсчеты по обоим концам, которые имеют половину от этой полосы частот, т.е. *1/N*. Это объясняет масштабирующий коэффициент *2/N* между синусоидальными амплитудами и частотной областью, а также дополнительный коэффициент *два* необходимый для первого и последнего отсчетов.

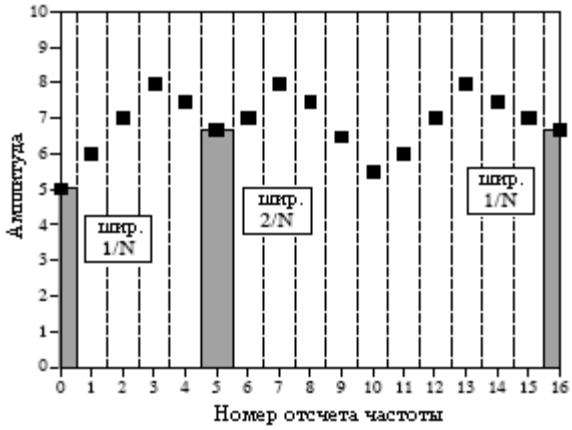


Рис. 8.7 Ширина диапазона отсчетов частотной области

Почему ставится минус перед мнимой частью? Это делается исключительно только для того, чтобы сделать *действительное ДПФ* совместимым с его старшим братом, *комплексным ДПФ*. В Главе 29 мы покажем, что это необходимо для того, чтобы заставить математику комплексного ДПФ работать. Большое число авторов, если имеют дело только с действительным ДПФ, не включают этот минус. По этой же причине многие авторы не включают даже масштабирующий коэффициент *2/N*. Будьте готовы в некоторых обсуждениях обнаружить отсутствие их обоих. Здесь они включены по исключительно важной причине: наиболее эффективный способ вычислить ДПФ - сделать это при помощи алгоритма быстрого преобразования Фурье (БПФ), представленного в Главе 12. БПФ генерирует частотную область, определяемую в соответствии с уравнениями (8.2) и (8.3). Если Вы начнете беспорядочно обращаться с этими нормализующими коэффициентами, то ваши программы, содержащие БПФ, как ожидается, и не будут собираться работать.

Анализ, вычисление ДПФ

ДПФ может быть вычислено тремя полностью отличающимися друг от друга способами. Во-первых, к проблеме можно подойти как к набору *совместно решаемых уравнений*. Такой подход полезен для понимания ДПФ, но он очень неэффективен для практического использования. Второй метод построен на идее из предыдущей главы: *корреляции*. Он базируется на обнаружении в другом сигнале известной формы волны. Третий метод, названный быстрым преобразованием Фурье (БПФ), является остроумным алгоритмом, который осуществляет разложение ДПФ, содержащего *N* точек, в *N* ДПФ, каждое из которых содержит одну единственную точку. Обычно БПФ в сотни раз быстрее, чем другие методы. Здесь обсуждаются первые два метода, в то время как БПФ является темой Главы 12. Очень важно запомнить, что все три этих метода дают одинаковые результаты. Какой же из них Вам следует использовать? В реальной практике

корреляция является предпочтительной техникой, если ДПФ содержит менее приблизительно 32 точек, в противном случае используется БПФ.

ДПФ посредством совместно решаемых уравнений

Подумайте о вычислении ДПФ следующим образом. Вам даны *N* значений во временной области и Вас просят вычислить *N* значений в частотной области (игнорируя два значения в частотной области, которые, как Вы знаете, являются нулями). Основания алгебры дают следующий ответ: для того, чтобы найти *N* неизвестных, Вы должны иметь возможность написать *N* линейных независимых уравнений. Для того чтобы сделать это, возьмите первый отсчет из каждой синусоиды и сложите их вместе. Сумма должна быть равна первому отсчету сигнала во временной области, так получается первое уравнение. Подобным образом может быть написано уравнение для каждой из оставшихся точек в сигнале временной области, давая в результате требуемые *N* уравнений. Затем, при помощи использования точных методов решения систем уравнений, таких как метод гауссовых исключений (метод Гаусса – прим. перев.), может быть найдено решение. К сожалению, этот способ требует огромного числа вычислений и фактически никогда не используется в ЦОС. Однако он очень важен по другим причинам, он показывает, *почему* возможно разложение сигнала на синусоиды, *сколько* требуется синусоид, и что базисные функции должны быть линейно независимыми (более подробно об этом вскоре).

ДПФ посредством корреляции

Давайте пойдем по лучшему пути, *стандартному пути* вычисления ДПФ. Пример покажет, как работает этот метод. Предположим, что мы пытаемся вычислить ДПФ *64* точечного сигнала. Это означает, что нам нужно вычислить *33* точки в действительной части и *33* точки в мнимой части частотной области. В этом примере мы только покажем, как вычислить один отсчет *Im X[3]*, т.е. амплитуду синусной волны, содержащей три полных периода между точкой *0* и точкой *63*. Подобным же образом вычисляются все другие значения частотной области.

Рис. 8.8 иллюстрирует использование корреляции для вычисления *Im X[3]*. Рис. 8.8a и рис. 8.8b показывают два примера сигналов временной области названных соответственно *x1*[ ] и *x2*[ ]. Первый сигнал *x1*[ ] составлен из ничего другого, как из синусоидальной волны, содержащей три периода между точкой *0* и точкой *63*. Напротив, *x2*[ ] составлен из нескольких синусоидальных и косинусоидальных волн, ни одна из которых не содержит три периода между точками *0* и *63*. Эти два сигнала иллюстрируют то, что должен делать алгоритм для вычисления *Im X[3]*. После ввода *x1*[ ] алгоритм должен выдать значение равное *32* амплитуде синусоидальной волны, содержащейся в сигнале (измененной в соответствии с масштабирующим коэффициентом уравнения (8.3)). Для сравнения, когда в алгоритм вводится другой сигнал *x2*[ ], должно быть получено значение *0*, указывающее что эта конкретная синусная волна не содержится в этом сигнале.

Концепция корреляции была представлена в Главе 7. Как Вы помните, для обнаружения известной формы волны, содержащейся в другом сигнале, два сигнала перемножают, и в результирующем сигнале все точки складывают. Единственное число, получающееся в результате этой процедуры, является мерой того, насколько похожи эти два сигнала. Этот подход иллюстрируется на рис. 8.8. Рисунки 8.8c и 8.8d оба показывают сигнал, который мы ищем, синусоидальную волну, содержащую *3* периода между точкой *0* и точкой *63*. Рисунок 8.8e показывает результат перемножения 8.8a и 8.8c. В то время как рис. 8.8f показывает результат перемножения 8.8b и 8.8d. Сумма всех точек на рис. 8.8e равна *32*, тогда как сумма всех точек на рис. 8.8f равна нулю, показывая что мы нашли желаемый алгоритм.

Другие отсчеты в частотной области вычисляются таким же образом. Эта процедура формализована в уравнении анализа, математическом способе вычисления частотной области по временной области:

$$\operatorname{Re} X[k] = \sum_{i=0}^{N-1} x[i] \cos\left(\frac{2\pi k i}{N}\right),$$
$$\operatorname{Im} X[k] = -\sum_{i=0}^{N-1} x[i] \sin\left(\frac{2\pi k i}{N}\right). \qquad (8.4)$$

Выражаясь словами, каждый отсчет в частотной области находится умножением сигнала временной области на искомую синусную или косинусную волну и суммированием результирующих точек. Если кто-нибудь спросит Вас, что Вы делаете, скажите с уверенностью: ”Я коррелирую входной сигнал с каждой базисной функцией”. В таблице 8.2 показана компьютерная программа для вычисления ДПФ таким способом.

Уравнение анализа *не* требует как уравнение синтеза специальной обработки первой и последней точек. Однако здесь есть знак минус перед мнимой частью в уравнении (8.4). Также как и раньше этот знак минус делает *действительное ДПФ* совместимым с *комплексным ДПФ*, и не всегда вводится.

Таблица 8.2

```
100 'THE DISCRETE FOURIER TRANSFORM
110 'The frequency domain signals, held in REX[ ] and IMX[ ], are calculated from
120 'the time domain signal, held in XX[ ].
130 '
140 DIM XX[511]              'XX[ ] holds the time domain signal
150 DIM REX[256]             'REX[ ] holds the real part of the frequency domain
160 DIM IMX[256]             'IMX[ ] holds the imaginary part of the frequency domain
170 '
180 PI = 3.14159265          'Set the constant, PI
190 N% = 512                 'N% is the number of points in XX[ ]
200 '
210 GOSUB XXXX               'Mythical subroutine to load data into XX[ ]
220 '
230 '
240 FOR K% = 0 TO 256      'Zero REX[ ] & IMX[ ] so they can be used as accumulators
250 REX[K%] = 0
260 IMX[K%] = 0
270 NEXT K%
280 '
290 '                        'Correlate XX[ ] with the cosine and sine waves, Eq. 8-4
300 '
310 FOR K% = 0 TO 256        'K% loops through each sample in REX[ ] and IMX[ ]
320 FOR I% = 0 TO 511        'I% loops through each sample in XX[ ]
330 '
340 REX[K%] = REX[K%] + XX[I%] * COS(2*PI*K%*I%/N%)
350 IMX[K%] = IMX[K%] - XX[I%] * SIN(2*PI*K%*I%/N%)
360 '
370 NEXT I%
```

```
380 NEXT K%
390 '
400 END
```

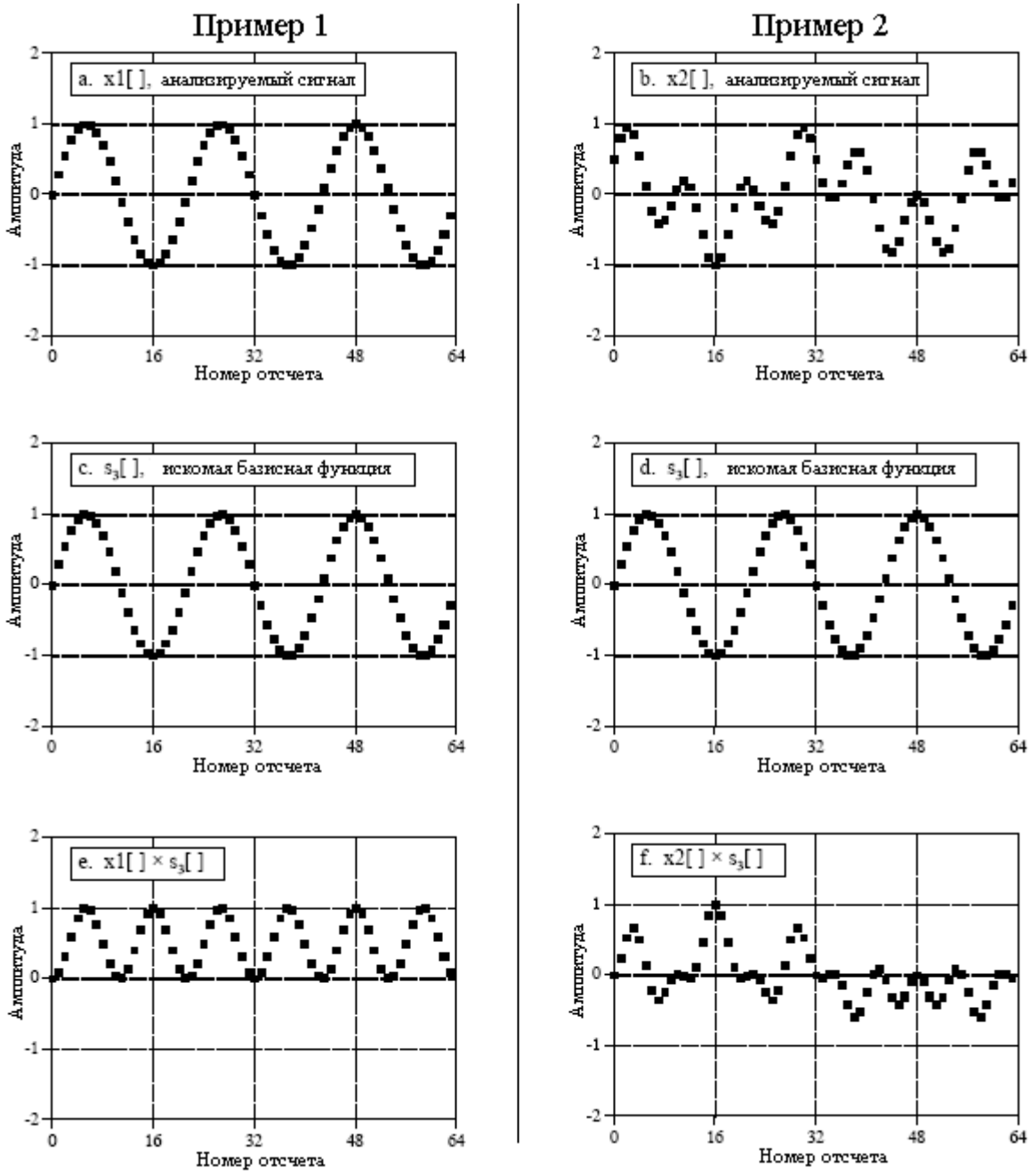


Рис. 8.8 Пример двух сигналов (a) и (b) анализируемых на содержание определенных базисных функций показанных на (c) и (d)

Для того чтобы этот корреляционный алгоритм работал, базисные функции должны иметь интересное свойство: каждая из них должна быть полностью *некоррелированной* со всеми другими. Это означает, что если Вы перемножите любые две базисные функции, сумма полученных в результате точек будет равна нулю. Базисные функции, обладающие таким свойством, называются **ортогональными**. Существует большое число других ортогональных базисных функций, включая: прямоугольные

волны, треугольные волны, импульсы и т.д. Используя корреляцию, сигналы могут быть разложены на эти другие ортогональные базисные функции точно так же, как это сделано здесь с синусоидами. Но это не предполагает, что это *полезно*, только потому, что это *возможно*.

Как было показано ранее в таблице 8.1, *обратное ДПФ* может быть реализовано в компьютерной программе двумя способами. Отличие включает в себя *взаимную замену* между внутренним и внешним циклами при выполнении синтеза. Хотя это не изменяет выходные данные программы, это вносит различие в то, как Вы *рассматриваете* то, что происходит. Программа *ДПФ* в таблице 8.2 тоже может быть изменена в такой же манере, взаимной заменой между внутренним и внешним циклами в строках с 310 по 380. Также как и раньше выходные данные программы будут такими же, но способ того, как Вы представляете вычисление, является другим. (Эти два разных подхода к представлению ДПФ и обратного ДПФ могли бы быть описаны как алгоритмы "входной стороны" и "выходной стороны", так же, как и для свертки.)

В том виде, в котором написана программа в таблице 8.2, она описывает то, как на индивидуальный отсчет в частотной области влияют все отсчеты во временной области. То есть, программа вычисляет каждое из значений в частотной области друг за другом, а не всю группу целиком. Когда внутренний и внешний циклы меняются местами программа проходит через каждый отсчет во временной области, вычисляя вклад этой точки в частотную область. Вся частотная область находится суммированием вкладов от индивидуальных точек временной области. Это вызывает наш следующий вопрос: какого вида вклад вносит отдельный отсчет временной области в частотную область? Ответ содержится в интересном аспекте анализа Фурье называемом *дуальностью*.

Дуальность

Уравнения синтеза и анализа (уравнения (8.2) и (8.4)) удивительно похожи. Для перехода из одной области в другую известные значения перемножаются с базисными функциями, а результаты произведений складываются. Тот факт, что *ДПФ* и *обратное ДПФ* используют один и тот же математический подход, учитывая, что мы пришли к этим двум процедурам вообще разными путями, в самом деле замечателен. Действительно, единственным значимым различием между двумя уравнениями является то, что результатом временной области является *один N* точечный сигнал, а частотной области – *два N/2+1* точечных сигнала. Как обсуждалось в предыдущих главах *комплексное ДПФ* представляет обе временную и частотную области как комплексные сигналы в *N* точек каждый. Это делает две области полностью симметричными, а уравнения для перехода между ними фактически *идентичными*.

Такая симметрия между временной и частотной областями называется **дуальностью** и дает очень много интересных свойств. Например, одна точка в частотной области соответствует синусоиде во временной области. В следствии дуальности справедливо также обратное, одна точка во временной области соответствует синусоиде в частотной области. В качестве другого примера, свертка во временной области соответствует умножению в частотной области. В следствии дуальности обратное также справедливо: свертка в частотной области соответствует умножению во временной области. Эти и другие соотношения дуальности обсуждаются более детально в Главах 10 и 11.

Полярные обозначения

Как до сих пор было описано, частотная область представляет собой группу амплитуд косинусных и синусных волн (с небольшими масштабными модификациями). Это называется **прямоугольной** системой обозначения. Альтернативно, частотная

область может быть выражена в **полярной** форме. В этой системе обозначений *Re X*[ ] и *Im X*[ ] заменяются двумя другими множествами называемыми **Модуль *X*[ ]**, в уравнениях записывается как ***Mag X*[ ]**, и **Фаза** *X*[ ], записывается как ***Phase X*[ ]**. Модуль и фаза являются заменой пары на пару для действительной и мнимой частей. Например, *Mag X*[*0*] и *Phase X*[*0*] вычисляются с использованием только *Re X*[*0*] и *Im X*[*0*]. Подобным образом *Mag X*[*14*] и *Phase X*[*14*] вычисляются с использованием только *Re X*[*14*] и *Im X*[*14*], и так далее. Для того чтобы понять преобразование, посмотрите что происходит тогда, когда Вы складываете косинусную и синусную волны одной и той же частоты. Результатом является косинусная волна той же частоты, но с новой амплитудой и новым фазовым сдвигом. В виде уравнения эти два представления соотносятся как:

$$A\cos(x) + B\sin(x) = M\cos(x+\theta). \qquad (8.5)$$

Важным является то, что в этом процессе никакая информация не теряется; задав одно представление, Вы можете вычислить другое. Другими словами, информация, содержащаяся в амплитудах *A* и *B*, также содержится в переменных *M* и *θ*. Хотя это уравнение включает в себя синусную и косинусную волны, оно подчиняется тем же уравнениям преобразования, что и простые векторы. Подобное представление вектора, где две переменные *A* и *B* могут рассматриваться в прямоугольной системе координат, в то время как *M* и *θ* являются параметрами в полярных координатах, показано на рис. 8.9.

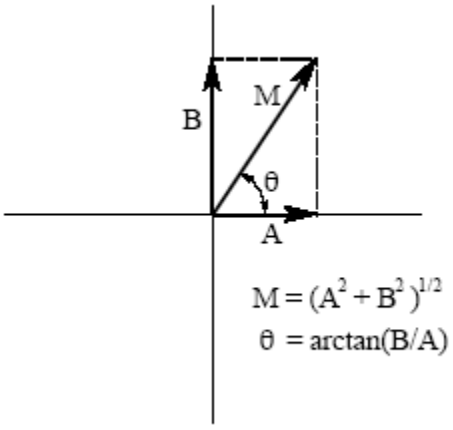


Рис. 8.9 Преобразование прямоугольной системы обозначений в полярную

В полярных обозначениях *Mag X*[ ] содержит амплитуду косинусной волны (*M* в уравнении (8.5) и на рис. 8.9), в то время как *Phase X*[ ] содержит фазовый угол косинусной волны (*θ* в уравнении (8.5) и на рис. 8.9). Следующие уравнения осуществляют преобразование частотной области из прямоугольных обозначений в полярные и наоборот.

$$MagX[k] = \left(\mathrm{Re}\,X[k]^2 + \mathrm{Im}\,X[k]^2\right)^{1/2}$$
$$PhaseX[k] = \arctan\left(\frac{\mathrm{Im}\,X[k]}{\mathrm{Re}\,X[k]}\right) \qquad (8.6)$$

$$\begin{aligned}\mathrm{Re}\,X[k] &= MagX[k]\cos(PhaseX[k]) \\ \mathrm{Im}\,X[k] &= MagX[k]\sin(PhaseX[k])\end{aligned} \tag{8.7}$$

Прямоугольное и полярное обозначения позволяют Вам рассматривать ДПФ двумя различными способами. В прямоугольных обозначениях ДПФ осуществляет декомпозицию *N* точечного сигнала на *N/2+1* косинусных волн и *N/2+1* синусных волн, каждая из которых имеет свою собственную *амплитуду*. В полярных обозначениях ДПФ осуществляет декомпозицию *N* точечного сигнала на *N/2+1* косинусных волн, каждая из которых имеет свою собственную *амплитуду* (называемую *модулем*) и *фазовый сдвиг*. Почему в полярных обозначениях используются косинусные волны, а не синусные? С помощью синусных волн невозможно представить постоянную составляющую сигнала, поскольку синусная волна нулевой частоты состоит из одних нулей (см. рис. 8.5 a и b).

Даже притом, что полярное и прямоугольное представления содержат точно ту же самую информацию, существует множество примеров, где одно представление легче использовать, чем другое. Например, на рис. 8.10 показана частотная область сигнала в обоих прямоугольном и полярном представлениях. Предупреждение: не пытайтесь понять форму кривых действительной и мнимой частей; Ваша голова может лопнуть! Для сравнения, полярные кривые прямонаправленные: присутствуют только частоты ниже приблизительно 0.25, а фазовый сдвиг приблизительно пропорционален частоте. Это частотный отклик низкочастотного фильтра.

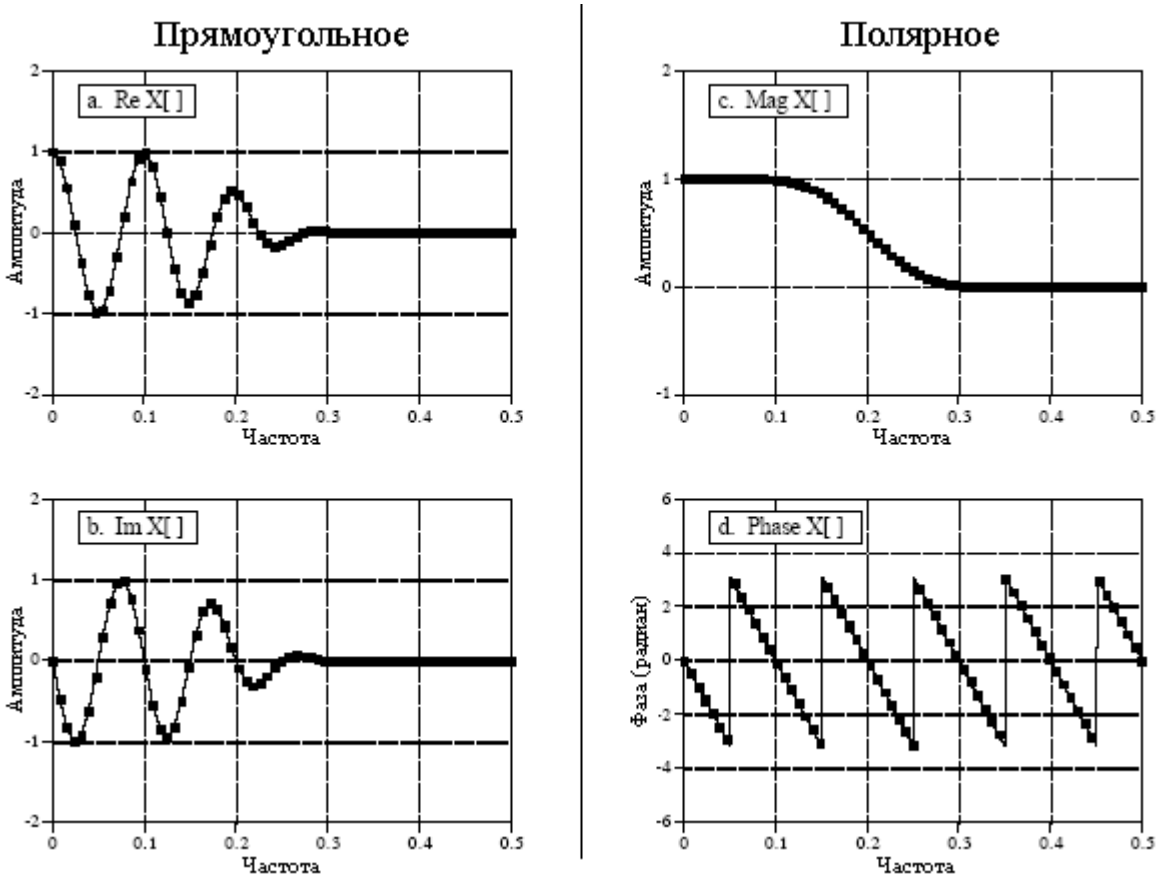


Рис. 8.10 Пример прямоугольной и полярной частотных областей

Когда же Вам следует использовать прямоугольные обозначения, и когда Вам следует использовать полярные? Лучшим выбором для вычислений, таких как в

уравнениях и компьютерных программах обычно являются прямоугольные обозначения. Для сравнения, графики почти всегда представляются в полярной форме. Как показано в предыдущем примере, рассматривая действительную и мнимую части, для *человека* почти невозможно понять характеристики сигнала в частотной области. В обычной программе сигналы частотной области хранятся в прямоугольных обозначениях до тех пор, пока наблюдателю не потребуется взглянуть на них, в это время и производится преобразование из прямоугольной системы в полярную.

Почему понимать частотную область в полярных обозначениях легче? Этот вопрос подводит нас к сердцу того, почему разложение сигнала на синусоиды является *полезным*. Вспомните свойство *синусоидального качества* из Главы 5: если синусоида подается на вход линейной системы, на выходе также будет синусоида и точно такой же частоты, как на входе. Могут измениться только амплитуда и фаза. Полярная система обозначений непосредственно представляет сигналы в терминах амплитуды и фазы составляющих косинусных волн. В свою очередь, системы могут быть представлены при помощи того, как они изменяют амплитуду и фазу каждой из этих косинусных волн.

А сейчас рассмотрим то, что случается, если с этим сценарием используется прямоугольная система обозначений. Смесь косинусных и синусных волн входящих в линейную систему, дает смесь косинусных и синусных волн, выходящих из системы. Проблема заключается в том, что косинусная волна на входе может дать обе косинусную и синусную волны на выходе. Аналогично, синусная волна на входе может дать обе косинусную и синусную волны на выходе. Хотя эти перекрестные члены могут быть упорядочены, весь метод в целом не гармонирует с тем, почему мы, прежде всего хотели использовать синусоиды.

Полярные неприятности

Существует масса неприятностей связанных с использованием полярной системы обозначений. Ни одна из них не является непреодолимой, но в действительности раздражает! В таблице 8.3 показана компьютерная программа для преобразования между прямоугольной и полярной системами обозначений, обеспечивающая разрешение некоторых из этих неприятностей.

Неприятность 1: радианы против градусов

Существует возможность выражать фазу либо в *градусах*, либо в *радианах*. При выражении в градусах значения фазы сигнала находятся между –180 и 180. При использовании радиан каждое из значений будет между $-\pi$ и π, то есть, между –3,141592 и 3,141592. Для вычисления тригонометрических функций, таких как косинус, синус, арктангенс и т.п., большинство языков программирования требуют использования радиан. Работа с этими длинными десятичными числами может вызывать раздражение и трудности при интерпретации полученных вами данных. Например, если вы хотите ввести в сигнал фазовый сдвиг в 90 градусов, Вы должны прибавить к фазе 1,570796. Хотя добавление этого к вашей программе и не убьет Вас, но это становится утомительным. Наилучшим способом разрешения этой проблемы является определение константы *PI=3.141592* в начале вашей программы. Тогда фазовый сдвиг в 90 градусов может быть записан как *PI/2*. В ЦОС широко используются и градусы и радианы и Вам нужно чувствовать себя уверенно при работе с обоими.

Таблица 8.3

```
100 'RECTANGULAR-TO-POLAR & POLAR-TO-RECTANGULAR CONVERSION
110 '
120 DIM REX[256]          'REX[ ] holds the real part
```

```
130 DIM IMX[256]          'IMX[ ] holds the imaginary part
140 DIM MAG[256]          'MAG[ ] holds the magnitude
150 DIM PHASE[256]        'PHASE[ ] holds the phase
160 '
170 PI = 3.14159265
180 '
190 GOSUB XXXX            'Mythical subroutine to load data into REX[ ] and IMX[ ]
200 '
210 '
220 '                      'Rectangular-to-polar conversion, Eq. 8-6
230 FOR K% = 0 TO 256
240 MAG[K%] = SQR( REX[K%]^2 + IMX[K%]^2 )   'from Eq. 8-6
250 IF REX[K%] = 0 THEN REX[K%] = 1E-20      'prevent divide by 0 (nuisance 2)
260 PHASE[K%] = ATN( IMX[K%] / REX[K%] )     'from Eq. 8-6
270 ' 'correct the arctan (nuisance 3)
280 IF REX[K%] < 0 AND IMX[K%] < 0 THEN PHASE[K%] = PHASE[K%] - PI
290 IF REX[K%] < 0 AND IMX[K%] >= 0 THEN PHASE[K%] = PHASE[K%] + PI
300 NEXT K%
310 '
320 '
330 '                      'Polar-to-rectangular conversion, Eq. 8-7
340 FOR K% = 0 TO 256
350 REX[K%] = MAG[K%] * COS( PHASE[K%] )
360 IMX[K%] = MAG[K%] * SIN( PHASE[K%] )
370 NEXT K%
380 '
390 END
```

Неприятность 2: ошибка деления на ноль

При преобразовании из прямоугольной системы обозначений в полярную, очень часто встречаются частоты, для которых действительная часть равна нулю, а мнимая часть является некоторой не нулевой величиной. Это просто означает, что фаза точно равна 90 или –90 градусов. Попытайтесь объяснить это вашему компьютеру! Когда ваша программа пытается вычислить фазу из выражения: *Phase X*[*k*] = *arctan* (*Im X*[*k*] / *Re X*[*k*]), происходит *ошибка деления на ноль*. Если даже выполнение программы не остановится фаза, которую Вы получите для этой частоты, не будет верной. Для того чтобы избежать этой проблемы, действительная часть до деления должна быть проверена на равенство нулю. Если она равна нулю, мнимая часть должна быть проверена на соответствие положительной или отрицательной величине для определения, устанавливать ли фазу соответственно на $\pi/2$ или $-\pi/2$. Наконец, должно быть пропущено деление. Во всех этих шагах нет ничего трудного, это просто потенциал для раздражения. Альтернативный способ разрешения этой проблемы показан в строке 250 таблицы 8.3. Если действительная часть равна нулю, измените ее до пренебрежимо малого числа, чтобы осчастливить математический сопроцессор во время деления.

Неприятность 3: неправильный арктангенс

Рассмотрим отсчет частотной области, в котором *Re X*[*k*]=1 и *Im X*[*k*]=1. Уравнение (8.6) дает соответствующие полярные величины *Mag X*[*k*]=1,414 и *Phase X*[*k*]=45^o. Теперь рассмотрим другой отсчет, в котором *Re X*[*k*]=-1 и *Im X*[*k*]=-1. Уравнение (8.6) снова дает соответствующие полярные величины *Mag X*[*k*]=1,414 и *Phase X*[*k*]=45^o. Проблема заключается в том, что фаза является неверной! Она должна быть равна –135^o. Такая

ошибка случается тогда, когда действительная часть отрицательна. Эта проблема может быть скорректирована проверкой действительной и мнимой частей, после того как была вычислена фаза. Если обе действительная и мнимая части отрицательны, отнимите от вычисленной фазы 180^o (или π радиан). Если действительная часть отрицательна, а мнимая часть положительна, прибавьте 180^o (или π радиан). Строки 340 и 350 программы в таблице 8.3 показывают, как это сделать. Если Вам не удастся решить эту проблему, то вычисленное значение фазы будет изменяться только между $-\pi/2$ и $\pi/2$, а не между $-\pi$ и π. Зарубите это себе на носу. Если Вы видите фазу изменяющейся только между $\pm 1,5708$, Вы забыли скорректировать неоднозначность при вычислении арктангенса.

Неприятность 4: фазы очень маленьких модулей

Представьте следующий сценарий. Вы усердно работаете над некоторой задачей ЦОС и вдруг замечаете, что часть фазы выглядит неправильно. Она может быть зашумлена, повсюду прыгает, или просто обыкновенно *ошибочна*. После того как Вы потратите следующий час на просмотр сотен строк компьютерных кодов, Вы находите ответ. Соответствующие величины модулей настолько малы, что они погребены в шуме округления. Если модуль пренебрежимо мал, фаза не имеет никакого смысла и может принимать необычные значения. Пример этого показан на рис. 8.11. Обычно очевидно, что когда сигнал с некоторой *амплитудой* теряется в шуме, его значения настолько малы, что Вы вынуждены признать, что эти значения являются бессмысленными. С фазой другое дело. Когда полярный сигнал загрязнен шумом, значения фазы являются случайными числами между $-\pi$ и π. К сожалению, часто это *выглядит* как реальный сигнал, а не чепуха, как это на самом деле.

Неприятность 5: 2π неоднозначность фазы

Посмотрите снова на рис. 8.10d и обратите внимание на несколько разрывов в данных. Каждый раз, когда точка выглядит так, как будто она собирается нырнуть ниже –3,14592, она мгновенно отскакивает назад к 3,141592. Это результат периодического характера синусоид. Например, фазовый сдвиг θ точно такой же, как и фазовый сдвиг $\theta+2\pi$, $\theta+4\pi$, $\theta+6\pi$ и т.д. Любая синусоида не изменяется, когда Вы добавляете к фазе целое, умноженное на 2π. Очевидные разрывы в сигнале это результат компьютерного алгоритма, делающего свой излюбленный выбор из бесконечного числа эквивалентных возможностей. Всегда выбирается самое маленькое возможное значение, удерживающее фазу между $-\pi$ и π.

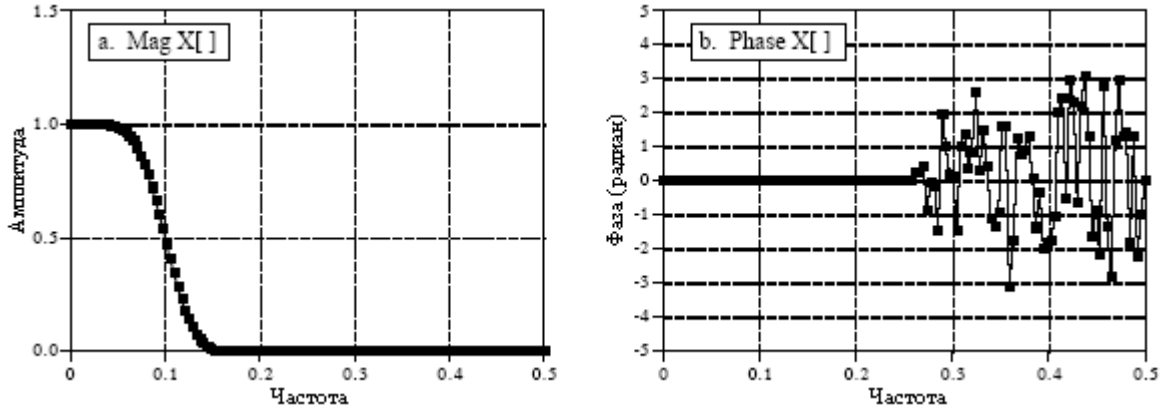


Рис. 8.11 Фаза сигналов с малыми модулями

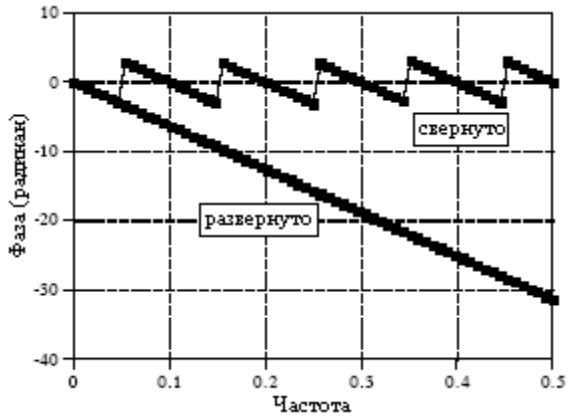


Рис. 8.12 Пример развертывания фазы

Часто фазу легче понять, если она не имеет этих разрывов, даже если это означает, что фаза простирается выше π или ниже $-\pi$. Это называется **развертывание фазы**, а на рис 8.12 показан пример этого. Как показано в программе из таблицы 8.4, к значению фазы прибавляется или отнимается число кратное 2π. Точное значение определяется при помощи алгоритма, который минимизирует разность между смежными отсчетами.

Таблица 8.4

```
100 ' PHASE UNWRAPPING
110 '
120 DIM PHASE[256]                  'PHASE[ ] holds the original phase
130 DIM UWPHASE[256]                'UWPHASE[ ] holds the unwrapped phase
140 '
150 PI = 3.14159265
160 '
170 GOSUB XXXX                      'Mythical subroutine to load data into PHASE[ ]
180 '
190 UWPHASE[0] = 0                  'The first point of all phase signals is zero
200 '
210 '                               'Go through the unwrapping algorithm
220 FOR K% = 1 TO 256
230 C% = CINT( (UWPHASE[K%-1] - PHASE[K%]) / (2 * PI) )
240 UWPHASE[K%] = PHASE[K%] + C%*2*PI
250 NEXT K%
260 '
270 END
```

Неприятность 6: модуль всегда положителен (π неоднозначность фазы)

На рис. 8.13 показан сигнал частотной области в прямоугольной и полярной форме. Действительная часть плавная, и ее легко понять, в то время как мнимая часть всецело равна нулю. Для сравнения, полярный сигнал содержит резкие разрывы и острые углы. Это происходит потому, что модуль *по определению* должен всегда быть положительным. Всякий раз когда действительная часть ныряет ниже нуля, модуль остается положительным за счет изменения фазы на π (или $-\pi$, что тоже самое). Хотя для математиков это и не является проблемой, нерегулярные кривые при интерпретации могут представлять трудность.

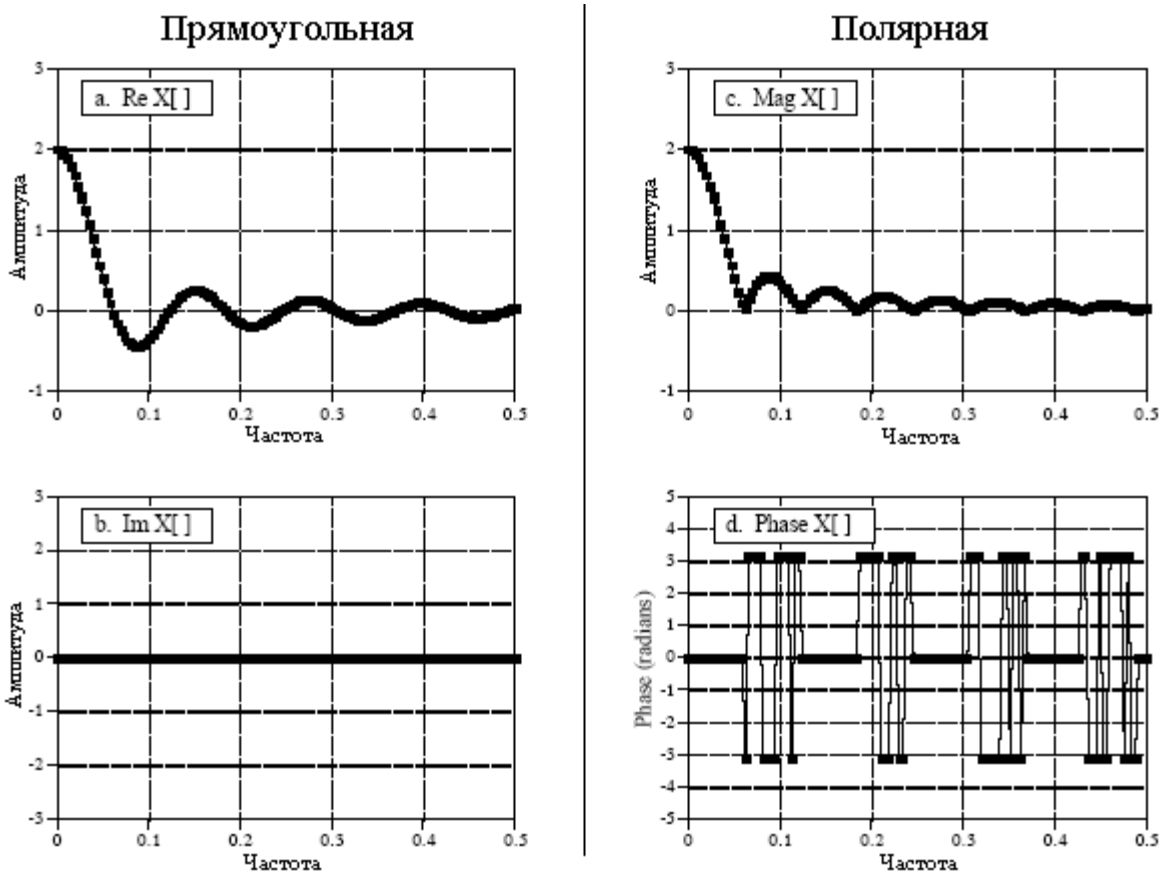


Рис. 8.13 Пример сигналов в прямоугольной и полярной формах

Одно из решений состоит в том, чтобы позволить модулю иметь *отрицательные* значения. В примере на рис. 8.13 это заставило бы модуль выглядеть таким же, как и действительная часть, в то время как фаза была бы всецело равна нулю. В этом нет ничего плохого, если это помогает вашему пониманию. Просто будьте тогда аккуратны, не называйте сигнал с отрицательными значениями “модулем”, поскольку это нарушает его формальное определение. В данной книге мы используем мягкие слова: *развернутый модуль* для обозначения “модуля”, позволяющего иметь отрицательные значения.

Неприятность 7: пики между π и -π

Поскольку π и $-\pi$ представляют один и тот же фазовый сдвиг, шум округления может стать причиной быстрого переключения фазы соседних точек между двумя значениями. Как показано на рис. 8.13d, это может вызвать резкие разрывы и пики на, при других условиях, гладкой кривой. Не дайте себя обмануть, в действительности фаза не имеет этих разрывов.

ГЛАВА
9

ПРИМЕНЕНИЯ ДПФ

Дискретное преобразование Фурье (ДПФ) является одним из наиболее важных инструментов в цифровой обработке сигналов. В этой главе обсуждаются три общих способа его применения. Во-первых, ДПФ может вычислить *частотный спектр* сигнала. Это прямое исследование информации закодированной в частоте, фазе и амплитуде составляющих сигнал синусоид. Например, речь и слух человека используют сигналы с кодированием такого типа. Во-вторых, ДПФ может находить частотный отклик системы по её импульсному отклику и наоборот. Это позволяет анализировать системы в *частотной области* так же, как свертка позволяет анализировать системы во *временной области*. В-третьих, ДПФ можно использовать как промежуточный этап в более сложных методах обработки сигнала. Классическим примером этого является *БПФ свертка*, алгоритм, осуществляющий свертку сигналов в сотни раз быстрее, чем традиционные методы.

Спектральный анализ сигналов

Очень часто информация закодирована в синусоидах, которые собственно формируют сам сигнал. Это присуще как естественно возникающим сигналам, так и тем, что создаются человеком. В нашей вселенной большое число вещей генерирует колебания. Например, результатом вибрации голосовых связок является речь человека; звезды и планеты изменяют свою яркость по мере вращения их вокруг своих осей и поворота относительно друг друга; винты кораблей производят периодическое перемещение воды и т.д. В этих сигналах *очертание* формы волны временной области не важно; ключевая информация содержится здесь в *частоте*, *фазе* и *амплитуде* составляющих сигнал синусоид. Для извлечения этой информации применяется ДПФ.

Как это все работает, покажет следующий пример. Предположим, что мы хотим исследовать распространяющиеся в океане звуки. Для начала в воду помещается микрофон и результирующий электронный сигнал усиливается до приемлемого уровня, скажем в несколько вольт. Затем, для того чтобы сигнал можно было оцифровать при скорости 160 отсчетов в секунду, используется аналоговый низкочастотный фильтр для удаления всех частот свыше 80 герц. А что дальше, после получения и записи нескольких тысяч отсчетов?

Прежде всего, нужно просто *посмотреть* на данные. На рис. 9.1a показаны 256 отсчетов из нашего воображаемого эксперимента. Все что можно увидеть - это зашумленная форма волны, которая для глаза человека дает мало информации. По причинам, которые мы объясним вскоре, следующим шагом является умножение этого сигнала на гладкую кривую называемую **окном Хемминга**, показанную на рис. 9.1b. (В Главе 16 приводятся уравнения для окна Хемминга и других окон; см. уравнения (16.1), (16.2) и рис. 16.2a). Результатом этого является сигнал из 256 точек, в котором отсчеты возле концов были уменьшены по амплитуде так, как это показано на рис. 9.1c.

Взятие ДПФ и приведение к полярной системе обозначений в результате дает 129 точечный частотный спектр, показанный на рис. 9.1d. К сожалению, он тоже напоминает

беспорядочный шум. Это связано с тем, что в исходных 256 точках нет достаточной информации для получения хорошо ведущей себя кривой. Использование более длинного ДПФ в решении этой проблемы, ничего не дает. Например, если используется 2048 точечное ДПФ, частотный спектр становится 1025 отсчетов в длину. Даже если исходные 2048 точек содержат больше информации, большее число отсчетов в спектре разбавляет информацию в то же самое число раз. Более длинное ДПФ дает лучшее частотное разрешение, но тот же самый уровень шума.

Ответ лежит в использовании как можно большей части исходного сигнала таким способом, который не увеличивает число точек в частотном спектре. Это может быть сделано за счет разбиения входного сигнала на большое число 256 точечных *сегментов*. Каждый из этих сегментов умножается на окно Хемминга, пропускается через 256 точечное ДПФ и преобразуется к полярным обозначениям. Затем результирующие частотные спектры *усредняются* для формирования единственного 129 точечного частотного спектра. На рис. 9.1e показан пример усреднения 100 частотных спектров подобных спектру на рис. 9.1d. Улучшение очевидно; шум был уменьшен до уровня, позволяющего наблюдать интересные характеристики сигнала. В такой манере усредняется только *модуль* частотной области, *фаза* обычно отбрасывается, поскольку она не содержит полезной информации. Случайный шум уменьшается пропорционально *корню квадратному* из количества сегментов. Хотя типичной величиной является 100 сегментов, некоторые приложения для выявления слабых характеристик могут усреднять *миллионы* сегментов.

Существует также второй метод снижения спектрального шума. Начните с взятия очень длинного ДПФ, скажем 16384 точечного. Результирующий частотный спектр обладает высоким разрешением (8193 отсчета), но очень зашумлен. Затем, для *сглаживания* спектра используется цифровой низкочастотный фильтр, снижающий шум за счет разрешения. Например, для получения каждого отсчета в отфильтрованном спектре, самый простой цифровой фильтр мог бы усреднить 64 смежных отсчета в исходном спектре. Проделав вычисления, получим почти такой же шум и разрешение как в первом методе, где 16384 точки были разбиты на 64 сегмента по 256 точек в каждом.

Какой же метод Вам следует использовать? Первый метод проще, потому что не нужен цифровой фильтр. Второй метод имеет *потенциал* лучшего функционирования, поскольку цифровой фильтр может быть спроектирован так, чтобы оптимизировать соотношение между шумом и разрешением. Однако это улучшенное функционирование такого беспокойства не стоит. Это связано с тем, что оба и уровень шума и разрешение могут быть улучшены за счет использования *большего количества данных* из входного сигнала. Например, представьте, что полученные данные разбиваются на 10000 сегментов по 16384 отсчета каждый. Такой результирующий частотный спектр обладает высоким разрешением (8193 точки) и низким шумом (10000 усреднений). Проблема решена! По этой причине в этой дискуссии мы будем рассматривать только метод усредненного сегмента.

На рис. 9.2 показан спектр примера с нашим подводным микрофоном, иллюстрирующий характеристики, которые обычно появляются в частотных спектрах принятых сигналов. Игнорируйте на некоторое время острые пики. Между 10 и 70 герцами сигнал состоит из относительно плоского участка. Он называется **белым шумом**, поскольку содержит равное количество абсолютно всех частот, точно так же, как и белый свет. Это следствие *некоррелированного* от отсчета к отсчету шума формы волны временной области. То есть, знание величины шума присутствующего в любом единичном отсчете не дает никакой информации о величине шума присутствующего в любом другом отсчете. Белый шум создается, например, хаотичным движением электронов в электронных схемах. В качестве более знакомого примера, белым шумом является звук брызг воды, падающей на пол в душе. Белый шум, показанный на рис. 9.2,

мог бы исходить от любых нескольких источников, включая аналоговую электронику, или сам океан.

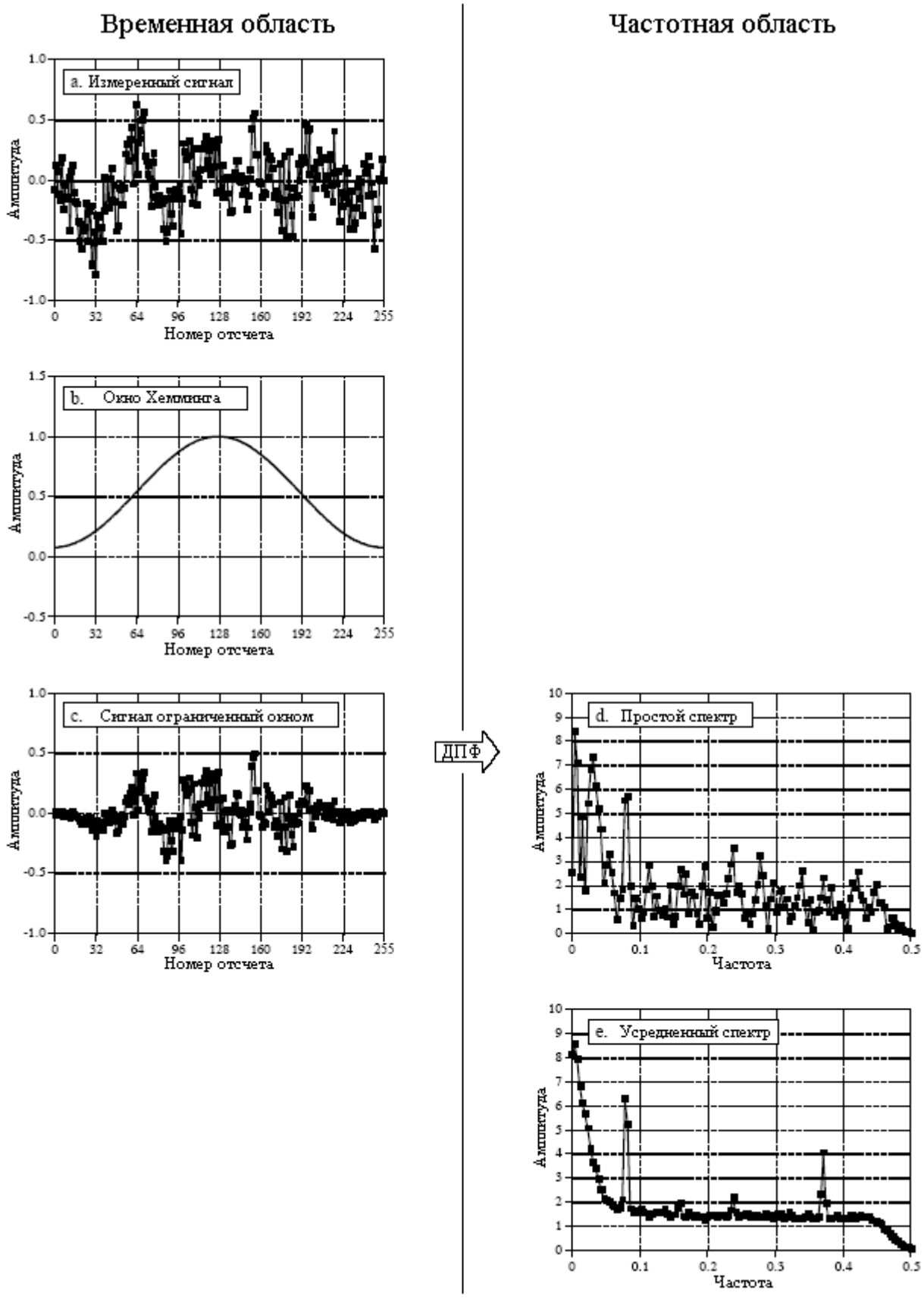


Рис. 9.1 Пример спектрального анализа

Выше 70 герц белый шум быстро уменьшается по амплитуде. Это результат завала фильтра антисовмещения. Идеальный фильтр пропустил бы все частоты ниже 80 герц и заградил бы все частоты выше. На практике идеально резкий срез невозможен, и Вы должны ожидать, что увидите такое постепенное снижение. Если его нет, то следует полагать, что существует проблема совмещения имен.

Ниже приблизительно 10 герц шум быстро нарастает из-за необычности, называемой **шум 1/f** (шум единица на f) (фликкер-шум – прим. перев.). Шум 1/f это загадка. Он был измерен в системах совершенно разного типа, таких как плотность движения на автострадах и электронный шум в транзисторах. Вероятно, если заглянуть в достаточно низкую область частот, он мог бы быть измерен во всех системах. Несмотря на то, что он часто встречается, общая теория и понимание шума 1/f от исследователей ускользнули. В некоторых специфических системах причина этого шума может быть определена, однако это не дает ответа на вопрос, почему шум 1/f присутствует везде. Для обычной аналоговой электроники и большинства физических систем переход между белым шумом и шумом 1/f происходит приблизительно между 1 и 100 герцами.

Ну а сейчас мы переходим к острым пикам на рис. 9.2. Наиболее простым в объяснении является пик 60 герц, результат электромагнитных наводок от коммерческой электрической сети (частота промышленной электрической сети в США составляет 60 герц, отечественная и европейская промышленная электрическая сеть работает на частоте 50 герц – прим. перев.). Поскольку форма волны линии электропередачи не является *идеальной* синусоидой, ожидайте также увидеть меньшие пики при значениях кратных этой частоте (120, 180, 240 герц, и т.д.). Обычным также является обнаружение пиков наводок между 25-40 кГц частотами, пользующимися популярностью у разработчиков импульсных источников электропитания. Близко расположенные радио и телевизионные станции дают пики наводок в мегагерцовом диапазоне. Низкие частотные пики могут быть вызваны вибрирующими при тряске компонентами в системе. Это называется *микрофонным эффектом* и обычно создает пики в полосе от 10 до 100 герц.

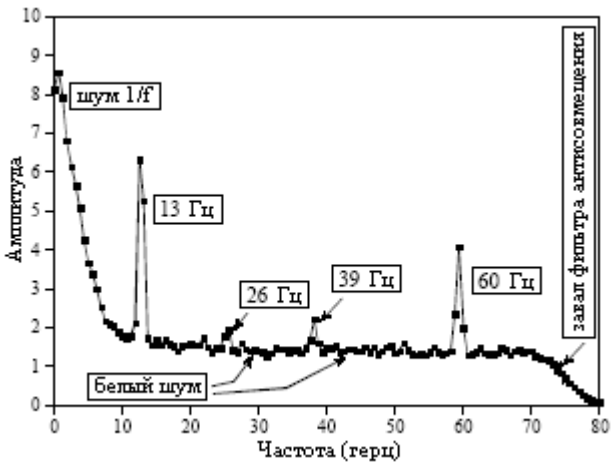


Рис. 9.2 Пример частотного спектра

Теперь мы переходим к настоящим сигналам. Существует сильный пик при 13 герцах с более слабыми пиками при 26 и 39 герцах. Как будет обсуждено в следующей главе, это частотный спектр несинусоидальной волны периодической формы. Пик при 13 герцах называется основной частотой, в то время как пики при 26 и 39 герцах называются второй и третьей гармоникой, соответственно. Вам также следует ждать обнаружения пиков и на других кратных 13 герц частотах, таких как 52, 65, 78 герц и т.д. Вы не видите их на рис. 9.2, потому что они спрятаны в белом шуме. Данный сигнал в 13 герц мог бы быть произведен, например, трех лопастным винтом подводной лодки, вращающимся с

частотой 4,33 оборота в секунду. Опознавание подводных звуков по их частотам и гармоническим составляющим, это основы *пассивной* гидролокации.

Предположим, что есть пики очень близко расположенные друг к другу, так как показано на рис. 9.3. Существуют два фактора ограничивающих разрешение, которое может быть получено по частоте, то есть то, насколько близки могут быть пики без того, чтобы слиться в одно целое. Первым фактором является длина ДПФ. Частотный спектр, получаемый с помощью *N* точечного ДПФ, состоит из *N/2+1* отсчетов равномерно расположенных между нулем и половиной частоты дискретизации. Для разделения двух близко расположенных частот, расстояние между отсчетами должно быть *меньше*, чем дистанция между двумя пиками. Например, для того чтобы разделить пики на рис. 9.3, достаточно 512 точечного ДПФ, в то время как 128 точечного ДПФ недостаточно.

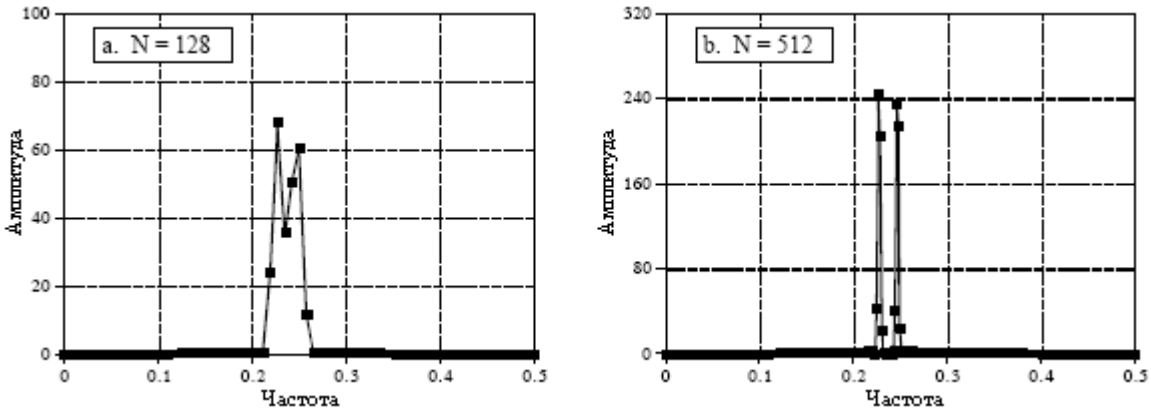


Рис. 9.3 Разрешение частотного спектра

Второй фактор, ограничивающий разрешение, более сложен. Представьте сигнал, созданный суммированием двух синусоидальных волн отличающихся исключительно небольшой разницей их частот. Форма волны этого сигнала будет выглядеть как *простая* синусоидальная волна лишь на коротком сегменте, скажем, в несколько периодов. Чем ближе друг к другу частоты, тем длиннее должен быть сегмент, для того чтобы сделать заключение, что в сигнале присутствует более чем одна частота. Другими словами, *длина* сигнала ограничивает разрешение по частоте. Это непохоже на первый фактор потому, что здесь *длина входного сигнала* может быть не такой, как *длина ДПФ*. Например, 256 точечный сигнал может быть дополнен нулями для того, чтобы сделать его 2048 точек длиной. Взятие 2048 точечного ДПФ, дает частотный спектр с 1025 отсчетами. Добавленные нули не изменяют очертание спектра, они только дают большее количество отсчетов в частотной области. Несмотря на такое очень плотное расположение отсчетов, способность разделить близко расположенные пики будет лишь чуть-чуть лучше, чем при 256 точечном ДПФ. В случаях, когда ДПФ имеет ту же самую длину что и входной сигнал, разрешение ограничивается двумя этими факторами приблизительно одинаково. Вскоре мы вернемся к этой проблеме.

Следующий вопрос: Что происходит, если входной сигнал содержит синусоиду с частотой *между* двумя базисными функциями? Ответ показан на рис. 9.4. Это частотный спектр сигнала, состоящего из двух синусоидальных волн, причем одна имеет частоту *соответствующую* базисной функции, а другая частоту, лежащую *между* двумя базисными функциями. Как Вы должны догадываться, первая синусоидальная волна представлена одной точкой. Другой пик более труден для понимания. Поскольку он не может быть представлен одним отсчетом, он становится пиком с **хвостами**, простирающимися на значительное расстояние от него.

Решение? Перед тем как взять ДПФ, как описывалось ранее, умножьте сигнал на окно Хемминга. На рис. 9.4b показано, что за счет использования окна спектр изменился в

трех направлениях. Во-первых, два пика стали выглядеть более одинаковыми. Это хорошо. Во-вторых, сильно сократились хвосты. Это тоже хорошо. В-третьих, окно снизило разрешение в спектре, сделав пики более широкими. Это плохо. На жаргоне ЦОС окна обеспечивают компромисс между *разрешением* (ширина пика) и *растеканием спектра* (величина хвостов).

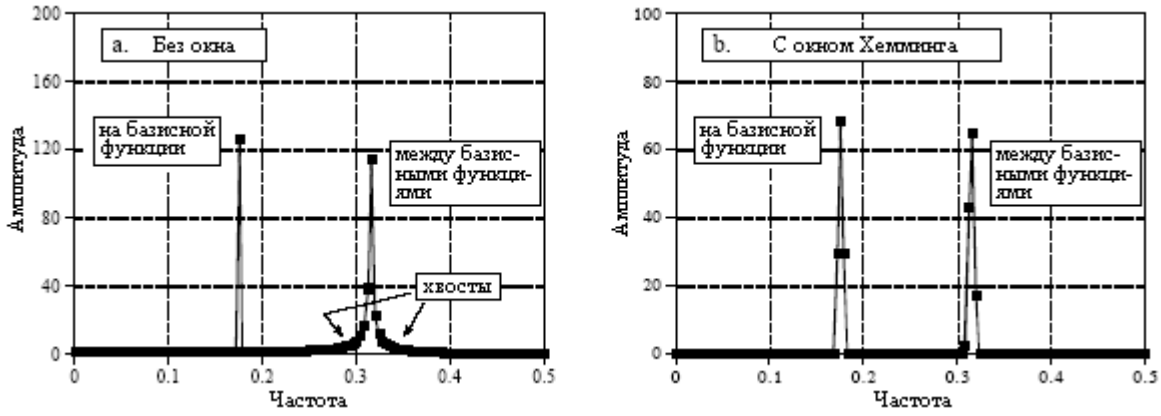


Рис. 9.4 Пример использования окна в спектральном анализе

Чтобы исследовать теоретические аспекты этого более подробно, вообразите бесконечно длинную дискретную синусоидальную волну с частотой дискретизации 0,1. Частотный спектр этого сигнала представляет собой бесконечно малый узкий пик, при равенстве нулю на всех других частотах. Конечно, ни этот сигнал, ни его частотный спектр не могут быть введены в цифровой компьютер, из-за их, соответственно, бесконечной и бесконечно малой природы. Для того чтобы это обойти, мы изменяем сигнал двумя способами, оба из которых искажают истинный частотный спектр.

Во-первых, мы проводим *ограничение* информации в сигнале за счет умножения его на окно. Например, 256 точечное *прямоугольное окно* позволило бы 256 точкам сохранить их правильное значение, в то время как все другие отсчеты в бесконечно длинном сигнале будут установлены в нулевое значение. Аналогично, окно Хемминга *придаст конфигурацию* попавшим в него отсчетам и, кроме того, установит все точки за пределами окна в ноль. Сигнал останется бесконечно длинным, но ненулевое значение будет иметь только конечное число отсчетов.

Как это ограничение с помощью окна влияет на частотную область? Как будет обсуждено в Главе 10, когда *перемножаются* два сигнала временной области, происходит *свертка* соответствующих частотных областей. Поскольку исходный спектр представляет собой бесконечно малый узкий пик (т.е. дельта функцию), спектр ограниченного окном сигнала представляет собой спектр окна, сдвинутый к местоположению пика. На рис. 9.5 показано как выглядел бы спектральный пик при использовании четырех различных видов окна (Если Вам нужна справка по dB, то загляните вперед в Главу 14). Кривая на рис. 9.5a - результат использования прямоугольного окна. Рисунки 9.5b и 9.5c получились от использования двух популярных окон, Хемминга и Блэкмена (как упоминалось ранее, для получения информации об этих окнах, см. уравнения (16.1), (16.2) и рис. 16.2a).

Как показано на рис. 9.5, все эти окна ухудшают исходный спектр, расширяя пик и добавляя хвосты, составленные из многочисленных боковых лепестков. Это неизбежный результат использования только части исходного сигнала временной области. Здесь мы можем наблюдать компромисс между тремя окнами. Окно Блэкмена имеет самый широкий основной лепесток (плохо), но самую низкую амплитуду хвостов (хорошо). Прямоугольное окно имеет самый узкий основной лепесток (хорошо), но самые большие хвосты (плохо). Окно Хемминга находится между двумя этими окнами.

Обратите внимание, что на рис. 9.5 частотные спектры являются непрерывными кривыми, а не дискретными отсчетами. После ограничения окном, сигнал временной области остается бесконечно длинным, даже притом, что большинство отсчетов равно нулю. Это означает, что частотный спектр состоит из *∞/2+1* отсчетов между *0* и *0,5*, это то же самое, что непрерывная линия.

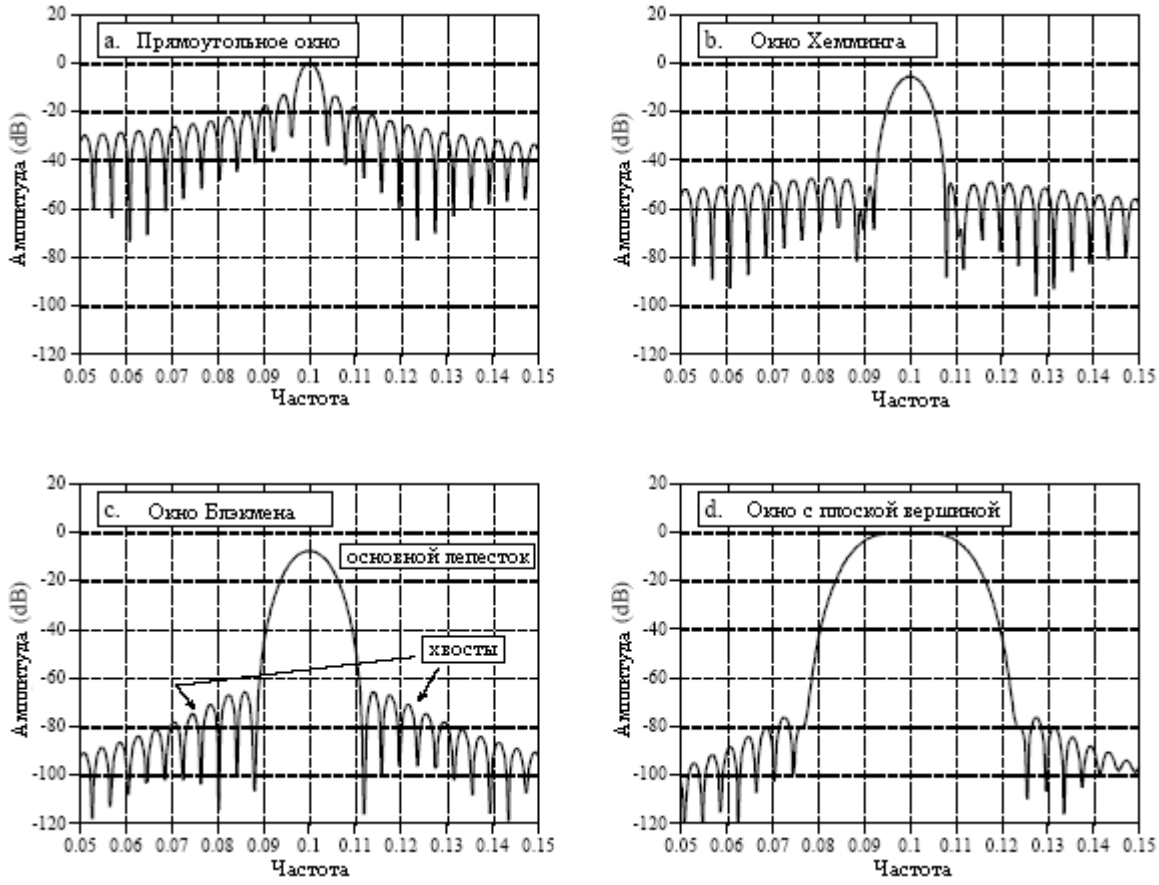


Рис.9.5 Детальный взгляд на спектральный пик при использовании различных окон

Данное приводит ко второму способу, при помощи которого нам нужно модифицировать сигнал временной области так, чтобы он мог быть представлен в компьютере: *выберите из сигнала N точек*. Эти *N* точек должны содержать все ненулевые точки, определенные окном, но они могут также включать и некоторое количество нулей. Это дает эффект осуществления *дискретизации* непрерывной кривой частотного спектра. Например, если *N* выбрано так, что оно равно 1024, то непрерывная кривая спектра будет дискретизирована 513 раз между 0 и 0,5. Если *N* выбрано так, что оно намного больше, чем длина окна, отсчеты в частотной области будут достаточно близко, так что все пики и впадины непрерывной кривой будут сохранены в новом спектре. Если *N* сделать таким же, как длина окна, то меньшее количество отсчетов в спектре приведет к тому, что регулярный рисунок пиков и впадин, в зависимости от того, куда довелось попасть отсчетам, превращается в хвосты неправильной формы. Это объясняет, почему два пика на рис. 9.4а не похожи друг на друга. Каждый пик на рис. 9.4а является *дискретизацией* основной кривой на рис. 9.5а. Наличие или отсутствие хвостов зависит от того, где располагаются отсчеты по отношению к пикам и впадинам. Если синусоидальная волна точно соответствует базисной функции, отсчеты появляются точно во впадинах, устраняя хвосты. Если синусоидальная волна находится между двумя базисными функциями, отсчеты появляются в каком-нибудь месте вдоль пиков и впадин, давая хвосты разнообразного вида.

Это подводит нас к **окну с плоской вершиной**, показанному на рис. 9.5d. В некоторых приложениях *амплитуда* спектрального пика должна быть измерена очень точно. Поскольку спектр ДПФ формируется из отсчетов, ни что не может гарантировать, что отсчет пришелся точно на вершину пика. Наиболее вероятно, что ближайший к вершине пика отсчет будет слегка смещен от центра, давая значение меньшее, чем действительная амплитуда. Решение состоит в том, чтобы использовать окно, которое дает спектральный пик с *плоской вершиной*, обеспечивая то, что всегда один или более отсчетов будут иметь правильное значение пика. Как показано в рис. 9.5d, наказанием за это является очень широкий основной лепесток, дающий плохое разрешение по частоте.

Как оказывается форма кривой, которую мы хотим найти для получения окна с плоской вершиной точно такая же, как форма ядра фильтра нижних частот. Теоретические причины этого мы будем обсуждать в более поздних главах; а сейчас, вот описание рецепта как использовать эту технику. В Главе 16 обсуждается низкочастотный фильтр называемый *фильтром с ограниченной окном Синк функцией*. Уравнение (16.4) описывает, как получить ядро фильтра (которое мы хотим использовать как окно), а на рис. 16.4a показана типичная форма кривой. Чтобы использовать это уравнение, Вам нужно знать значение двух параметров: M и f_c. Они находятся из соотношений: $M=N-2$, и $f_c=s/N$, где N представляет собой длину используемого ДПФ, а s представляет собой желаемое Вами число отсчетов, попадающих на плоскую часть пика (обычно между 3 и 5). В таблице 16.1 показана программа для вычисления ядра фильтра (наше окно), содержащая две тонкие особенности: нормализационную константу K, и как избежать ошибки деления на ноль на центральном отсчете. При использовании этого метода помните, что значение постоянной составляющей, равное *единице* во временной области, даст пик с амплитудой равной *единице* в частотной области. Однако синусоида, с амплитудой равной *единице* во временной области, даст спектральный пик с амплитудой равной только *одной второй*. (Это обсуждается в предыдущей главе: *Синтез*, *вычисление обратного ДПФ*.)

Частотный отклик систем

Во *временной области* системы анализируются при помощи свертки. Подобный анализ может быть выполнен и в *частотной области*. С помощью преобразования Фурье каждый входной сигнал может быть представлен как группа косинусоидальных волн, каждая из которых имеет свою специфическую амплитуду и фазовый сдвиг. Аналогично, для представления каждого выходного сигнала в такой же форме можно использовать ДПФ. Это означает, что любую линейную систему можно *полностью* описать тем, как она изменяет амплитуду и фазу проходящих через нее косинусных волн. Такая информация называется **частотным откликом** системы (частотной характеристикой системы – прим. перев.). Поскольку оба и импульсный отклик (импульсная характеристика – прим. перев.), и частотный отклик содержат полную информацию о системе, они должны соответствовать друг другу. Задавшись одним из них, Вы можете вычислить другой. Соотношение между импульсным откликом и частотным откликом является одним из начал обработки сигналов: *Частотный отклик системы это преобразование Фурье ее импульсного отклика*. Рис. 9.6 иллюстрирует это соотношение.

Придерживаясь стандартной системы обозначений ЦОС, при обозначении импульсных откликов используются строчные переменные, в тот время как при обозначении соответствующих частотных откликов используют заглавные. Поскольку $h[\ ]$ - это типичный символ для обозначения импульсного отклика, для обозначения частотного отклика используется $H[\ ]$. Во временной области системы описываются при помощи свертки, то есть: $x[n]*h[n]=y[n]$. В частотной области входной спектр *умножается* на частотный отклик, давая в результате выходной спектр. В виде уравнения

это выглядит так: $X[f] \times H[f] = Y[f]$. То есть *свертка* во временной области соответствует *умножению* в частотной области.

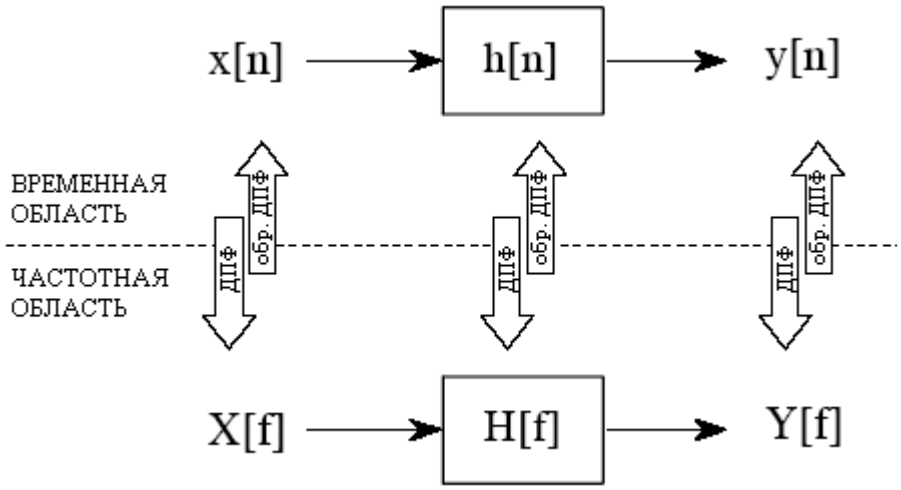


Рис. 9.6 Сравнение действия системы во временной и в частотной областях

На рис. 9.7 показан пример использования ДПФ для преобразования импульсного отклика системы в ее частотный отклик. На рис. 9.7a показан импульсный отклик системы. Просмотр данной кривой не дает Вам даже намека на то, что делает эта система. Взятие от этого импульсного отклика 64 точечного ДПФ дает частотный отклик системы, показанный на рис. 9.7b. Теперь функция этой системы становится очевидной, она пропускает частоты, лежащие между 0,2 и 0,3, и подавляет все другие частоты. Это полосно-пропускающий фильтр. Может быть исследована также и *фаза* частотного отклика, однако, ее интерпретация *более* сложна и *менее* интересна. Она будет обсуждена в последующих главах.

Из-за небольшого числа отсчетов, задающих кривую на рис. 9.7b, кривая очень зазубрена. Эта ситуация может быть улучшена путем **дополнения** импульсного отклика нулями перед взятием ДПФ. Например, добавление нулей к импульсному отклику, увеличивающее его длину до 512 отсчетов, как это показано на рис. 9.7c, дает частотный отклик с более высоким разрешением, показанный на рис. 9.7d.

Насколько большое разрешение можно получить в частотном отклике? Ответом будет: *бесконечно* большое, если Вы пожелаете дополнить импульсную характеристику *бесконечным* числом нулей. Другими словами, нет ничего ограничивающего разрешение по частоте за исключением длинны ДПФ. Это приводит к очень важной концепции. Даже тогда, когда *импульсная* характеристика является дискретным сигналом, соответствующий частотный отклик является *непрерывным*. *N* точечное ДПФ импульсного отклика дает *N/2+1 отсчет* этой непрерывной кривой. Если Вы сделаете ДПФ длиннее, разрешение улучшится, и Вы получите лучшее представление о том, как выглядит непрерывная кривая. Помните, что представляет собой частотный отклик: амплитудные и фазовые изменения, испытываемые косинусоидальными волнами при прохождении их через систему. Поскольку входной сигнал может содержать *любую* частоту, лежащую между *0* и *0,5*, частотный отклик системы *должен* быть непрерывной кривой во всем этом диапазоне.

Это может быть понято лучше, если взять другой член семейства преобразований Фурье – **дискретно-временное преобразование Фурье (ДВПФ)**. Рассмотрим сигнал, состоящий из *N* отсчетов, пропускаемый через *N* точечное ДПФ, дающее частотную область с *N/2+1* отсчетами. Как Вы помните из предыдущей главы, ДПФ рассматривает сигнал временной области как *бесконечно длинный* и *периодический*. То есть, *N* точек повторяются снова и снова, от минус до плюс бесконечности. Теперь рассмотрим, что произойдет, когда мы начнем дополнять сигнал временной области все время

увеличивающимся числом нулей, для того чтобы получить в частотной области все улучшающуюся и улучшающуюся дискретность. Добавление нулей делает период временной области *длиннее*, при одновременном *сближении друг к другу* отсчетов частотной области.

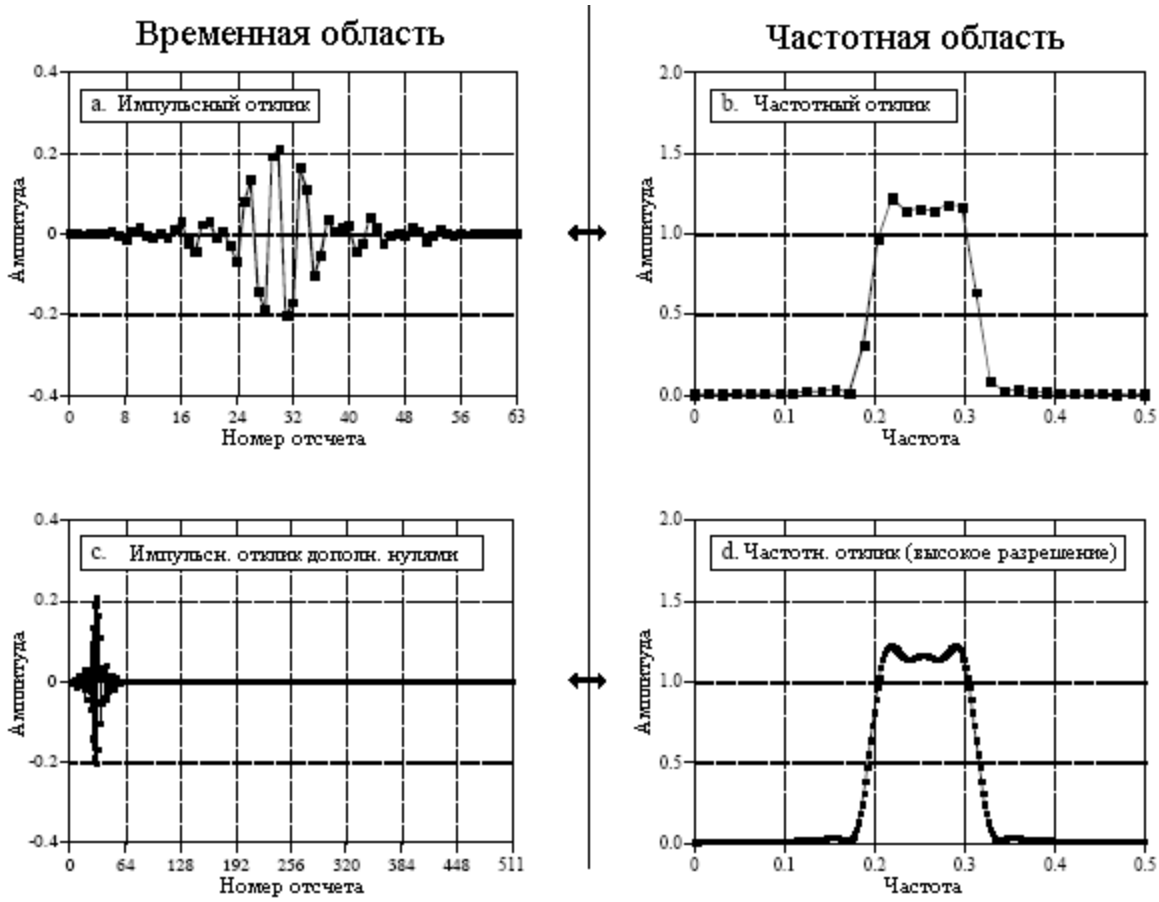


Рис. 9.7 Нахождение частотного отклика из импульсного отклика

Теперь мы доведем это до крайности, добавив *бесконечное* число нулей к сигналу временной области. Это дает иную ситуацию в двух отношениях. Во-первых, сигнал временной области теперь обладает бесконечно длинным периодом. Другими словами, он превратился в *апериодический* сигнал. Во-вторых, в частотной области достигнута бесконечно малая величина промежутка между отсчетами. То есть она стала *непрерывным сигналом*. Это и есть ДВПФ, процедура, изменяющая дискретный апериодический сигнал временной области в непрерывную кривую частотной области. Выражаясь математическим языком, частотный отклик системы находится взятием ДВПФ от ее импульсного отклика. Поскольку в компьютере это реализовано быть не может, для вычисления *отсчетов* надлежащей частотной характеристики используется ДПФ. В этом и состоит различие между тем, что Вы делаете при помощи компьютера (ДПФ) и тем, что Вы делаете при помощи математических уравнений (ДВПФ).

Свертка через частотную область

Предположим, что Вы презираете свертку. Что Вы собираетесь делать, если Вам задан и входной сигнал, и импульсный отклик и нужно найти результирующий выходной сигнал? Рис. 9.8 дает ответ: преобразуйте оба сигнала в частотную область, перемножьте их, а затем преобразуйте результат обратно во временную область. Это заменило одну свертку двумя ДПФ, одним умножением и обратным ДПФ. Даже притом, что

промежуточные шаги являются очень разными, конечный результат *идентичен* результату стандартного алгоритма свертки.

Неужели кто-то ненавидит свертку до такой степени, чтобы идти на такие хлопоты? Ответом будет - да. Свертки избегают по двум причинам. Во-первых, иметь дело со сверткой *математически* трудно. Например, предположим, Вам заданы импульсный отклик системы и ее выходной сигнал. Как Вы определите, что представляет собой входной сигнал? Это называется **обратной сверткой**, и ее во временной области фактически понять невозможно. Однако в частотной области обратная свертка может быть выполнена как простое *деление*, операция обратная умножению. Как только сложность преобразования Фурье становится меньше, чем сложность свертки, частотная область становится привлекательнее. Ведь вопрос не в том, что Вам больше нравится, вопрос в том, что Вы меньше ненавидите.

Вторая причина избегать свертки - это *скорость вычисления*. Например, предположим, Вы разрабатываете цифровой фильтр с ядром (импульсным откликом) содержащим 512 отсчетов. Используя стандартный алгоритм свертки, на персональном компьютере с числами с плавающей запятой, при тактовой частоте 200 Мгц на вычисление каждого отсчета в выходном сигнале потребуется около одной миллисекунды. Другими словами, производительность системы приблизительно составляет всего 1000 отсчетов в секунду. Это в 40 раз медленнее, чем необходимо для высококачественного воспроизведения звука, и в 10000 раз медленнее, чем необходимо для воспроизведения изображения с телевизионным качеством!

Стандартный алгоритм свертки является медленным из-за огромного числа умножений и сложений, которые должны быть выполнены. К сожалению, простой перенос проблемы в частотную область через ДПФ вообще не помогает. Просто для вычисления ДПФ требуется так же много вычислений, как и для непосредственного вычисления свертки. Прорыв в этой проблеме был сделан в начале 1960-х, когда было разработано *быстрое преобразование Фурье* (*БПФ*).

БПФ - это умный алгоритм для быстро вычисления ДПФ. При использовании БПФ свертка с помощью умножения в частотной области может быть в сотни раз быстрее, чем традиционная свертка. Задачи, вычисление которых занимало часы, выполняются за минуты. Вот почему БПФ и обработка сигналов в частотной области вызывают у людей интерес. БПФ будет представлено в Главе 12, а способ БПФ свертки в Главе 18. А сейчас сфокусируемся на том, как осуществляется свертка сигналов с помощью умножения в частотной области.

Для начала, нам нужно определить, как умножить один сигнал частотной области на другой, т.е. что означает запись: $X[f]\times H[f]=Y[f]$. В полярной форме эта запись означает, что модули умножаются: $Mag\ Y[f]=Mag\ X[f]\times Mag\ H[f]$, а фазы складываются: $Phase\ Y[f]=Phase\ X[f]+Phase\ H[f]$. Для того чтобы это понять, представьте входящую в систему косинусную волну с некоторой амплитудой и фазой. Аналогично, выходной сигнал также является косинусной волной с некоторой амплитудой и фазой. Полярная форма частотного отклика непосредственно описывает, как соотносятся две амплитуды и две фазы.

При выполнении умножения в *прямоугольной форме* между действительной и мнимой частями есть перекрестные члены. Например, поступающая на вход системы синусоидальная волна может дать на ее выходе обе, как косинусоидальную, так и синусоидальную волны. Умножать сигналы в частотной области при использовании прямоугольной системы обозначений, следует так:

$$\begin{aligned}\mathrm{Re}\,Y[f] &= \mathrm{Re}\,X[f]\mathrm{Re}\,H[f] - \mathrm{Im}\,X[f]\mathrm{Im}\,H[f] \\ \mathrm{Im}\,Y[f] &= \mathrm{Im}\,X[f]\mathrm{Re}\,H[f] + \mathrm{Re}\,X[f]\mathrm{Im}\,H[f]\end{aligned} \tag{9.1}$$

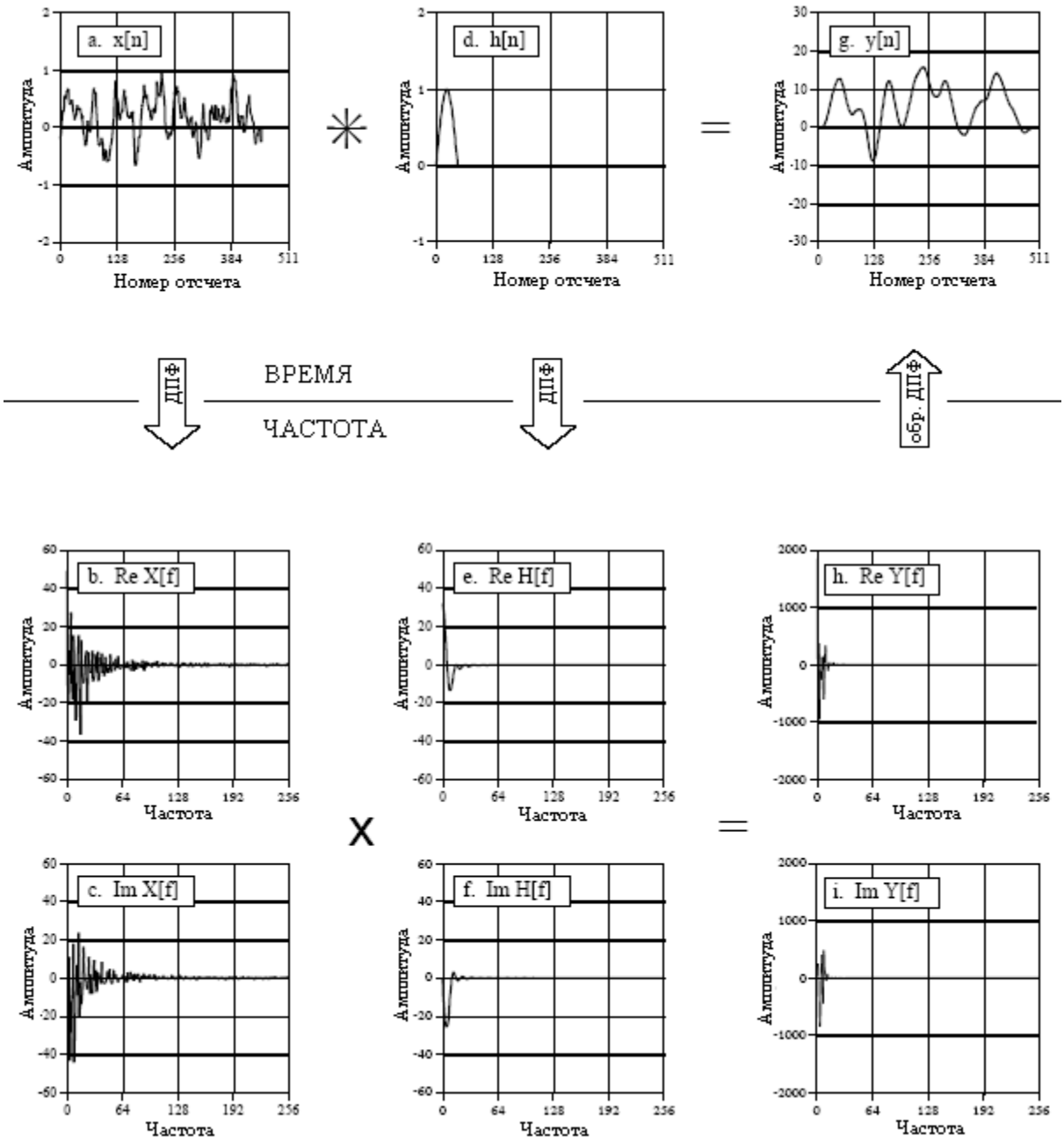


Рис. 9.8 Свертка через частотную область

Сфокусируемся на понимании умножения при использовании *полярной системы обозначений* и на идее проходящих через систему косинусных волн. Условимся, что результат этих более сложных уравнений точно такой же, как и у этой же операции, выполненной в прямоугольной форме. Например, давайте посмотрим на *деление* одного сигнала частотной области на другой. В полярной форме деление сигналов частотной области достигается операцией обратной той, что мы использовали при умножении. Для вычисления : $H[f]=Y[f]/X[f]$ разделите модули и вычтите фазы, т.е. $Mag\ H[f]=Mag\ Y[f]/Mag\ X[f]$, $Phase\ H[f]=Phase\ Y[f]-Phase\ X[f]$. В прямоугольной форме это выглядит следующим образом:

$$\mathrm{Re}\,H[f] = \frac{\mathrm{Re}\,Y[f]\mathrm{Re}\,X[f] + \mathrm{Im}\,Y[f]\mathrm{Im}\,X[f]}{\mathrm{Re}\,X[f]^2 + \mathrm{Im}\,X[f]^2}$$
$$\mathrm{Im}\,H[f] = \frac{\mathrm{Im}\,Y[f]\mathrm{Re}\,X[f] - \mathrm{Re}\,Y[f]\mathrm{Im}\,X[f]}{\mathrm{Re}\,X[f]^2 + \mathrm{Im}\,X[f]^2} \quad (9.2)$$

Теперь вернемся назад к свертке через частотную область. Вы, может быть, заметили, что на рис. 9.8 мы слегка смухлевали. Помните, что свертка *N* точечного сигнала с *M* точечным импульсным откликом дает *N+M-1* точечный выходной сигнал. Мы смухлевали, сделав всю последнюю часть входного сигнала *нулевой*, для того чтобы позволить такому расширению произойти. Определенно, рис. 9.8a содержит 453 ненулевых отсчета, а рис.9.8d содержит 60 ненулевых отсчетов. Это означает, что свертка их обоих, показанная на рис. 9.8g, может удобно разместиться в 512 получаемых точках.

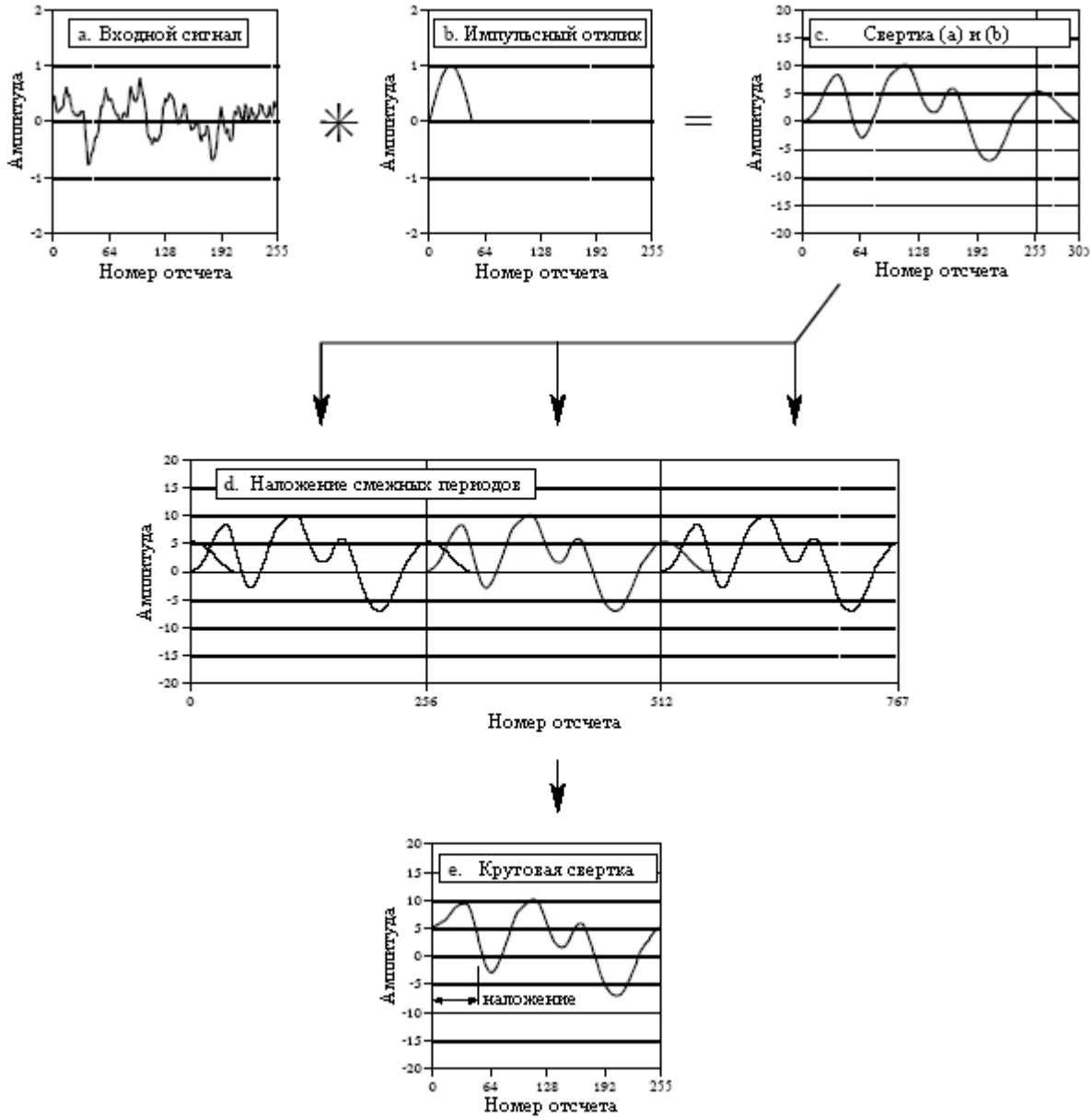


Рис. 9.9 Круговая свертка

А теперь рассмотрим более общий случай, показанный на рис. 9.9. Входной сигнал на рис. 9.9a длиной 256 точек, тогда как импульсный отклик на рис. 9.9b содержит 51 ненулевую точку. Это делает свертку двух данных сигналов, как показано на рис. 9.9c, длиной 306 точек. Проблема состоит в том, что если мы используем умножение в частотной области для выполнения операции свертки, то в выходном сигнале *допускается* наличие только 256 отсчетов. Другими словами, для переноса сигналов, показанных на рис. 9.9a и рис. 9.9b, в частотную область используется 256 точечное ДПФ. Для нахождения выходного сигнала после умножения используется 256 точечное обратное ДПФ. Как же Вы втисните 306 значений истинного сигнала в 256 точек, обеспечиваемых алгоритмом частотной области? Ответом будет - никак! 256 точечное завершение будет искаженной версией истинного сигнала. Этот процесс называется **круговой сверткой**. Она является важной, поскольку Вы желаете ее *избежать*.

Для того чтобы понять круговую свертку, следует помнить, что *N* точечное ДПФ рассматривает временную область, как бесконечно длинный периодический сигнал с *N* отсчетами на периоде. На рис. 9.9d показаны три периода того, как ДПФ рассматривает выходной сигнал в этом примере. Поскольку *N=256*, каждый период состоит из 256 точек: 0-255, 256-511 и 512-767. Свертка посредством частотной области пытается разместить 306 точечный *истинный выходной сигнал*, показанный на рис. 9.9c, в каждый из этих 256 точечных периодов. Это приводит к тому, что 49 отсчетов размещаются в соседнем периоде справа, где они накладываются на отсчеты, находящиеся здесь законно. Эти перекрывающие друг друга участки складываются и в результате, каждый из периодов приобретает вид *круговой свертки*, показанной на рис. 9.9e.

Как только природа круговой свертки понята, ее становится легко избежать. Просто дополните каждый из сигналов участвующих в свертке достаточным количеством нулевых позиций для того, чтобы позволить пространству выходного сигнала обработать *N+M-1* точек в безошибочной свертке. Например, сигналы на рис. 9.9a и рис. 9.9b могут быть дополнены нулями для того, чтобы сделать их 512 точек в длину, что позволит использовать 512 точечное ДПФ. После выполнения свертки через частотную область выходной сигнал будет состоять из 306 ненулевых отсчетов, плюс 206 отсчетов со значением равным нулю. Детально эта процедура объясняется в Главе 18.

Почему это называется круговой сверткой? Вернитесь к рис. 9.9d и изучите центральный период, отсчеты с 256 по 511. Поскольку все периоды одинаковы, часть сигнала, которая следует за этим периодом *справа*, является точно такой же, что находится от этого периода *слева*. Если Вы рассматриваете только один период, такой как на рис. 9.9e, то *становится очевидным*, что правая сторона сигнала как-то *связана* с левой стороной. Представьте змею, укусившую себя за свой собственный хвост: отсчет 255 расположен сразу перед отсчетом 0 точно так же, как отсчет 100 расположен сразу перед отсчетом101. Когда часть сигнала уходит вправо, она волшебным образом появляется слева. Другими словами, *N* точечная временная область ведет себя так, как будто она является *круглой*.

В предыдущей главе мы сформулировали вопрос: действительно ли имеет значение, если вместо бесконечно длинного периодического сигнала с периодом *N*, временная область ДПФ рассматривается как *N* точечная? Круговая свертка является примером, где это *имеет* значение. Если сигнал временной области понимается как *периодический*, то искажение, с которым сталкиваются в круговой свертке, может быть объяснено просто как расширившийся из одного периода в другой сигнал. Для сравнения, если рассматриваются только *N* точек временной области, то приходим к довольно причудливому заключению. То есть, свертка через частотную область действует так, как если бы временная область каким-то образом деформировалась в круглое кольцо так, что *0* отсчет располагался сразу за *N-1* отсчетом.

ГЛАВА
10

СВОЙСТВА ПРЕОБРАЗОВАНИЯ ФУРЬЕ

Временная и частотная области являются альтернативными способами представления сигналов. Преобразование Фурье является математическим соотношением, устанавливающим связь между этими двумя представлениями. Если сигнал изменяется в одной области, он будет также изменяться и в другой области, хотя обычно и не таким же образом. Например, в предыдущей главе было показано, что *свертка* сигналов временной области дает *произведение* их частотных спектров. Другие математические операции, такие как суммирование, масштабирование и сдвиг, также имеют соответствующие операции в противоположной области. Эти соотношения, описывающие, как математические изменения в одной области выражаются математическими изменениями в другой области, называются свойствами преобразования Фурье.

Линейность преобразования Фурье

Преобразование Фурье является *линейным*, то есть оно обладает свойствами *однородности* и *аддитивности*. Это справедливо для всех четырех членов семейства преобразований Фурье (преобразования Фурье, ряда Фурье, ДПФ и ДВПФ).

На рисунке 10.1 приводится пример того, что однородность является свойством Преобразования Фурье. На рисунке 10.1a показан произвольный сигнал временной области и соответствующий ему частотный спектр, приведенный на рис. 10.1b. Мы будем называть эти два сигнала соответственно $x[\ ]$ и $X[\ ]$. *Однородность* означает, что изменения амплитуды в одной области дают идентичные изменения амплитуды в другой области. Это должно чувствоваться на уровне интуиции: когда амплитуда формы волны временной области изменяется, амплитуда синусных и косинусных волн составляющих данную форму волны должна также изменяться на равную величину.

В математической форме, если $x[\ ]$ и $X[\ ]$ являются парой преобразования Фурье, тогда $kx[\ ]$ и $kX[\ ]$ для любой постоянной k, также являются парой преобразования Фурье. Если частотная область представлена в *прямоугольной* системе обозначений, $kX[\ ]$ означает, что обе и действительная и мнимая части умножаются на k. Если частотная область представлена в *полярной* системе обозначений, $kX[\ ]$ означает, что модуль умножается на k, а фаза остается без изменений.

Аддитивность преобразования Фурье означает, что *сложение* в одной области соответствует *сложению* в другой области. Пример этого показан на рис. 10.2. В этой иллюстрации на рис. 10.2a и рис. 10.2b представлены сигналы, называемые во временной области соответственно $x_1[\ ]$ и $x_2[\ ]$. Сложение этих сигналов дает третий сигнал временной области, называемый $x_3[\ ]$, показанный на рис. 10.2c. Каждый из этих трех сигналов имеет частотный спектр, состоящий из действительной и мнимой частей, показанных на рис. 10.2 от d до i. Поскольку для того, чтобы дать третий сигнал временной области, два сигнала временной области *складываются*, два соответствующих спектра *складываются*, для того чтобы дать третий спектр. В прямоугольной системе обозначений частотные спектры складываются сложением действительной части с действительной частью, а мнимой части с мнимой частью. Если: $x_1[n]+x_2[n]=x_3[n]$, тогда

Re $X_1[f]$+*Re* $X_2[f]$=*Re* $X_3[f]$ и *Im* $X_1[f]$+*Im* $X_2[f]$=*Im* $X_3[f]$. Поразмышляйте об этом в терминах косинусоидальных и синусоидальных волн. Складываются все косинусоидальные волны (действительные части) и все синусоидальные волны (мнимые части) без каких бы то ни было перекрестных членов между ними.

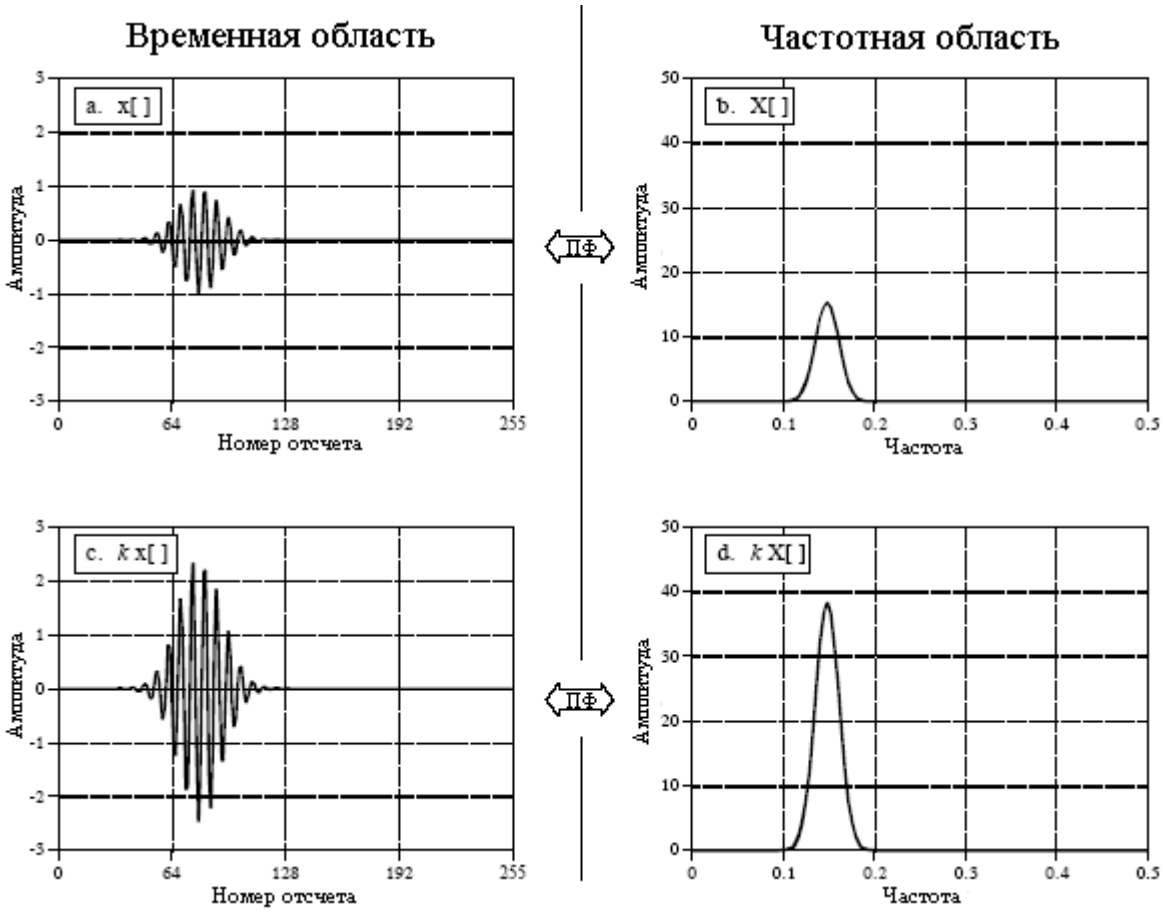


Рис. 10.1 Однородность преобразования Фурье

В полярной форме частотные спектры не могут непосредственно складываться, они должны быть преобразованы в прямоугольную систему обозначений, сложены, а затем преобразованы обратно в полярную форму. Это также может быть понято в терминах того, как ведут себя синусоиды. Представьте сложение двух синусоид имеющих туже самую частоту, но разные амплитуды (A_1 и A_2) и фазы (φ_1 и φ_2). Если так случится, что две фазы будут одинаковыми ($\varphi_1=\varphi_2$), при сложении синусоид, амплитуды будут суммироваться (A_1+A_2). Однако если так случится, что две фазы будут точно противоположны ($\varphi_1= -\varphi_2$), при сложении синусоид амплитуды будут *вычитаться* (A_1-A_2). Вывод следующий, когда синусоиды (или спектры) представлены в полярной форме, они *не могут* быть сложены простым суммированием их модулей и фаз.

Будучи линейным, преобразование Фурье *не является* инвариантным сдвигу. Другими словами, сдвиг во временной области *не соответствует* сдвигу в частотной области. Это является темой следующего раздела.

Характеристики фазы

В математической форме: если $x[n] \leftrightarrow$ *Mag* $X[f]$ и *Phase* $X[f]$. Тогда сдвиг во временной области приводит к: $x[n+s] \leftrightarrow$ *Mag* $X[f]$ и *Phase* $X[f]+2\pi sf$ (где f выражается как часть частоты дискретизации, изменяющаяся между 0 и 0,5). Выражаясь словами, сдвиг на

s отсчетов во временной области оставляет модуль без изменений, но добавляет к фазе линейный член $2\pi sf$. Давайте посмотрим пример того, как это все работает.

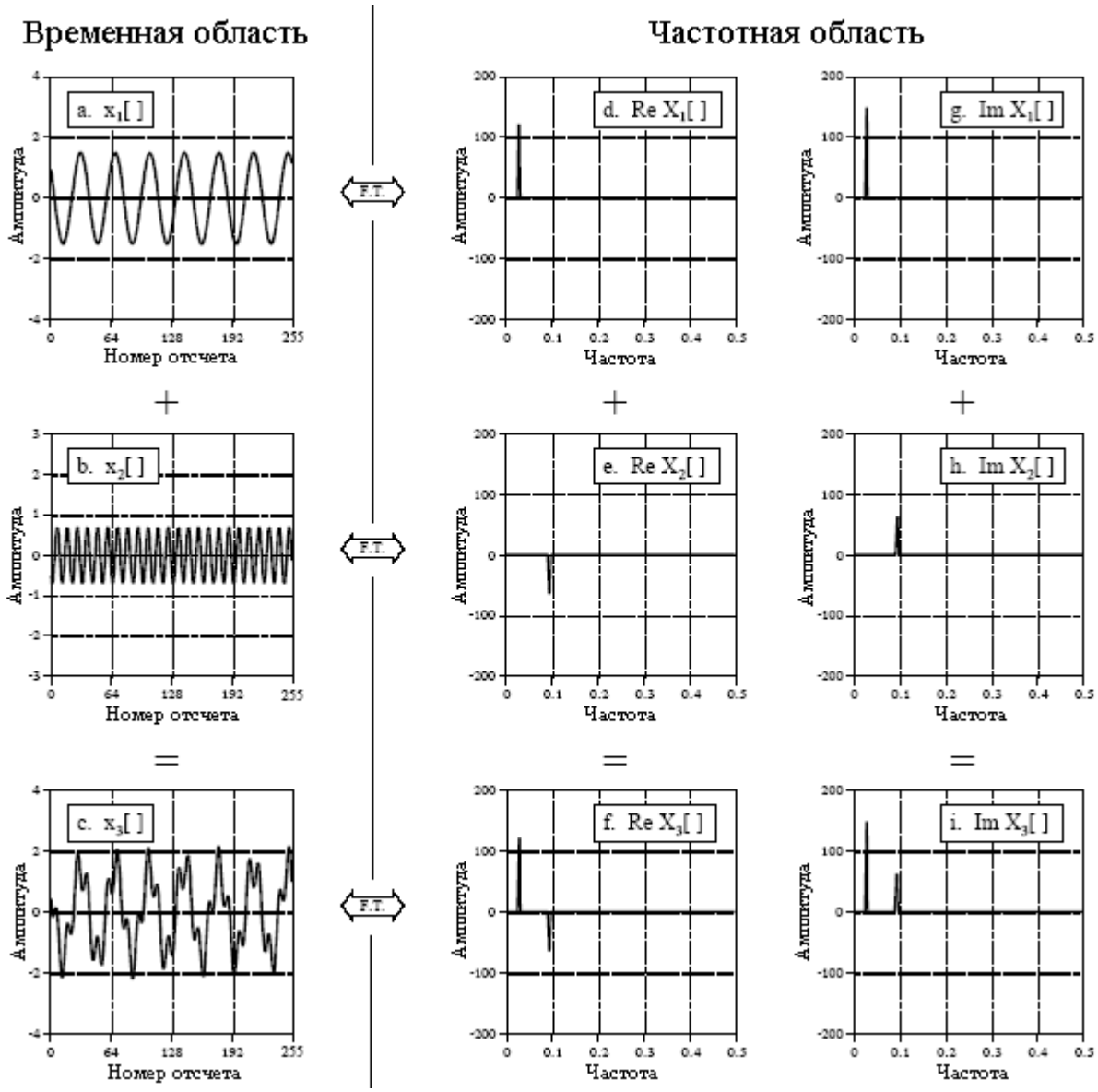


Рис. 10.2 Аддитивность преобразования Фурье

На рис. 10.3 показано как влияет на фазу сдвиг формы волны во временной области влево или вправо. Модуль не был включен в эту иллюстрацию, поскольку он не представляет интереса, при сдвиге во временной области он не изменяется. На рис. 10.3 с a по d форма волны постепенно сдвигается от положения с центром пика на отсчете 128 к положению с центром пика на отсчете 0. Эта последовательность графиков учитывает, что ДПФ рассматривает временную область как *круговую*; т.е. когда части формы волны уходят вправо, они вновь появляются слева.

Форма волны временной области на рис. 10.3 симметрична относительно вертикальной оси, то есть левая и правая стороны являются зеркальным отражением друг друга. Как отмечалось в Главе 7, сигналы с таким типом симметрии называются сигналами с *линейной фазой*, поскольку фаза их частотного спектра является *прямой линией*. Подобно этому, сигналы, не имеющие такой симметрии слева направо, называются сигналами с *нелинейной фазой* и обладают фазой, которая является чем-то другим, нежели прямой линией. На рис. 10.3 с e по h показана фаза для сигналов,

приведенных соответственно на рис. 10.3 с a по d. Как описывалось в Главе 8, данные сигналы фазы *развернуты*, что дает возможность рассматривать их без разрывов, ассоциирующихся с удержанием значения фазы между π и $-\pi$.

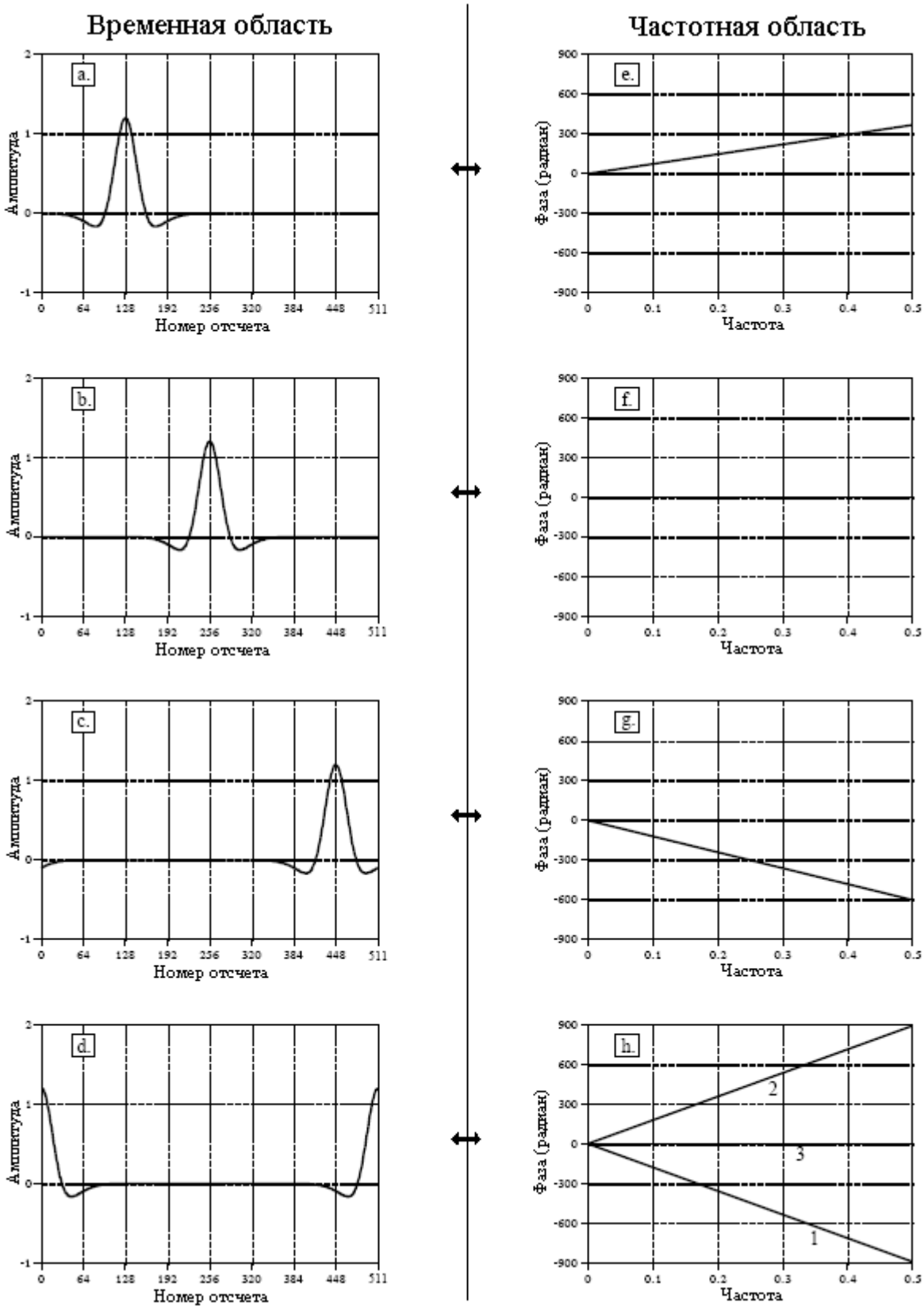


Рис. 10.3 Фазовые изменения как результат сдвига во временной области

Когда форма волны временной области сдвигается вправо, фаза остается прямой линией, но претерпевает *уменьшение* наклона. Когда форма волны временной области сдвигается влево, наблюдается *увеличение* наклона. Это основное свойство, которое Вам следует запомнить из этого раздела; сдвиг формы волны во временной области соответствует изменению наклона фазы.

На рис. 10.3b и рис. 10.3f показан уникальный случай, в котором фаза везде равна нулю. Это происходит тогда, когда сигнал временной области *симметричен* относительно *нулевого* отсчета. На первый взгляд эта симметрия на рис. 10.3b может быть не очевидна, вместо этого может показаться, что сигнал симметричен относительно отсчета 256 (то есть, *N/2*). Помните, что ДПФ рассматривает временную область как круговую, с нулевым отсчетом неразрывно связанным с отсчетом *N-1*. Любой сигнал симметричный относительно нулевого отсчета будет также симметричен относительно отсчета *N/2*, и наоборот. При использовании членов семейства преобразований Фурье, которые не рассматривают временную область как периодическую (таких как ДВПФ), для получения нулевой фазы симметрия должна быть относительно нулевого отсчета.

На рис. 10.3d и рис. 10.3h изображено что-то похожее на загадку. Прежде всего, представьте, что сигнал на рис. 10.3d был сформирован за счет сдвига формы волны вправо немного большего, чем на рис. 10.3c. Это означает, что фаза на рис. 10.3h будет иметь слегка более отрицательный наклон, чем на рис. 10.3g. Эта фаза показана как линия 1. Далее, представьте, что сигнал на рис. 10.3d был сформирован из кривой, показанной на рис. 10.3a, за счет сдвига ее формы волны влево. В этом случае фаза должна иметь слегка более положительный наклон, чем на рис. 10.3e, как это показано при помощи линии 2. Наконец, обратите внимание, что кривая на рис. 10.3d симметрична относительно отсчета *N/2* и, следовательно, имеет нулевую фазу, как это показано при помощи линии 3. Какая же из этих трех фаз правильная? Все они являются правильными в зависимости от того, какая обуславливается неоднозначность фазы: π или 2π. Например, каждый отсчет на линии 2 отличается от соответствующего отсчета на линии 1 на целое число, кратное 2π, делающее их равными. Чтобы установить взаимосвязь линии 3 с линиями 1 и 2, π неоднозначность должна также быть принята во внимание.

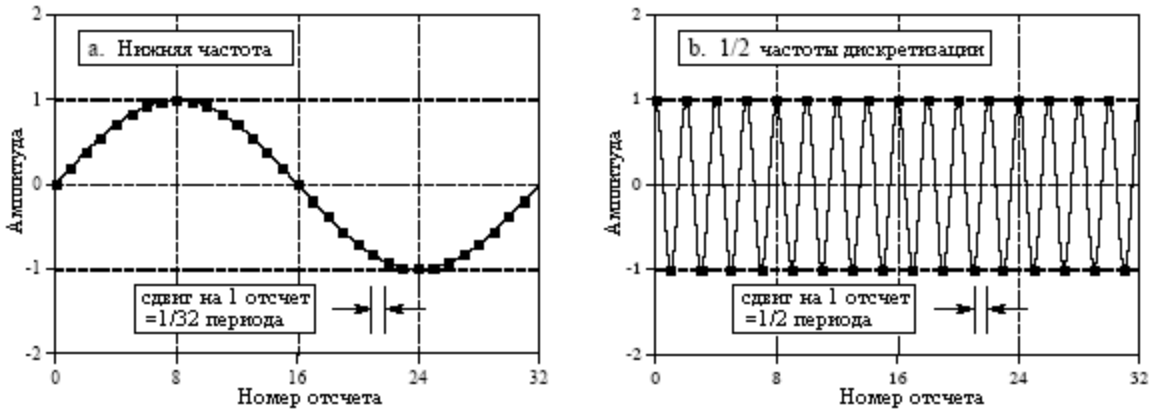


Рис. 10.4 Связь между отсчетами и фазой

Для того чтобы понять, почему фаза ведет себя таким образом, представьте сдвиг формы волны на *один* отсчет вправо. Это значит, что все синусоиды, составляющие форму волны, также должны быть сдвинуты на *один* отсчет вправо. На рис. 10.4 показаны две синусоиды, которые могут быть частью формы волны. На рис. 10.4a синусоидальная волна имеет низкую частоту, и сдвиг на один отсчет является только небольшой частью всего периода. На рис. 10.4b синусоидальная волна имеет частоту равную половине частоты дискретизации, наивысшую частоту, которая только может существовать в оцифрованных данных. Сдвиг на один отсчет на этой частоте равен половине всего

периода или π радиан. То есть, когда сдвиг выражается в терминах изменения фазы, он становится *пропорциональным* частоте сдвигаемой синусоиды.

Например, рассмотрим форму волны симметричную относительно нулевого отсчета и, следовательно, имеющую нулевую фазу. На рис. 10.5a показано, как изменяется фаза этого сигнала при его сдвиге влево или вправо. На самой высокой частоте, равной половине частоты дискретизации, фаза увеличивается на π при каждом сдвиге на один отсчет влево и уменьшается на π при каждом сдвиге на один отсчет вправо. На нулевой частоте фазовый сдвиг отсутствует, а для всех частот, лежащих между ними, находится на прямой линии.

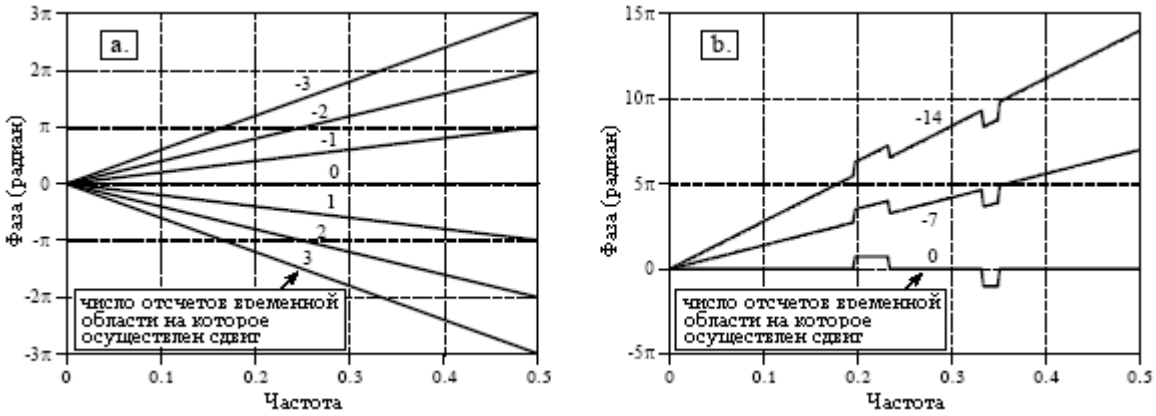


Рис. 10.5 Фазы получающиеся в результате сдвига во временной области

Все примеры, которые мы использовали до сих пор, являются примерами сигналов с *линейной* фазой. На рис. 10.5b показано, что сигналы с *нелинейной* фазой реагируют на сдвиг точно таким же образом. В этом примере нелинейная фаза представляет собой прямую линию с двумя прямоугольными пульсациями. Когда осуществляется сдвиг во временной области, эти нелинейные особенности просто накладываются на изменяющийся наклон.

Что происходит с *действительной* и *мнимой частями*, при осуществлении сдвига формы волны во временной области? Вспомните, что сигналы частотной области в прямоугольной системе обозначений почти недоступны для понимания человека. Действительная и мнимая части обычно выглядят как произвольные колебания с неочевидным рисунком. Когда осуществляется сдвиг сигнала во временной области, то волнистые образцы действительной и мнимой частей становятся еще более колебательными и трудными для интерпретации. Не тратьте свое время на понимание этих сигналов или на то, как они изменяются при сдвиге во временной области.

На рис. 10.6 приведена интересная демонстрация того, какая информация содержится в *фазе*, а какая информация содержится в *модуле*. Форма волны на рис. 10.6a имеет две очень характерные отличительные особенности: нарастающий фронт на отсчете 55 и падающий фронт на отсчете 110. Фронты очень важны при кодировании информации в *форме* волны. Фронт указывает на *момент, в который случается* некоторое событие, отделяя все, что находится слева, от того, что находится справа. Это информация закодированная во временной области, в ее самом чистом виде. Для того чтобы начать демонстрацию, от сигнала, показанного на рис. 10.6a, берется ДПФ, и частотный спектр преобразуется в полярную систему обозначений. Для получения сигнала на рис. 10.6b данные о фазе были заменены случайным набором чисел, лежащих между $-\pi$ и π, затем, для реконструкции сигнала временной области использовалось обратное ДПФ. Другими словами, кривая на рис. 10.6b основана только на информации, содержащейся в *модуле*. Подобным же образом находится кривая на рис. 10.6c, основанная исключительно на информации, содержащейся в *фазе*.

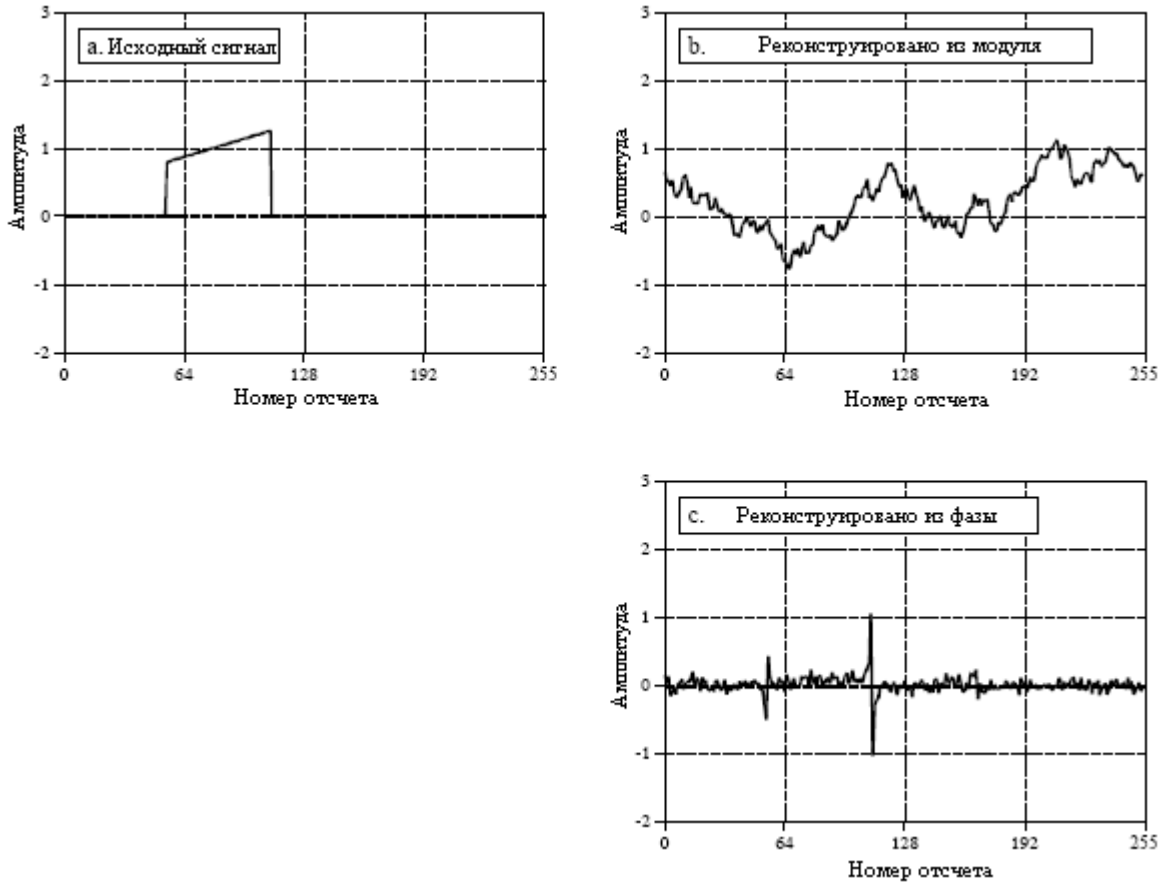


Рис. 10.6 Информация содержащаяся в фазе

Каков же результат? Местоположение фронтов четко представлено на рис. 10.6c, но полностью отсутствует на рис. 10.6b. Это объясняется тем, что фронт формируется тогда, когда множество синусоид *нарастает* в одном и том же месте, что возможно только тогда, когда их *фазы* скоординированы. Вкратце, большая часть информации о виде формы волны временной области содержится в *фазе*, а не в *модуле*. Этому можно противопоставить сигналы, подобные аудио сигналам, у которых их информация закодирована в частотной области. Для этих сигналов модуль является более важным, а фаза играет незначительную роль. В последующих главах мы увидим, что такой способ понимания вещей вооружает стратегиями для конструирования фильтров и других методов обработки сигналов. Первым шагом в успешной ЦОС всегда является понимание того, каким образом информация представлена в сигнале.

Почему же симметрия слева направо соответствует нулевой либо линейной фазе? Ответ на этот вопрос дает рис. 10.7. Как показано на рис. 10.7a, b и c, такой сигнал может быть разложен на левую и правую половины. Отсчет, находящийся в центре симметрии (в данном случае нулевой) ровно поделен между левой и правой половинами, позволяя двум сторонам быть точными зеркальными изображениями друг друга. Как показано на рис. 10.7e и рис. 10.7f, модули этих двух половин будут *идентичными*, а фазы, как это показано на рис. 10.7h и рис. 10.7i, будут противоположны по знаку. Отсюда следуют две важные концепции. Во-первых, каждый сигнал, обладающий симметрией слева направо, будет иметь линейную фазу *потому*, что нелинейная фаза левой половины полностью аннулирует нелинейную фазу правой половины.

Во-вторых, представьте поворот кривой на рис. 10.7b такой, что она становится как на рис. 10.7c. Подобный переворот слева направо во временной области ничего не делает с модулем, но изменяет знак каждой точки по фазе. Аналогично, изменение знака фазы

переворачивает сигнал временной области слева направо. Если сигнал является непрерывным, поворот происходит относительно нуля. Если сигнал является дискретным, поворот происходит одновременно относительно нулевого отсчета *и* отсчета *N/2*.

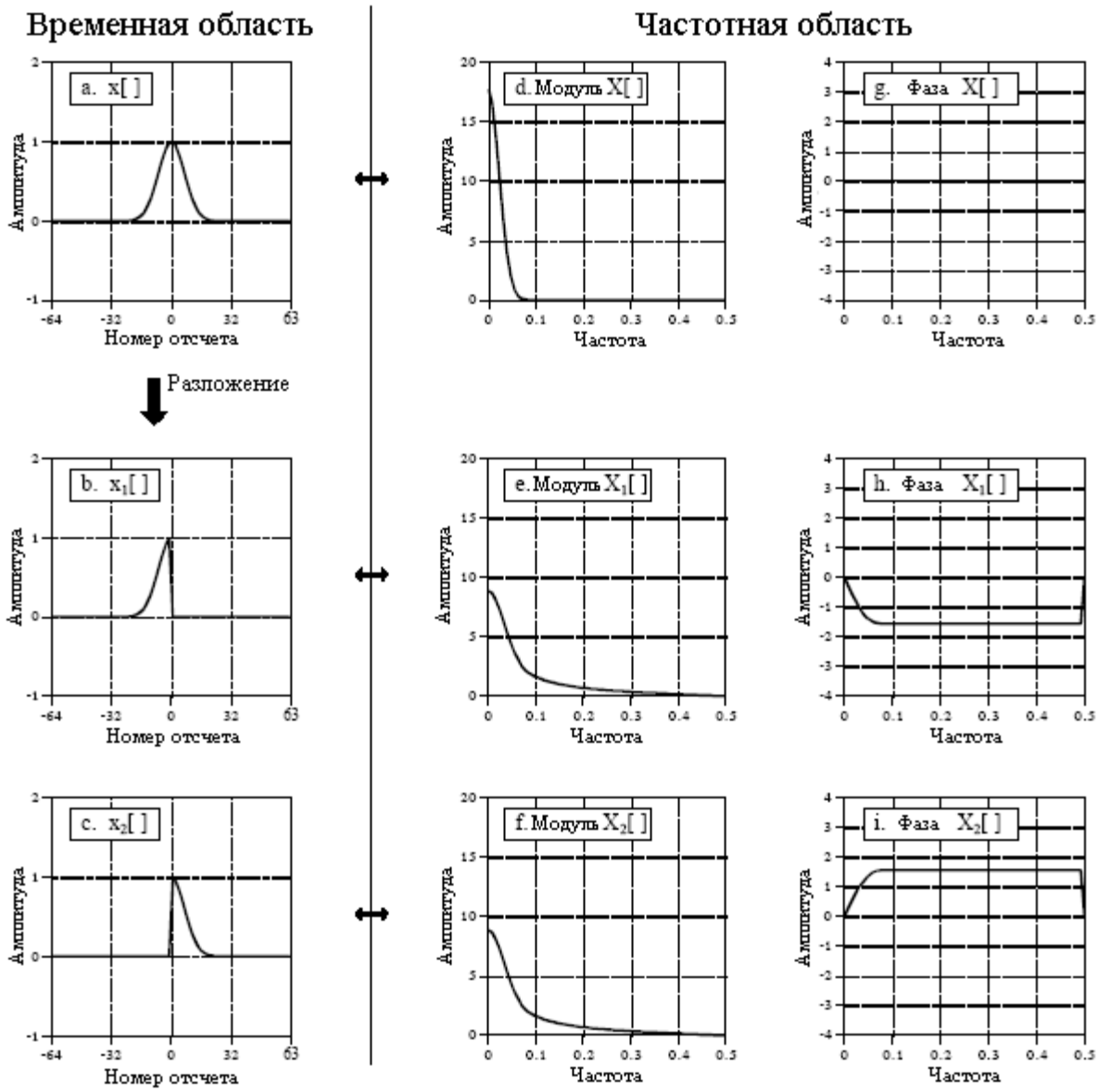


Рис. 10.7 Фазовые характеристики симметрии слева на право

Изменение знака фазы - это достаточно обычная операция, которой присвоено свое собственное название и символ. Это название - **комплексное спряжение**, и оно представляется размещением звездочки в верхнем правом углу переменной. Например, если $X[f]$ состоит из *Mag* $X[f]$ и *Phase* $X[f]$, тогда $X^*[f]$ называется *комплексно сопряженным* и состоит из *Mag* $X[f]$ и *-Phase* $X[f]$. В прямоугольной системе обозначений комплексно сопряженная величина находится изменением знака мнимой части, при этом действительная часть остается без изменений. Выражаясь математическими терминами, если $X[f]$ состоит из *Re* $X[f]$ и *Im* $X[f]$, тогда $X^*[f]$ состоит из *Re* $X[f]$ и *-Im* $X[f]$.

Вот несколько примеров того, как комплексное сопряжение используется в ЦОС. Если $x[n]$ имеет преобразованием Фурье $X[f]$, тогда $x[-n]$ имеет преобразованием Фурье $X^*[f]$. Выражаясь словами, переворачивание временной области слева направо соответствует изменению знака фазы. В качестве еще одного примера, вспомните из Главы 7, что корреляция может быть представлена как свертка. Это осуществляется

переворачиванием одного из сигналов слева направо. В математической форме $a[n]*b[n]$ является сверткой, в то время как $a[n]*b[-n]$ является корреляцией. В частотной области эти операции соответствуют $A[f]\times B[f]$ и $A[f]\times B^*[f]$, соответственно. В качестве последнего примера, рассмотрим произвольный сигнал $x[n]$и его частотный спектр $X[f]$. Частотный спектр может быть изменен до спектра с *нулевой фазой* умножением его на его комплексное сопряжение, то есть $X[f]\times X^*[f]$. Выражаясь словами, какой бы ни была фаза $X[f]$, она будет аннулирована прибавлением ее противоположности (помните, при перемножении частотных спектров их фазы складываются). Во временной области это означает, что $x[n]*x[-n]$ (осуществляется свертка сигнала с его собственной версией перевернутой слева направо) будет иметь симметрию слева направо относительно нулевого отсчета, независимо от того, что представляет собой $x[n]$.

Для многих инженеров и математиков этот вид манипуляции и *есть* ЦОС. Если Вы хотите быть способными общаться с этой группой специалистов, то Вы должны привыкать к использованию их языка.

Периодический характер ДПФ

В отличие от других трех преобразований Фурье, ДПФ рассматривает *обе* и временную область, и частотную область как *периодические*. Это может приводить в замешательство и быть неудобным, поскольку большинство сигналов, используемых в ЦОС, *не являются* периодическими. Тем не менее, если Вы хотите использовать ДПФ, Вы должны приспособиться к взглядам ДПФ на мир.

На рис. 10.8 показаны две различных интерпретации сигнала временной области. Прежде всего, посмотрите на верхний сигнал, где временная область рассматривается как N точечная. Он представляет то, как обычно снимаются цифровые сигналы в научных экспериментах и инженерных приложениях. Например, данные 128 отсчетов могли быть получены дискретизацией некоторого параметра через одинаковые интервалы *времени*. Отсчет 0 является отличным и отдельным от отсчета 127, поскольку они были получены в *разное* время. Из того, как был сформирован этот сигнал, нет никаких оснований полагать, что отсчеты слева от сигнала каким-то образом связаны с отсчетами справа от сигнала.

К сожалению ДПФ не рассматривает вещи данным образом. Как показано на нижнем рисунке ДПФ рассматривает эти 128 точек как единичный период бесконечно длинного периодического сигнала. Это означает, что левая сторона полученного сигнала соединена с правой стороной копии этого сигнала. Аналогично, правая сторона полученного сигнала соединена с левой стороной идентичного периода. Об этом можно также думать как о правой стороне полученного сигнала, изогнутой вокруг, и соединенной с его левой стороной. С этой точки зрения отсчет 0 появляется следующим за отсчетом 127, аналогично тому, как отсчет 44 появляется следующим за отсчетом 43. Это точно так же, как перемещаться по **кругу,** и идентично рассмотрению сигнала как *периодического*.

Наиболее серьезным последствием периодичности временной области является **совмещение имен во временной области**. Для иллюстрации этого, предположим, что мы берем сигнал временной области и пропускаем его через ДПФ, для того чтобы найти его частотный спектр. Мы можем немедленно пропустить этот частотный спектр через обратное ДПФ для восстановления исходного сигнала временной области, но вся эта процедура не была бы очень интересной. Вместо этого, перед использованием обратного ДПФ мы будем некоторым образом модифицировать частотный спектр. Например, определенные частоты могли бы быть удалены, изменены по амплитуде или фазе, перемещены по кругу и т.д. Это те виды вещей, которые обычно проделываются в ЦОС. К сожалению, такие изменения в частотной области могут привести к созданию сигнала временной области слишком длинного, чтобы уместиться в одном периоде. Это заставляет сигнал переходить из одного периода в смежные периоды. Когда временная область

рассматривается как *круговая*, части сигнала, которые выходят за пределы справа, вдруг, как кажется, вновь появляются на левой стороне сигнала, и наоборот. То есть, выходящие за пределы части сигнала *совмещают* сами себя с новым местом во временной области. Если случается так, что это новое место уже содержит существующий сигнал, весь ералаш суммируется, приводя к потере информации. Круговая свертка, получающаяся из умножения в частотной области (обсужденная в Главе 9) является великолепным примером совмещения имен такого типа.

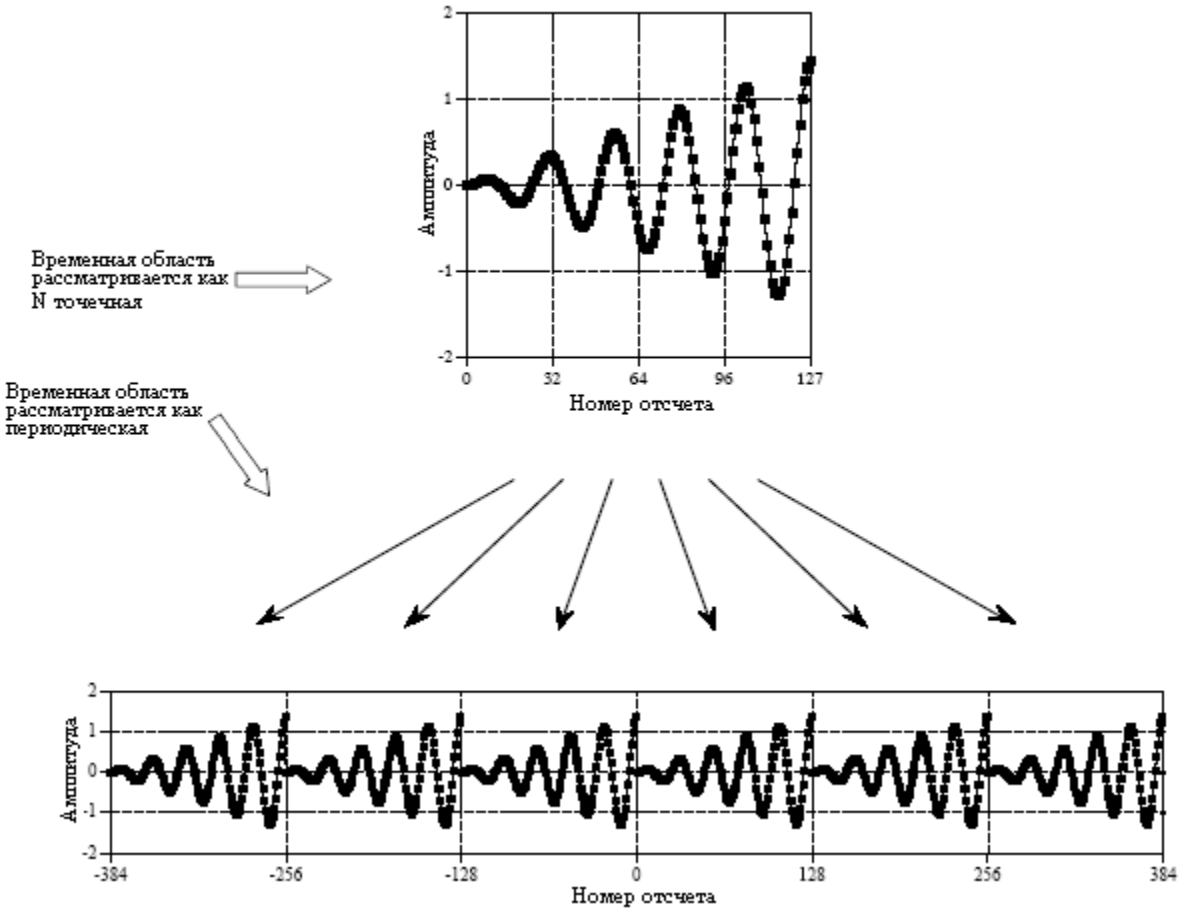


Рис. 10.8 Периодичность сигнала ДПФ во временной области

Периодичность в частотной области проявляется во многом таким же образом, но является более сложной. Такой пример показан на рис. 10.9. Верхние рисунки показывают модуль и фазу частотного спектра, рассматриваемого как состоящего из *N/2+1* отсчетов, расположенных между 0 и 0,5 от частоты дискретизации. Это наиболее простой способ рассмотрения частотного спектра, но он не объясняет многие свойства ДПФ.

Два нижних рисунка показывают, как ДПФ рассматривает такой частотный спектр в виде периодического. Ключевой характеристикой является то, что частотный спектр между 0 и 0,5, оказывается, имеет *зеркальное изображение* частот, лежащих между 0 и –0,5. Это зеркальное изображение **отрицательных частот** слегка отличается по модулю и фазе сигналов. По модулю, сигнал перевернут слева направо. По фазе, сигнал перевернут слева направо *и* изменен по знаку. Как Вы помните, этим двум типам симметрии даны названия: говорят, модуль является **четным** сигналом (он обладает *четной* симметрией), в то время как фаза, говорят, является **нечетным** сигналом (она обладает *нечетной* симметрией). Если частотный спектр приводится к действительной и мнимой частям, *действительная часть* всегда будет *четной*, в то время как *мнимая часть* всегда будет *нечетной*.

Принимая во внимание эти отрицательные частоты, ДПФ рассматривает частотную область как периодическую, с периодом в 1,0 от частоты дискретизации, таким как от –0,5 до 0,5 или от 0 до 1,0. В терминах номеров отсчетов это делает длину периода частотной области равной *N*, такой же, как и во временной области.

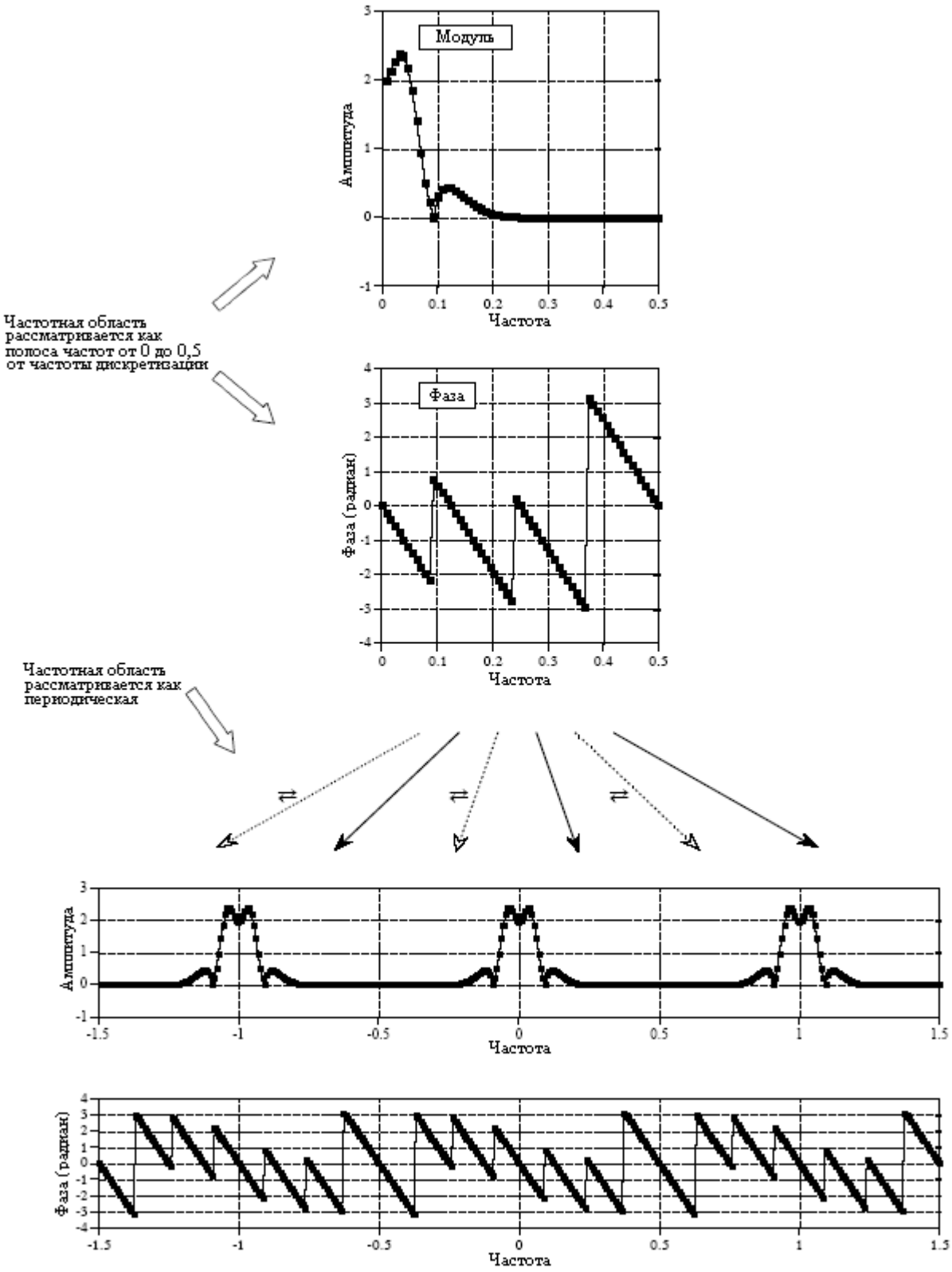


Рис. 10.9 Периодичность частотной области ДПФ

Периодичность в частотной области делает ее чувствительной к **совмещению имен в частотной области,** полностью аналогичному предварительно описанному совмещению имен во временной области. Представьте сигнал временной области соответствующий некоторому частотному спектру. Если сигнал временной области модифицируется, то, очевидно, что частотный спектр тоже будет изменен. Если измененный частотный спектр не сможет уместиться в отведенное пространство, он протиснется в примыкающие периоды. Так же как и ранее, такое совмещение вызовет две проблемы: частоты находятся не там, где они должны быть, и разрушающее информацию суммирование наложившихся частот из разных периодов.

Совмещение имен в частотной области более тяжело для понимания, чем совмещение имен во временной области, поскольку в частотной области периодический рисунок более сложен. Рассмотрим одиночную частоту, которая вынуждена переместиться в частотной области из 0,01 в 0,49. Соответствующая отрицательная частота, следовательно, перемещается из -0,01 в –0,49. При перемещении положительных частот через барьер 0,5, отрицательные частоты перескакивают через барьер –0,5. Поскольку частотная область является периодической, точно такие же события происходят и в других периодах, например между 0,5 и 1,5. Клон положительной частоты пересекает частоту 1,5 слева направо, в то время как клон отрицательной частоты пересекает 0,5 справа налево. А сейчас представьте себе, как это выглядит, если Вы можете видеть только полосу частот от 0 до 0,5. Кажется, что частота, покинувшая эту полосу частот *вправо*, появляется *с правой* стороны, но движется в противоположном направлении.

Рисунок 10.10 показывает, как появляется совмещение имен во временной и частотной областях в случае если рассматривается только один период. Как показано на рис. 10.10a, если один конец сигнала временной области настолько длинен, что не помещается в пределах одного периода, то выступающий конец будет *отрезан* и *наклеен* на другой конец. Для сравнения, на рис. 10.10b показано, что когда сигнал частотной области выходит за пределы периода, выступающий конец *загибается*. Независимо от того, где заканчивается сегмент совмещения имен, он складывается с любым находящимся уже там сигналом, разрушая информацию.

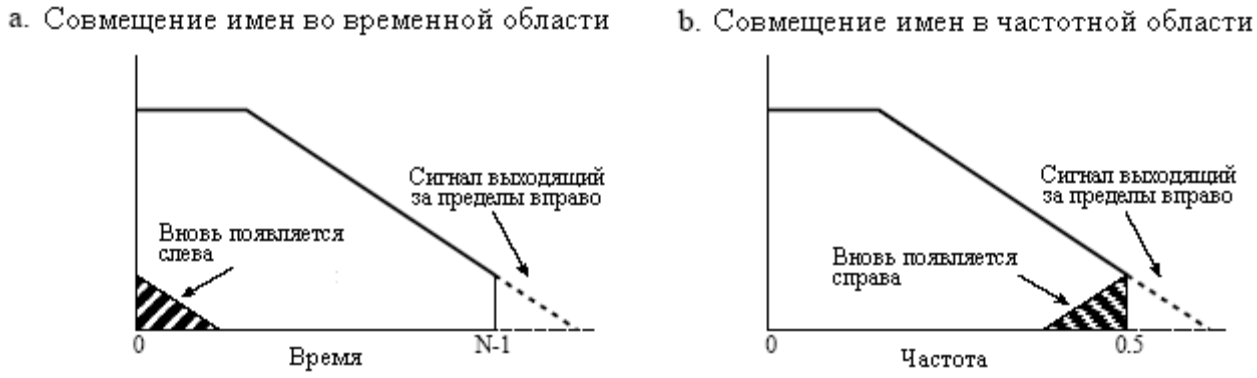


Рис. 10.10 Примеры совмещения имен во временной и частотной областях

Давайте повнимательнее рассмотрим те странные вещи, которые называются *отрицательными частотами*. Являются ли они просто некоторым причудливым экспонатом математики, или они имеют значение в реальном мире? На рис. 10.11 показано, что они из себя представляют. На рис. 10.11a представлен дискретный сигнал, состоящий из 32 отсчетов. Представьте, что Вам дана задача найти частотный спектр, соответствующий этим 32 точкам. Для того чтобы облегчить Вам работу, Вам сказали, что эти точки представляют дискретную косинусоидальную волну. Другими словами, Вы должны найти частоту и фазовый сдвиг (f и θ) так, чтобы $x[n]=cos(2\pi nf/N+\theta)$

соответствовало заданным отсчетам. До того как Вы придумали решение, показанное на рис. 10.11b, то есть, $f=3$ и $\theta=-\pi/4$, прошло не так много времени. Если Вы прекратите свой анализ в этой точке, Вы получите только 1/3 похвалы за решение этой проблемы. Это связано с тем, что существует два других решения, которые Вы пропустили. Как показано на рис. 10.11c, вторым решением является $f=-3$ и $\theta=\pi/4$. Даже если идея *отрицательных частот* оскорбляет ваши чувства, это не меняет того факта, что это является математически правильным решением заданной задачи. Каждая синусоида с *положительной* частотой может альтернативно быть выражена как синусоида с *отрицательной* частотой. Это применимо как к непрерывным, так и к дискретным сигналам.

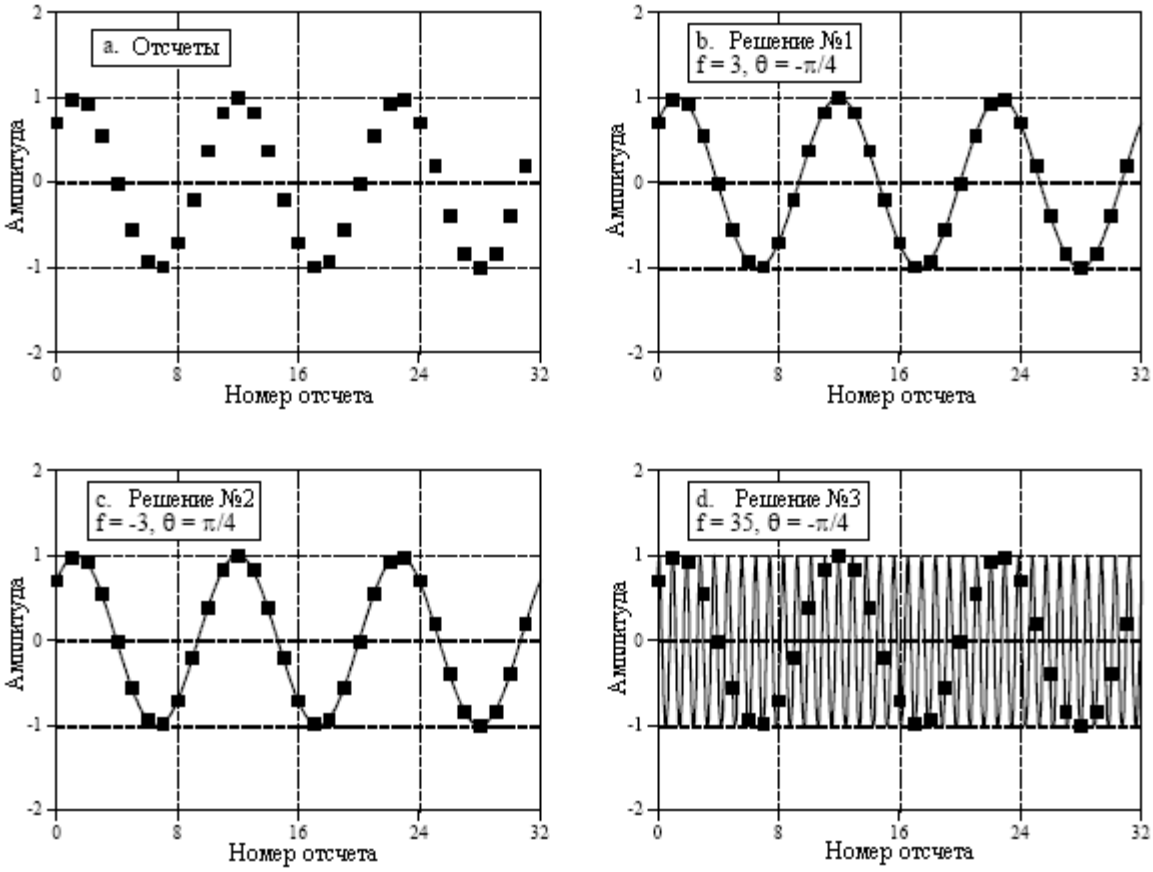


Рис. 10.11 Значение отрицательных частот

Третье решение является не единственным ответом, а бесконечным семейством синусоид. Как показано на рис. 10.11d, синусоида с $f=35$ и $\theta=-\pi/4$ проходит через все дискретные точки и, следовательно, является правильным решением. Может смущать тот факт, что между отсчетами укладывается почти колебание, однако это не лишает ответ достоверности. Аналогично, $f=\pm29$, $f=\pm35$, $f=\pm61$ и $f=\pm67$ - все являются решениями с многократными колебаниями между точками.

Каждое из этих трех решений соответствует разным участкам частотного спектра. Для дискретных сигналов первое решение соответствует частотам, находящимся меду 0 и 0,5 от частоты дискретизации. Второе решение находится в диапазоне частот от 0 до –0,5. Наконец, третье решение составляет бесконечное число дублированных частот, находящихся ниже –0,5 и выше чем 0,5. Если анализируемый нами сигнал является непрерывным, то результатом первого решения будут частоты, находящиеся в диапазоне от нуля до плюс бесконечности, в то время как результатом второго решения будут

частоты, находящиеся в диапазоне от нуля до минус бесконечности. Третьей группы решений для непрерывных сигналов не существует.

Многие методы ЦОС не требуют использования отрицательных частот или понимания периодичности ДПФ. Например, два часто используемых метода были описаны в предыдущей главе, *спектральный анализ* и *частотный отклик* систем. Для этих приложений вполне достаточно рассматривать временную область, как простирающуюся от *0* отсчета до *N-1*, а частотную область, как простирающуюся от нуля до одной второй частоты дискретизации. Эти методы могут использовать более простой взгляд на мир, поскольку они никогда не дают частей одного периода, перемещающихся в другой период. В таких случаях, рассматривание одного периода так же хорошо, как и рассматривание всего периодического сигнала.

Однако некоторые процедуры могут быть проанализированы *только* с учетом того, как сигналы перемещаются между периодами. Два примера этому также были уже представлены, *круговая свертка* и *аналого-цифровое преобразование*. В круговой свертке перемножение частотных спектров выражается в свертке сигналов временной области. Если результирующие сигналы временной области настолько длинны, что не умещаются в один период, они накладываются на соседние периоды, что приводит к *совмещению имен во временной области*. Напротив, аналого-цифровое преобразование является примером *совмещения имен в частотной области*. Во временной области осуществляется нелинейная операция, то есть посредством дискретизации непрерывный сигнал превращается в дискретный. Проблема заключается в том, что спектр исходного аналогового сигнала может быть настолько длинным, что не умещается внутри спектра дискретного сигнала. Когда мы вынуждаем такую ситуацию, концы спектра выходят в соседние периоды. Давайте рассмотрим еще два примера, где периодический характер ДПФ существенен, *сжатие* и *растяжение* сигналов и *амплитудная модуляция*.

Сжатие и растяжение, многоскоростные методы

Как показано на рис. 10.12, *сжатие* сигнала в одной области приводит к *растяжению* сигнала в другой области, и наоборот. Для непрерывных сигналов, если $X(f)$ является преобразованием Фурье от $x(t)$, тогда $1/k \times X(f/k)$ является преобразованием Фурье от $x(kt)$, где k является параметром, управляющим растяжением или сжатием. Если событие происходит *быстрее* (оно сжато во времени), оно должно содержать *более высокие* частоты. Если событие происходит *медленнее* (оно растянуто во времени), оно должно содержать *более низкие* частоты. Эта модель остается в силе, даже если ее довести до одной из двух крайностей. То есть, если сигнал временной области сжат настолько сильно, что он превратился в *импульс* (имеется ввиду предельный переход, когда ширина сигнала стремится к нулю – прим. перев.), соответствующий частотный спектр растягивается так сильно, что он становится *постоянной величиной*. Аналогично, если сигнал временной области растягивается до тех пор, пока он не станет постоянной величиной, соответствующий частотный спектр становится импульсом.

Дискретные сигналы ведут себя подобным же образом, но здесь имеется больше всяких подробностей. Первой проблемой с дискретными сигналами является *совмещение имен*. Представьте, что импульс, показанный на рис. 10.12a, сжат в несколько раз больше, чем это показано. Частотный спектр растянется в такое же число раз, а несколько горбов на рис. 10.12b переместятся в область частот за пределами 0,5. Полученное в результате совмещение имен разрушает простое соотношение сжатия/растяжения. Подобный тип совмещения имен может также произойти и во временной области. Представьте, что частотный спектр на рис. 10.12f сжимается значительно сильнее, приводя к расширению сигнала на рис. 10.12e в соседние периоды.

Вторая проблема, точно обусловить, что означает сжать или растянуть дискретный сигнал. Как показано на рис. 10.12a, дискретный сигнал сжимается за счет сжатия,

лежащей в его основе *непрерывной* кривой, на которой находятся отсчеты, и последующей дискретизации новой непрерывной кривой с целью нахождения нового дискретного сигнала. Аналогично, точно такой же процесс растяжения дискретных сигналов показан на рис. 10.12e. Когда происходит сжатие дискретного сигнала, события в сигнале (такие как ширина импульса) происходят за *меньшее* число отсчетов. Подобно этому, события в растянутом сигнале происходят за *большее* число отсчетов.

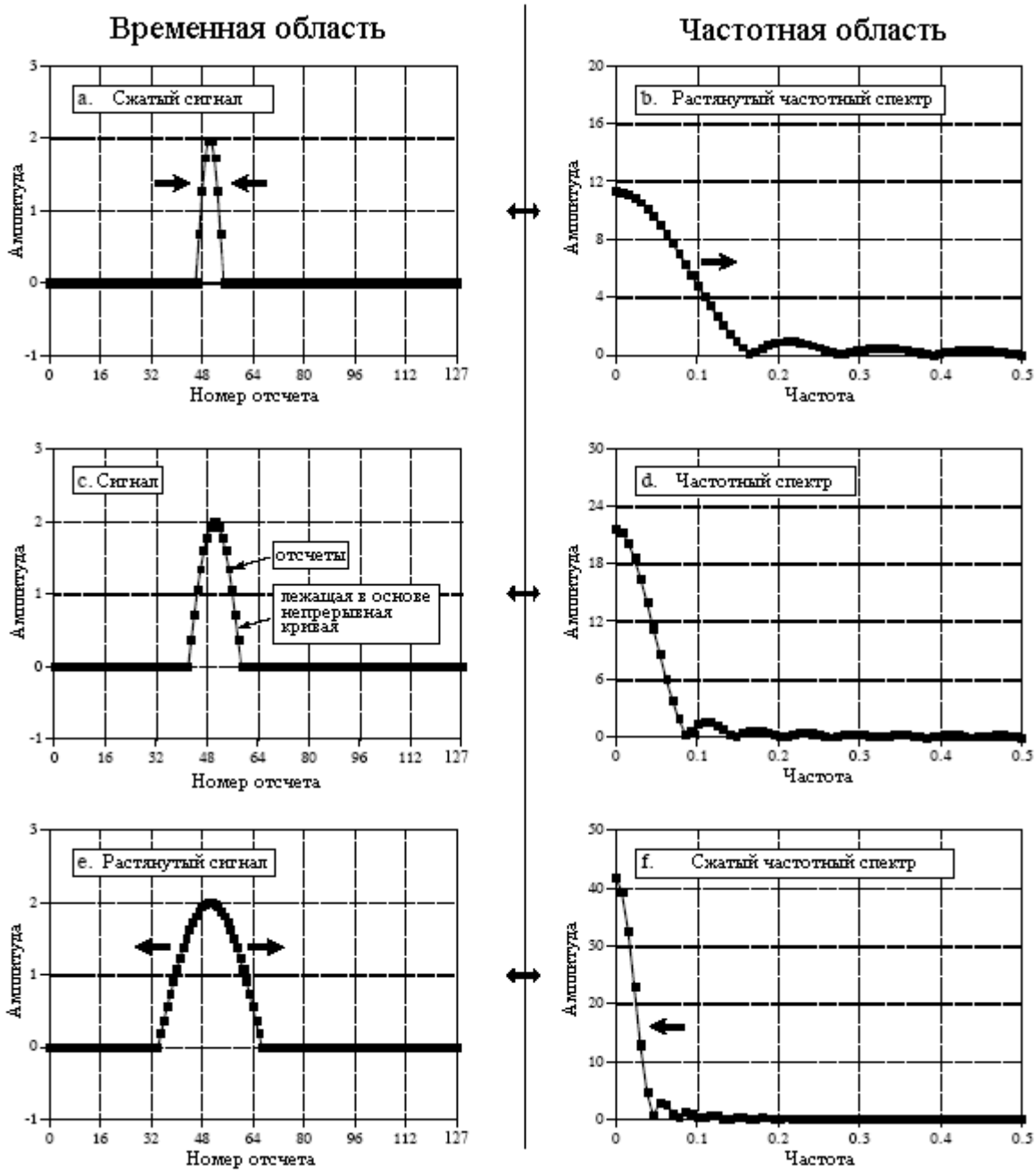


Рис. 10.12 Сжимание и растягивание

Эквивалентный способ рассмотрения этой процедуры заключается в том, чтобы сохранять лежащую в основе непрерывную кривую точно в том же самом виде, но осуществлять ее повторную дискретизацию при другой скорости. Например, взгляните на рис. 10.13a, дискретная Гауссова форма волны содержит 50 отсчетов. На рис. 10.13b та же самая лежащая в основе кривая представлена при помощи 400 отсчетов. Разница между кривыми на рис. 10.13a и рис. 10.13b может быть объяснена двумя способами: (1) скорость дискретизации сохранялась постоянной, но лежащая в основе форма волны была

расширена в восемь раз, или (2) лежащая в основе форма волны сохранялась постоянной, но скорость дискретизации была увеличена в восемь раз. Методы изменения скорости дискретизации таким способом называются **многоскоростными** методами. Если же добавляется большее количество отсчетов, они называются **интерполяционными**. Если для представления сигнала используется меньшее количество отсчетов, они называются **децимационными**. В Главе 3 описывается, как многоскоростные методы используются в АЦП и ЦАП.

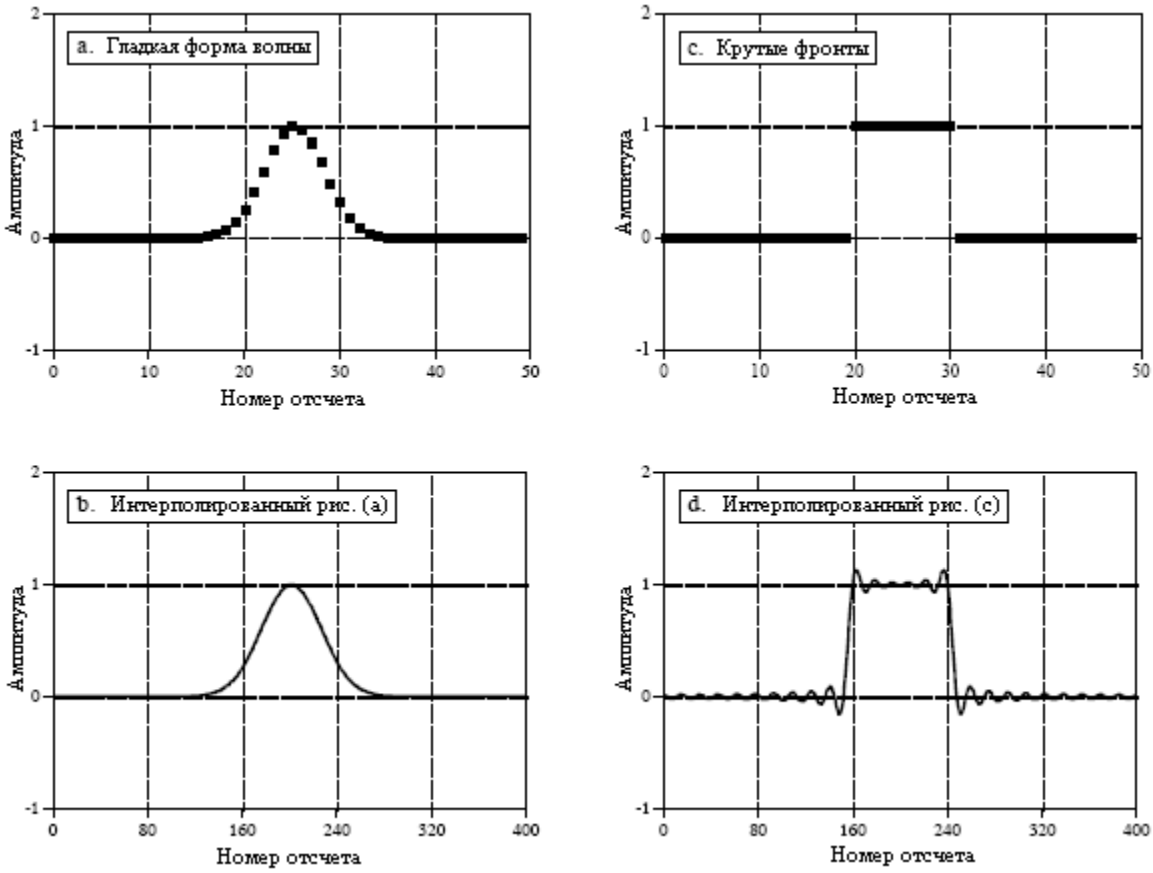


Рис. 10.13 Интерполяция дополнением частотной области

Существует, однако, следующая проблема: если нам задан произвольный дискретный сигнал то, как мы можем узнать, что представляет собой лежащая в его основе непрерывная кривая? Все зависит от того, во *временной области* или в *частотной области* закодирована информация сигнала. Для сигналов закодированных во временной области мы желаем, чтобы лежащая в основе непрерывная форма волны была гладкой кривой, проходящей через все отсчеты. В простейшем случае мы можем нарисовать между точками прямые линии, а затем скруглить грубые углы. Следующий уровень сложности - использовать алгоритм вычерчивания эмпирической кривой, подобный сплайн функциям или полиномиальному выравниванию. Единственного “правильного” ответа на эту проблему не существует. Данный подход основывается на минимизации нерегулярностей в форме волны *временной области* и полностью игнорирует частотную область.

Когда сигнал обладает информацией, закодированной в частотной области, мы игнорируем форму волны временной области и концентрируемся на частотном спектре. Как обсуждалось в предыдущей главе, наилучшая дискретность спектра (большее число отсчетов между частотами 0 и 0,5) может быть получена дополнением сигнала временной

области нулями, перед взятием ДПФ. Дуальность позволяет работать этому и в обратном направлении. Если мы хотим наилучшей дискретности во временной области (интерполяция), следует дополнить частотный спектр нулями перед взятием обратного ДПФ. Скажем к примеру, мы хотим провести интерполяцию сигнала, состоящего из 50 отсчетов, до сигнала, состоящего из 400 отсчетов. Это проделывается следующим образом. (1) Возьмите 50 отсчетов и добавьте нули для того, чтобы сделать сигнал длинной 64 отсчета. (2) Используйте 64 точечное ДПФ для нахождения частотного спектра, который будет состоять из 33 точек действительной части и 33 точек мнимой части. (3) Дополните правую часть частотного спектра (обе и действительную и мнимую части) 224 нулями для того, чтобы сделать частотный спектр длинной 257 точек. (4) Используйте 512 точечное обратное ДПФ для преобразования данных обратно во временную область. Это даст сигнал в 512 отсчетов, являющийся версией с высоким разрешением сигнала в 64 отсчета. Первые 400 отсчетов этого сигнала - это интерполированная версия исходных 50 отсчетов.

Ключевой особенностью этого метода является то, что интерполированный сигнал состоит из *точно* тех же частот, что и исходный сигнал. Это может обеспечить, а может и не обеспечить, хорошую подгонку во временной области. Например, на рис. 10.13a и рис. 10.13b показан сигнал в 50 отсчетов, интерполированный с помощью этого метода до сигнала в 400 отсчетов. Интерполированной является гладкая, вычерченная между исходными точками кривая, как если бы для этого использовалась графическая программа. Для сравнения, на рис. 10.13c и рис. 10.13d показан другой пример, где временная область представляется беспорядочной! На фронтах или других разрывах сигнала возникает колебательное поведение, показанное на рис. 10.13d. Сюда включаются также любые разрывы между нулевым и *N-1* отсчетами, поскольку временная область рассматривается как круговая. Такие выбросы (перерегулирование – прим. перев.) в точках разрыва называются *эффектом Гиббса* и обсуждаются в Главе 11. Другой метод интерполяции в частотной области представлен в Главе 3, добавление нулей между отсчетами временной области и низкочастотная фильтрация.

Умножение сигналов (Амплитудная модуляция)

Важным свойством преобразования Фурье является то, что *свертка* в одной области соответствует *умножению* в другой области. Одна сторона этого была обсуждена в предыдущей главе: свертка сигналов временной области может быть выполнена перемножением их частотных спектров. Амплитудная модуляция является примером обратной ситуации, перемножение сигналов во временной области соответствует их свертке в частотной области. Вдобавок амплитудная модуляция дает великолепный пример того, как иллюзорные *отрицательные* частоты входят в будничную науку и инженерные задачи.

Звуковые сигналы просто великолепны для связи на короткие расстояния: когда Вы разговариваете, то кто-то, находящийся в другом конце комнаты, Вас слышит. С другой стороны, радиочастоты очень хорошо распространяются на большие расстояния. Например, если синусоидальная волна напряжением 100 вольт и частотой 1 МГц подается на антенну, полученная радиоволна может быть обнаружена в другой *комнате*, другой *стране* и даже на другой *планете*. **Модуляция** является процессом объединения двух этих сигналов для формирования третьего сигнала с желаемыми характеристиками объединяемых сигналов. Вы не можете просто взять и сложить два сигнала; модуляция всегда включает нелинейные процедуры подобные умножению. В радиосвязи модуляция дает радиосигналы, которые могут распространяться на огромные расстояния *и* нести звуковую или другую информацию.

Радиосвязь это исключительно хорошо развитая отрасль знаний, в которой разработано большое количество схем модуляции. Одна из наиболее простейших

называется **амплитудной модуляцией**. На рис. 10.14 приведен пример того, как выглядит амплитудная модуляция в обеих временной и частотной областях. В этом примере будут использоваться непрерывные сигналы, поскольку модуляция обычно реализуется аналоговой электроникой. Однако если нужно, вся процедура может быть реализована в дискретной форме (форме будущего!).

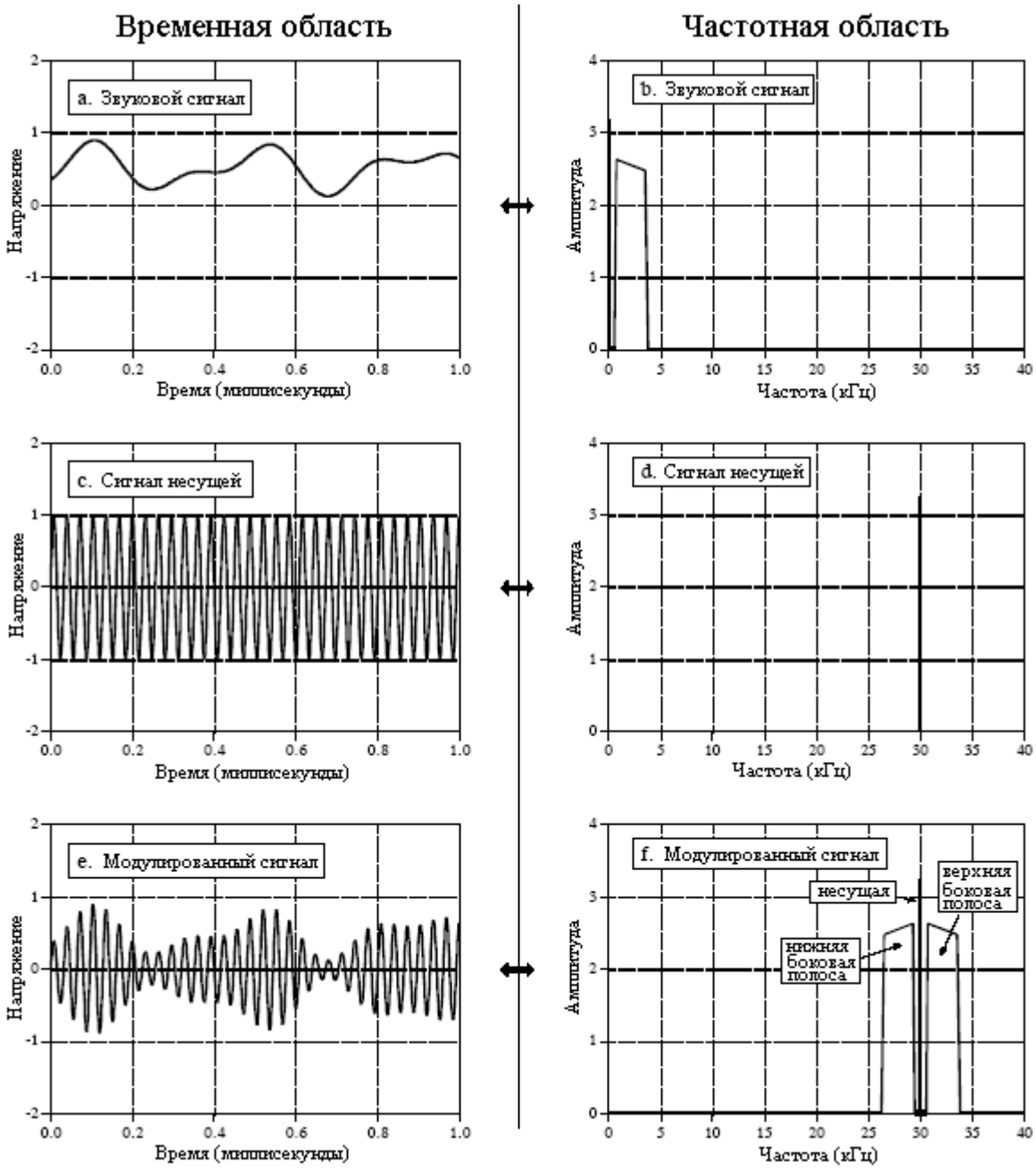


Рис. 10.14 Амплитудная модуляция

На рис. 10.14a показан звуковой сигнал с постоянной составляющей, смещенной таким образом, что сигнал всегда имеет положительную величину. На рис. 10.14b показано, что его частотный спектр состоит из частот от 300 Гц до 3 кГц, полоса необходимая для голосовой связи, плюс спектральная линия постоянной составляющей. Все другие частоты с помощью аналоговой фильтрации были удалены. На рис. 10.14c и рис. 10.14d показана **несущая волна**, чистая синусоида со значительно более высокой частотой, чем звуковой сигнал. Во временной области, амплитудная модуляция заключается в *умножении* звукового сигнала на несущую волну. Как показано на рис.

10.14e, это дает форму волны с изменяющейся амплитудой, мгновенное значение которой пропорционально амплитуде исходного звукового сигнала. На профессиональном жаргоне этой области, *огибающая* несущей эквивалентна модулирующему сигналу. Этот сигнал может быть направлен к антенне, преобразован в радиоволну, и затем обнаружен приемной антенной. Электроника радиоприемника выработает сигнал идентичный показанному на рис. 10.14e. Затем, для преобразования приведенной на рис. 10.14e формы волны, обратно в форму волны, показанную на рис. 10.14a, будет использован *детектор* или *демодулирующая* схема.

Поскольку сигналы временной области перемножаются, происходит свертка соответствующих частотных спектров. То есть, спектр на рис. 10.14f находится при помощи свертки спектров на рис. 10.14b и рис. 10.14d. Поскольку спектром несущей является сдвинутая дельта функция, спектр модулированного сигнала эквивалентен звуковому спектру, *сдвинутому* к частоте несущей. Это дает модулированный спектр, состоящий из трех компонент: **несущего колебания**, **верхней боковой полосы** и **нижней боковой полосы**.

Они соответствуют трем частям исходного звукового сигнала: постоянной составляющей, положительным частотам между 0,3 и 3 кГц и отрицательным частотам между –0,3 и –3 кГц, соответственно. Даже притом, что отрицательные частоты в исходном звуковом сигнале являются чем-то абстрактным и иллюзорным, результирующие частоты в нижней боковой полосе настолько же реальны, насколько Вы могли захотеть, чтобы это было так. Привидения приняли человеческий облик!

Инженеры связисты живут и умирают с данным видом анализа частотной области. В частности, рассмотрим частотный спектр телевизионной передачи. Стандартный телевизионный сигнал имеет частотный спектр от постоянной составляющей до 6 МГц (здесь и далее речь идет об американских телевизионных стандартах отличающихся от отечественных, так, например, в соответствии с отечественными стандартами телевизионный сигнал имеет полосу шириной 8 МГц – прим. перев.). С помощью такого метода частотного сдвига, концом к концу, уложены 82 телевизионных канала шириной по 6 МГц. Например, канал 3 размещается от 60 МГц до 66 МГц, канал 4 размещается от 66 МГц до 72 МГц, канал 83 размещается от 884 МГц до 990 МГц и т.д. Для отображения передачи на экране телевизионный приемник передвигает желаемый канал назад к диапазону от постоянной составляющей до 6 МГц. Такая схема размещения каналов называется **частотным мультиплексированием**.

Дискретно-временное преобразование Фурье

Дискретно-временное преобразование Фурье (ДВПФ) является членом семейства преобразований Фурье, которое работает с *апериодическими дискретными* сигналами. Лучший способ понять ДВПФ - это узнать, как оно соотносится с ДПФ. Для начала представьте, что Вы снимаете сигнал, содержащий N отсчетов, и хотите найти его частотный спектр. При использовании ДПФ, сигнал может быть разложен на $N/2+1$ синусных и косинусных волн с равномерно расположенными между нулем и одной второй от частоты дискретизации частотами. Как обсуждалось в предыдущей главе, дополнение сигнала временной области нулями делает период временной области *длиннее*, так же, как и интервалы между отсчетами в частотной области *уже*. Когда N приближается к бесконечности, временная область становится *апериодической*, а частотная область становится *непрерывным* сигналом. Это и есть ДВПФ, преобразование Фурье, которое соотносит *апериодический дискретный* сигнал с *периодическим непрерывным* частотным спектром.

Понять математику ДВПФ можно, начав с уравнений синтеза и анализа для ДПФ (уравнения (8.2), (8.3) и (8.4)), устремив N к бесконечности:

$$\operatorname{Re} X(\omega) = \sum_{n=-\infty}^{\infty} x[n]\cos(\omega n)$$

$$\operatorname{Im} X(\omega) = -\sum_{n=-\infty}^{\infty} x[n]\sin(\omega n) \qquad (10.1)$$

$$x[n] = \frac{1}{\pi}\int_{0}^{\pi} \operatorname{Re} X(\omega)\cos(\omega n) - \operatorname{Im} X(\omega)\sin(\omega n)\,d\omega \qquad (10.2)$$

В этих соотношениях много тонких деталей. Во-первых, сигнал временной области $x[n]$ все еще дискретный, и поэтому заключен в *квадратные скобки*. Для сравнения, сигналы *Re* $X(\omega)$ и *Im* $X(\omega)$ являются непрерывными и, таким образом, написаны с круглыми скобками. Поскольку частотная область является непрерывной, уравнение синтеза должно быть записано в виде интеграла, а не в виде суммы.

Как обсуждалось в Главе 8, частоты в частотной области ДПФ представляются при помощи одной из трех переменных: *k* – индекс, изменяющийся от *0* до *N/2*; *f* – доля частоты дискретизации, изменяющаяся от *0* до *0,5*, или ω – доля частоты дискретизации, выраженная в виде естественной частоты, изменяющаяся от *0* до π. Спектр ДВПФ непрерывен, так что может использоваться либо *f* либо ω. Наиболее частым выбором является ω, поскольку тогда уравнения становятся короче, за счет сокращения всегда присутствующего коэффициента 2π. Запомните, что когда используется ω, частотный спектр простирается от *0* до π, что соответствует диапазону от постоянной составляющей до половины частоты дискретизации. Чтобы усложнить себе жизнь еще больше, для представления этой частоты, некоторые авторы вместо ω (строчная буква омега) используют Ω (прописная буква омега).

При вычислении обратного ДПФ, прежде чем может быть выполнен синтез (уравнение (8.2)), отсчеты с номером *0* и *N/2* должны быть разделены на два (уравнение (8.3)). При ДВПФ в этом нет необходимости. Как Вы помните, это действие в ДПФ относится к частотному спектру, который определяется как *спектральная плотность*, т.е. амплитуда в единице полосы частот. Когда спектр становится непрерывным, специальное соглашение относительно крайних точек исчезает. Однако нормализующий коэффициент, который должен быть включен, все еще остается, *2/N* в ДПФ (уравнение (8.3)) и $1/\pi$ в ДВПФ (уравнение (10.2)). Некоторые авторы размещают эти члены перед уравнением *синтеза*, в то время как другие размещают их перед уравнением *анализа*. Предположим, Вы начинаете с некоторого сигнала временной области. После взятия преобразования Фурье, а затем обратного преобразования Фурье, Вы хотите закончить тем, с чего Вы начали. То есть, где-то по дороге, либо в прямом, либо в обратном преобразовании, должен быть учтен член $1/\pi$ (или член *2/N*). Некоторые авторы даже дробят этот член между двумя преобразованиями, помещая перед обоими $1/\sqrt{\pi}$.

Поскольку ДВПФ включает в себя бесконечные суммирования и интегралы, оно не может быть вычислено с помощью цифровых компьютеров. Его основное применение лежит в теоретических проблемах в качестве альтернативы ДПФ. Например, предположим, Вы хотите найти частотный отклик системы из ее импульсного отклика. Если импульсный отклик задан в виде *множества чисел*, которые могли бы быть получены из экспериментальных измерений или компьютерного моделирования, тогда на компьютере прогоняется программа ДПФ. Это дает частотный спектр в виде другого *множества чисел* равномерно расположенных между *0* и *0,5* от частоты дискретизации.

В других случаях, импульсный отклик может быть задан в виде *уравнения*, такого как синк функция (описана в следующей главе), или экспоненциально затухающая синусоида. Здесь, для математического вычисления частотной области в виде другого *уравнения*, определяющего всю непрерывную область между *0* и *0,5*, используется ДВПФ. Хотя для выполнения этих вычислений могло бы быть тоже использовано ДПФ, оно бы только дало уравнение для *отсчетов* частотной области, а не для всей кривой.

Равенство Парсеваля

Поскольку временная и частотная области являются эквивалентными представлениями одного и того же сигнала, они должны иметь одну и ту же *энергию*. Это соотношение, справедливое для всех членов семейства преобразований Фурье, называется равенством Парсеваля (иногда равенство Парсеваля называют теоремой Релея – прим. перев.). Для ДПФ равенство Парсеваля выражается следующим образом (для чисто активной единичной нагрузки – прим. перев.):

$$\sum_{i=0}^{N-1} x[i]^2 = \frac{2}{N} \sum_{k=0}^{N/2} MagX[k]^2 \,. \qquad (10.3)$$

Левая сторона этого уравнения является полной энергией, содержащейся в сигнале временной области, определяемой суммированием энергий *N* индивидуальных отсчетов. Подобно этому, правая сторона является энергией, содержащейся в частотной области, определяемой суммированием энергий *N/2+1* синусоид. Помните из физики, что *энергия* пропорциональна *квадрату амплитуды*. Например, энергия пружины пропорциональна квадрату деформации, а энергия, накопленная в конденсаторе, пропорциональна квадрату напряжения. В уравнении (10.3) *X*[*k*], с одной небольшой модификацией, является частотным спектром *x*[*i*]: первая и последняя частотные составляющие, *X*[*0*] и *X*[*N/2*], были разделены на $\sqrt{2}$. Эта модификация наряду с множителем *2/N* с правой стороны уравнения объясняет несколько тонких моментов вычисления и суммирования энергий.

Для того чтобы понять эти коррекции, начните с поиска представления сигнала в частотной области при помощи ДПФ. Затем, как предварительно определено в уравнении (8.3), преобразуйте частотную область в амплитуды синусоид, необходимые для восстановления сигнала. Это выполняется делением первой и последней точек (отсчеты *0* и *N/2*) на *2*, а затем делением всех точек на *N/2*. Хотя это и дает амплитуды синусоид, они выражены в виде *максимальных* амплитуд, а не в виде необходимых для вычисления энергии *среднеквадратичных* (действующих – прим. перев.) значений. Для синусоид, максимальная амплитуда преобразуется в среднеквадратичное значение делением на $\sqrt{2}$. Такая коррекция должна быть проведена со всеми величинами частотной области, кроме отсчетов *0* и *N/2*. Это связано с тем, что эти две синусоиды уникальны; одна является постоянной величиной, в то время как другая скачет между двумя постоянными значениями. Для этих двух особых случаев *максимальная* амплитуда уже равна *среднеквадратичному* значению. Все величины в частотной области возводятся в квадрат и затем суммируются. Последним шагом является деление суммарной величины на *N*, чтобы учесть каждый отсчет в частотной области, преобразованный в синусоиду, охватывающую *N* значений во временной области. Проработка всех этих деталей и даст уравнение (10.3).

Хотя равенство Парсеваля является интересным в отношении физики, которую оно описывает (закон сохранения энергии), оно имеет ограниченное практическое использование в ЦОС.

ГЛАВА
11

ПАРЫ ПРЕОБРАЗОВАНИЯ ФУРЬЕ

Для каждой формы волны *временной области* существует соответствующая форма волны *частотной области* и наоборот. Например, прямоугольному импульсу во временной области, в частотной области соответствует синк функция (т.е. $sin(x)/x$). В следствии дуальности справедливо также и обратное; прямоугольному импульсу в частотной области, во временной области соответствует синк функция. Формы волн соответствующие друг другу в такой манере называются *парами преобразования Фурье*. В этой главе представлено несколько типичных пар.

Пары дельта функции

Для дискретных сигналов дельта функция является простейшей формой волны и имеет одинаково простую пару преобразования Фурье. На рис. 11.1a показана дельта функция во временной области с ее частотным спектром, приведенным на рис. 11.1b и рис. 11.1c. Ее модуль является постоянной величиной, в то время как фаза везде равна нулю. Как обсуждалось в предыдущей главе, это можно понять с помощью свойства растяжения/сжатия. Когда временная область сжимается до тех пор, пока она не станет импульсом бесконечно малой длительности, частотная область растягивается до тех пор, пока она не станет постоянной величиной.

На рис. 11.1d и рис. 11.1g форма волны временной области сдвинута, соответственно, на четыре и на восемь отсчетов вправо. Как следует из свойств, обсужденных в предыдущей главе, сдвиг формы волны во временной области не действует на модуль, но добавляет линейный компонент к фазе. На этом рисунке сигналы фазы не были *развернуты* и, следовательно, располагаются только на интервале от $-\pi$ до π. Заметьте также, что горизонтальные оси в частотной области изменяются от *–0,5* до *0,5*. То есть, они показывают как *отрицательные* частоты в спектре, так и *положительные*. Отрицательные частоты представляют собой избыточную информацию, но их довольно часто включают в графики ЦОС, и Вам следует привыкнуть к их созерцанию.

На рис. 11.2 представлена та же самая информация, что и на рис. 11.1, но частотная область представлена здесь в *прямоугольной форме*. Тут есть два наставления, которые следовало бы изучить. Во-первых, сравним полярное и прямоугольное представления частотной области. В наиболее типичном случае полярную форму намного легче понять; модуль является ничем иным, как постоянной, в то время как фаза является прямой линией. Для сравнения, действительная и мнимая части являются синусоидальными колебаниями, которым трудно придать какой-то смысл.

Второй интересной чертой на рис. 11.2 является *дуальность* ДПФ. В обычном представлении каждый отсчет в частотной области ДПФ соответствует синусоиде во временной области. Однако, обратное этому также справедливо, каждый отсчет во временной области соответствует синусоидам в частотной области. Включение отрицательных частот в эти графики позволяет свойству дуальности быть более симметричным. Например, рис. 11.2d, рис. 11.2e и рис. 11.2f показывают, что импульс, приходящийся на отсчет номер четыре во временной области, дает четыре периода коси-

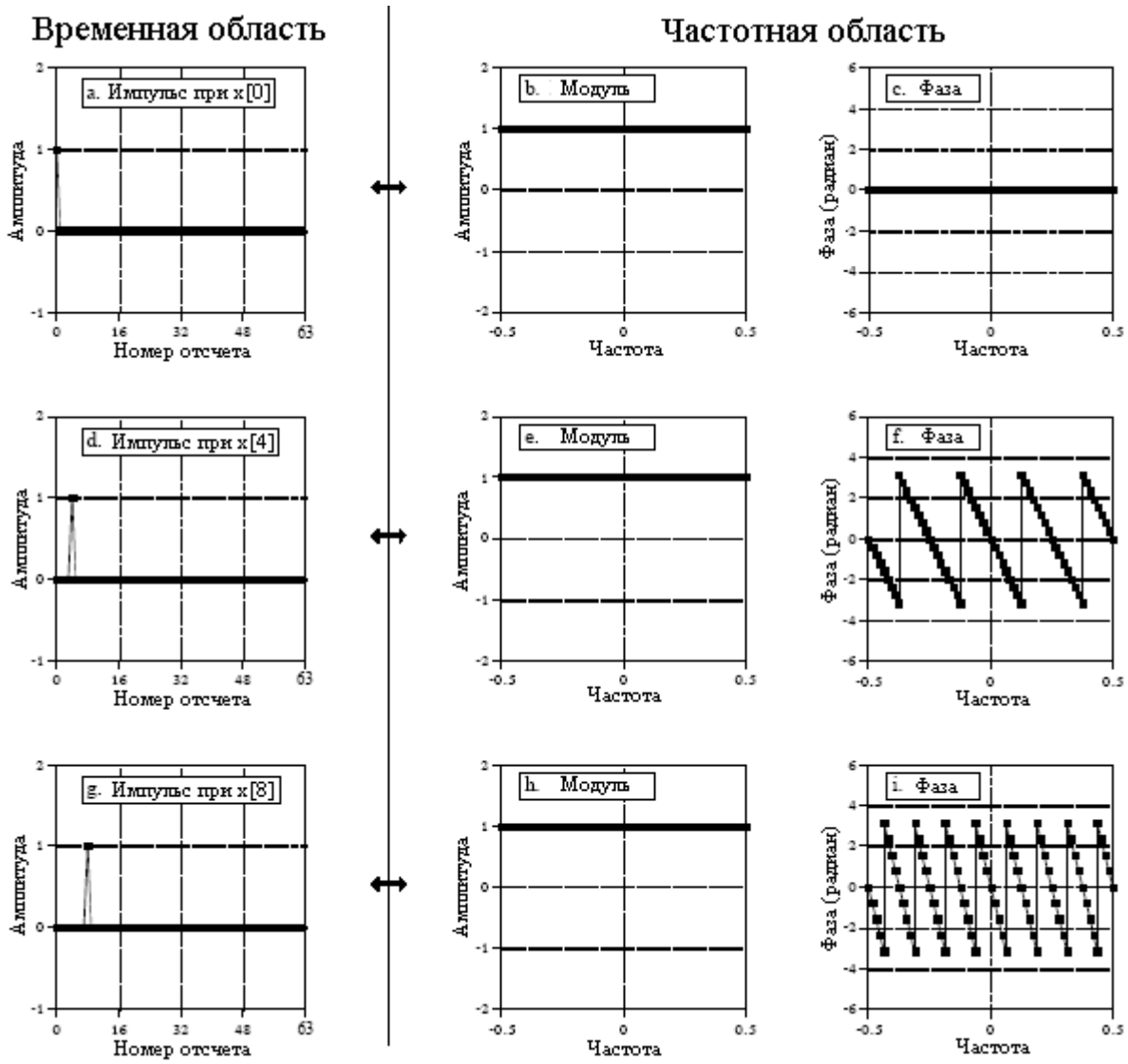


Рис. 11.1 Пары дельта функции в полярной форме

нусоидальной волны в действительной части частотного спектра и четыре периода отрицательной синусоидальной волны в мнимой части. Как Вы помните, импульс, приходящийся на отсчет с номером четыре в действительной части частотного спектра, дает четыре периода косинусоидальной волны во временной области. Аналогично, импульс, приходящийся на отсчет с номером четыре в мнимой части частотного спектра, дает четыре периода отрицательной синусоидальной волны, складываемой с волной временной области.

Как отмечалось в Главе 8, это может быть использовано в качестве другого способа вычисления ДПФ (помимо коррелирования временной области с синусоидами). Каждый отсчет временной области дает косинусоидальную волну, прибавляемую к действительной части частотной области, и отрицательную синусоидальную волну, прибавляемую к мнимой части. *Амплитуда* каждой синусоиды задается *амплитудой* отсчета временной области. *Частота* каждой синусоиды определяется порядковым *номером отсчета* точки временной области. Алгоритм включает: (1) проход через каждый отсчет временной области, (2) вычисление синусоидальной и косинусоидальной волн, соответствующих каждому отсчету и (3) сложение всех вносящих свой вклад синусоид. Конечная программа почти идентична методу корреляции (таблица 8.2), за исключением того, что внешние и внутренние циклы меняются местами.

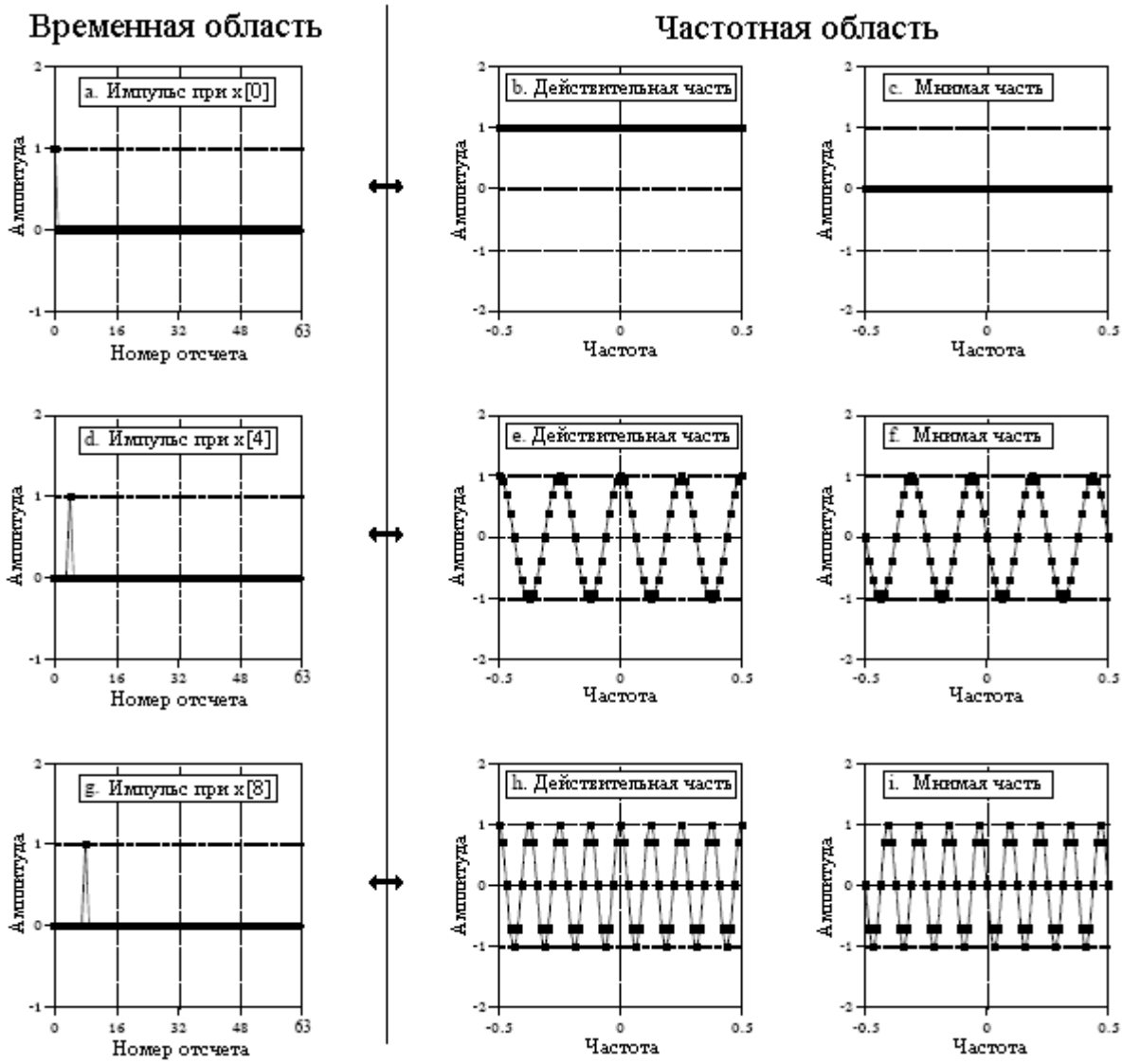


Рис. 11.2 Пары дельта функции в прямоугольной форме

Синк функция

Рис. 11.3 иллюстрирует часто встречающуюся пару преобразования: *прямоугольный импульс* и *sinc функцию* (произносится как “синк”). Синк функция определяется как $sinc(a)=sin(\pi a)/(\pi a)$, однако, часто можно встретить смутное утверждение: “синк функция является общей формой: $sin(x)/x$.” Другими словами, синк является синусоидальной волной затухающей по амплитуде как $1/x$. На рис. 11.3a прямоугольный импульс симметрично центрирован относительно нулевого отсчета, образуя на графике половину импульса справа и другую половину слева. Из-за периодичности временной области он рассматривается ДПФ как единый импульс. ДПФ этого сигнала показано на рис. 11.3b и 11.3c, на рис. 11.3d и рис. 11.3e показана *развернутая* версия.

Сначала посмотрите на развернутый спектр, рис. 11.3d и рис. 11.3e. *Развернутый модуль* представляет собой колебания, уменьшающиеся по амплитуде с увеличением частоты. Фаза состоит везде из нулей, как Вы и могли предполагать для сигнала временной области симметричного относительно нулевого отсчета. Мы используем термин *развернутый модуль*, чтобы подчеркнуть, что он может иметь как положительные,

так и отрицательные значения. Согласно определению *модуль* должен быть всегда положительным. Это показано на рис. 11.3b и рис. 11.3c, где модуль сделан везде положительным за счет введения фазового сдвига величиной π на всех частотах, где развернутый модуль на рис. 11.3d отрицателен.

На рис. 11.3f сигнал сдвинут таким образом, что он рассматривается как один непрерывный импульс, но он уже больше не центрирован относительно нулевого отсчета. Хотя это не изменяет модуль частотной области, это добавляет линейную компоненту к фазе, превращая ее в сумбурный ералаш. А как выглядит частотный спектр действительной и мнимой частей? Слишком запутанно, чтобы даже думать об этом.

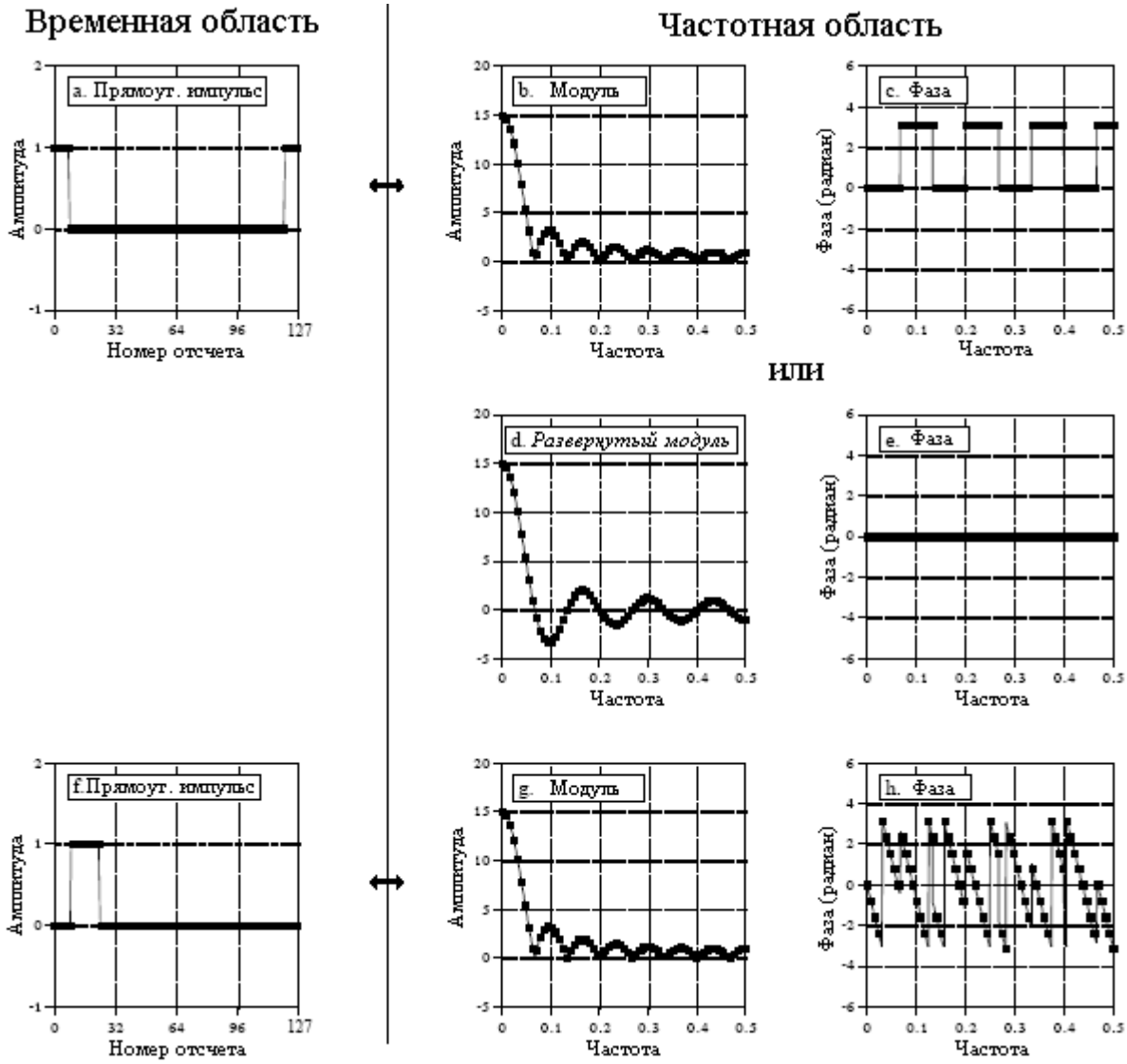


Рис. 11.3 ДПФ от прямоугольного импульса

N точечный сигнал временной области, содержащий прямоугольный импульс единичной амплитуды шириной *M* точек, имеет частотный спектр ДПФ, задаваемый как:

$$MagX[k] = \left| \frac{\sin\left(\pi k M / N\right)}{\sin \pi k / N} \right|. \qquad (11.1)$$

(Для того чтобы избежать деления на *0*, используйте $X[0]=M$.)

В качестве альтернативы для выражения частотного спектра, как части частоты дискретизации *f*, может использоваться ДВПФ:

$$MagX[k] = \left| \frac{\sin(\pi f M)}{\sin(\pi f)} \right|. \qquad (11.2)$$

(Для того чтобы избежать деления на *0*, используйте *Mag X*(*0*)=*M*.)

Другими словами, уравнение (11.1) дает *N/2+1 отсчетов* в частотном спектре, в то время как уравнение (11.2) дает *непрерывную кривую*, на которой лежат отсчеты. Данные уравнения определяют только модуль. Фаза определяется исключительно при помощи позиционирования формы волны временной области слева направо, как обсуждалось в предыдущей главе.

Заметьте на рис. 11.3b, что до достижения частоты *0,5*, амплитуда колебания до нуля не затухает. Как Вам и следует предполагать, форма волны продолжается в следующий период, где происходит *совмещение имен*. Это изменяет конфигурацию частотной области, эффект который включен в уравнения (11.1) и (11.2).

Очень часто важно понять, как выглядит частотный спектр, когда совмещение имен *отсутствует*. Это связано с тем, что *дискретные сигналы* очень часто используются для представления или моделирования *непрерывных сигналов*, а в непрерывных сигналах совмещения имен не происходит. Для устранения совмещения имен в уравнениях (11.1) и (11.2) производят замену знаменателя с *sin*(*πk*/*N*) на *πk*/*N* и с *sin*(*πf*) на *πf*, соответственно. Значительность этого показана на рис. 11.4. Количественное значение *πf* может изменяться только от 0 до 1,5708, поскольку *f* может изменяться от 0 до 0,5. В пределах этого диапазона большой разницы между *sin*(*πf*) и *πf* нет. На нулевой частоте они имеют одно и то же значение, а на частоте 0,5 разница составляет всего около 36%. При отсутствии совмещения имен кривая на рис. 11.3b показала бы слегка более низкую амплитуду около правой стороны графика, и никакого изменения около левой стороны.

Когда в частотном спектре прямоугольного импульса *нет* совмещения имен (вследствие непрерывного сигнала временной области, или потому, что Вы игнорируете совмещение имен), он имеет общую форму: *sin*(*x*)/*x*, т.е. синк функции. Для непрерывных сигналов *прямоугольный импульс* и *синк функция* являются парами преобразования Фурье. Для дискретных сигналов это лишь приближение с возникающей из-за совмещения имен погрешностью.

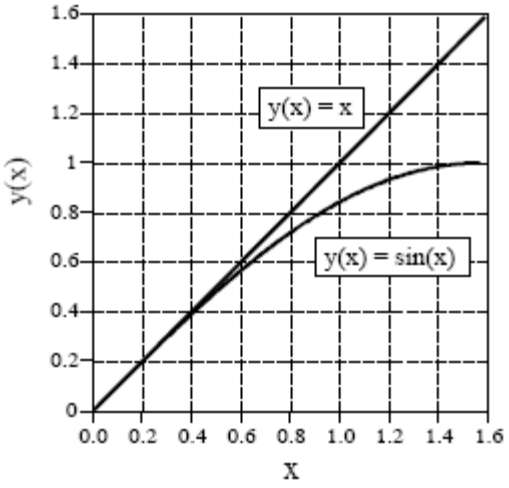


Рис. 11.4 Сравнение *x* и *sin*(*x*)

При $x=0$, синк функция обладает досадной проблемой, здесь $sin(x)/x$ становится *нулем*, деленным на *ноль*. Это не является трудной математической задачей; когда x становится очень маленьким, $sin(x)$ приближается к величине x (см. рис. 11.4). Это превращает синк функцию в x/x, имеющее значение *единицы*. Другими словами, когда x становится все меньше и меньше, значение $sinc(x)$ приближается к *единице*, что включает $sinc(0)=1$. Теперь попытайтесь объяснить это вашему компьютеру! Все, что он видит это деление на ноль, что вызывает его недовольство и останов Вашей программы. Важным моментом, который следует помнить, является то, что Ваша программа при вычислении синк функции должна включать специальную обработку для $x=0$.

Ключевой чертой синк функции является местоположение **пересечений нуля**. Они происходят на частотах, у которых целое число периодов синусоиды точно подогнано под прямоугольный импульс. Например, если ширина прямоугольного импульса 20 точек, первый ноль в частотной области находится на частоте, которая совершает один полный период за 20 точек. Второй ноль находится на частоте, которая совершает два полных периода за 20 точек и т.д. Это можно понять, если вспомнить, как вычисляется ДПФ при помощи корреляции. Амплитуда частотной составляющей находится умножением сигнала временной области на синусоиду и суммированием результирующих отсчетов. Если форма волны временной области является прямоугольным импульсом *единичной* амплитуды, то это то же самое, что и просто *сложение* отсчетов синусоид, находящихся в пределах прямоугольного импульса. Если это суммирование выполняется на *целом числе периодов синусоиды*, то результат будет нулевым.

Синк функция широко используется в ЦОС, поскольку она является парой преобразования Фурье для очень простой формы волны - прямоугольного импульса. Например, синк функция используется в *спектральном анализе*, обсужденном в Главе 9. Рассмотрим анализ бесконечно длинного дискретного сигнала. Поскольку ДПФ может работать только с сигналами *конечной* длины, для представления более длинного сигнала выбираются N отсчетов. Ключом здесь является то, что “выбор N отсчетов из более длинного сигнала” - это то же самое, что умножить более длинный сигнал на прямоугольный импульс. *Единицы* в прямоугольном импульсе сохраняют соответствующие отсчеты, в то время как *нули* ликвидируют их. А как это влияет на частотный спектр сигнала? Умножение сигнала временной области на прямоугольный импульс приводит к его *свертке* с *синк функцией* в частотной области. Это, как было показано ранее на рис. 9.5a, снижает разрешение частотного спектра.

Другие пары преобразования

На рис. 115a и рис. 11.5b показана дуальность выше сказанного: прямоугольному импульсу в частотной области соответствует синк функция (плюс совмещение имен) во временной области. Сигнал временной области, включая эффект совмещения имен, задается как:

$$x[i] = \frac{1}{N} \frac{\sin\left(2\pi i \frac{\left(M - 1/2\right)}{N}\right)}{\sin\left(\pi i / N\right)}. \qquad (11.3)$$

(Для того чтобы избежать деления на 0, используйте $x[0]=(2M-1)/N$.)

Для устранения из этого уравнения эффекта совмещения имен, представьте, что дискретность частотной области настолько маленькая, что она превращается в непрерывную кривую. Это делает временную область бесконечно длинной и

непериодической. ДВПФ является тем преобразованием Фурье, которое здесь надо использовать, что даст в результате сигнал временной области, описываемый следующим соотношением:

$$x[i] = \frac{\sin(2\pi f_c i)}{i\pi}. \qquad (11.4)$$

(Для того чтобы избежать деления на *0*, используйте $x[0]=2f_c$, где f_c - частота среза.)

Это уравнение очень важно в ЦОС, поскольку прямоугольный импульс в частотной области является идеальным *низкочастотным фильтром*. Следовательно, синк функция, описываемая этим уравнением, является ядром идеального низкочастотного фильтра. Это фундамент для очень полезного класса цифровых фильтров, называемых *фильтрами с ограниченной окном синк функцией*, описанными в Главе 16.

На рис. 11.5c и рис. 11.5d показано, что треугольный во временной области импульс совпадает с возведенной в *квадрат* синк функцией в частотной области (плюс совмещение имен). Эта пара преобразования не так важна, как *причина,* по которой это верно. Треугольник, содержащий *2M-1* точек во временной области, может быть сформирован при помощи свертки *M* точечного прямоугольного импульса с самим собой. Поскольку свертка во временной области дает умножение в частотной области, свертка формы волны с самой собой возведет в *квадрат* частотный спектр.

Существует ли форма волны, которая является своим собственным преобразованием Фурье? Ответом будет - да, и *только одна*: Гауссиан. На рис. 11.5e показана Гауссова кривая, а на рис. 11.5f показан соответствующий частотный спектр, также Гауссова кривая. Это соотношение справедливо только в случае, если Вы будете игнорировать совмещение имен. Соотношение между стандартным отклонением во временной области и в частотной области определяется как: $2\pi\sigma_f = 1/\sigma_f$. Хотя на рис. 11.5f показана только одна сторона Гауссиана, полную кривую с центром симметрии на нулевой частоте завершают отрицательные частоты.

На рис. 11.5g показано то, что может быть названо **Гауссовым пакетом**. Он формируется умножением синусоидальной волны на Гауссиан. Например, кривая на рис. 11.5g является синусоидальной волной, умноженной на такой же, как на рис. 11.5e Гауссиан. Соответствующая частотная область является Гауссианом с центром не на нулевой частоте, а где-нибудь в другом месте. Как и прежде, эта пара преобразования не столь важна как *причина*, по которой это верно. Поскольку сигнал временной области является перемножением двух сигналов, частотная область будет сверткой двух частотных спектров. Частотным спектром синусоидальной волны является дельта функция с центром на частоте синусоидальной волны. Частотным спектром Гауссиана является Гауссиан с центром на нулевой частоте. Свертка обоих даст Гауссиан с центром на частоте синусоидальной волны. Это должно показаться знакомым; это идентично процедуре *амплитудной модуляции*, описанной в предыдущей главе.

Эффект Гиббса

На рис. 11.6 показан сигнал временной области, синтезируемый из синусоид. На последнем графике рис. 11.6h показан восстанавливаемый сигнал. Поскольку этот сигнал длинной 1024 точки, то для полного восстановления понадобится 513 индивидуальных частот. На рисунках с 11.6a по 11.6g показано, как выглядят восстановленные сигналы, если используются только *некоторые* из этих частот. Например, на рис. 11.6f показан сигнал, восстановленный при использовании частот с *0* по *100*. Этот сигнал был создан взятием ДПФ от сигнала на рис. 11.6h при установке значений частот со 101 по 512 в

ноль, а затем использованием обратного ДПФ, для нахождения результирующего сигнала временной области.

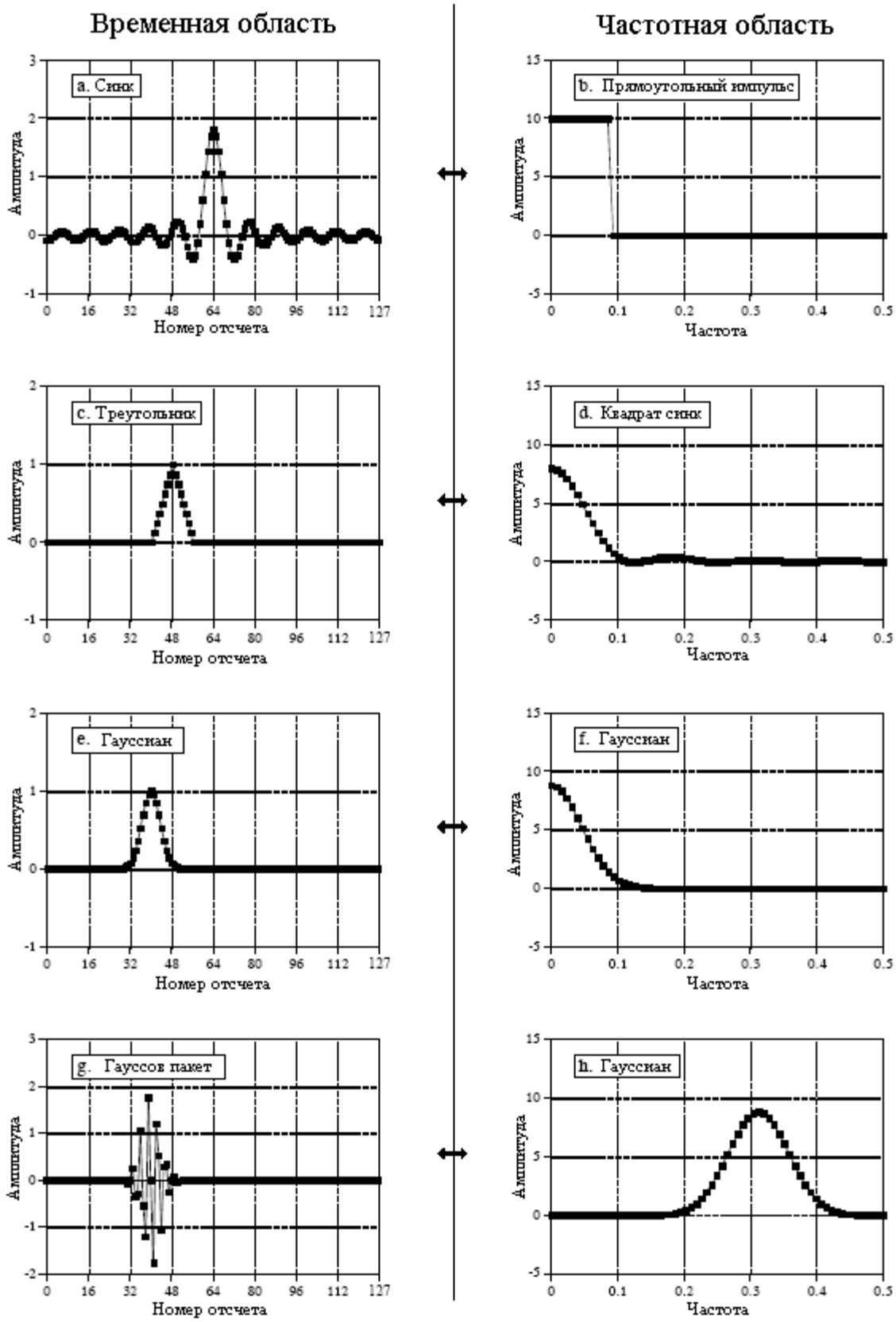


Рис. 11.5 Типичные пары преобразования

Чем больше суммируется частот при восстановлении, тем ближе становится сигнал к конечному результату. Интересным является то, *как* происходит приближение к конечному результату на *фронтах* сигнала. На рис. 11.6h есть три крутых фронта. Два из них являются фронтами прямоугольного импульса. Третий, поскольку ДПФ рассматривает временную область как периодическую, расположен между фронтами 1023 и 0. Когда при восстановлении используются только некоторые из частот, каждый фронт демонстрирует *выбросы* и *звон* (затухающие колебания). Такие выбросы и звон известны как **эффект Гиббса**, по имени математического физика Джосиаха Гиббса, объяснившего это явление в 1899.

Посмотрите внимательно на выброс на рис. 11.6e, рис. 11.6f и рис. 11.6g. По мере добавления синусоид *ширина* выбросов уменьшается; однако, *амплитуда* выбросов остается приблизительно такой же, грубо 9 процентов. Для дискретных сигналов это не является проблемой; выбросы исчезают, когда добавляется последняя частота. Однако, восстановление непрерывных сигналов не может быть объяснено так легко. Для синтеза непрерывного сигнала должно быть сложено бесконечное число синусоид. Проблема состоит в том, что по мере приближения числа синусоид к бесконечности, амплитуда выбросов не уменьшается, она остается почти такой же, те же 9%. Учитывая эту ситуацию (и другие аргументы), резонно задать вопрос *может* ли суммирование непрерывных синусоид восстановить фронт. Помните ссору между Лагранжем и Фурье?

Критическим фактором в разрешении этой головоломки является то, что чем больше вводится синусоид, тем меньше становится *ширина* выбросов. Даже при бесконечном числе синусоид выбросы все еще присутствуют, но их ширина равна *нулю*. Точно в точке разрыва значение восстанавливаемого сигнала сходится к средине скачка. Как было показано Гиббсом, сумма сходится к сигналу в том смысле, что *ошибка* между ними имеет нулевую энергию.

Проблемы, связанные с эффектом Гиббса, часто встречаются в ЦОС. Например, низкочастотный фильтр является устройством, *обрезающим* высокие частоты, что приводит к выбросам и звону на фронтах во *временной области*. Другой обычной процедурой является усечение концов сигнала временной области для того, чтобы препятствовать их продлению на соседние периоды. В следствии дуальности, это искажает фронты в *частотной области*. Эти проблемы появятся еще на поверхности в будущих главах по проектированию фильтров.

Гармоники

Если сигнал с частотой f является периодическим, то единственные частоты, составляющие сигнал являются целыми кратными f, то есть, f, $2f$, $3f$, $4f$ и т.д. Эти частоты называются **гармониками**. **Первая гармоника** - f, **вторая гармоника** - $2f$, **третья гармоника** - $3f$, и так далее. Первой гармонике (то есть f) еще дают специальное название, **основная частота**. Некоторый пример этого показан на рис. 11.7. На рис. 11.7a приведена чистая синусоида, а на рис. 11.7b ее ДПФ - отдельный пик. На рис. 11.7c синусоидальная волна была искажена выступами в верхней части пиков. На рис. 11.7d приводится результат этих искажений в частотной области. Вследствие того, что искаженный сигнал является периодическим с такой же частотой, как и исходная синусоидальная волна, частотная область состоит из исходного пика плюс гармоники. Амплитуды гармоник могут быть любыми; однако, обычно они становятся меньше по мере увеличения частоты. Как и для любого сигнала, *крутые фронты* дают *более высокие частоты*. Например, рассмотрим обычную логическую ячейку ТТЛ (транзисторно-транзисторная логика, один из типов логических схем – прим. перев.), генерирующую прямоугольные импульсы с частотой 1 кГц. Фронты, нарастающие за несколько наносекунд, производят генерирование гармоник почти до 100 МГц - *десятитысячная* гармоника!

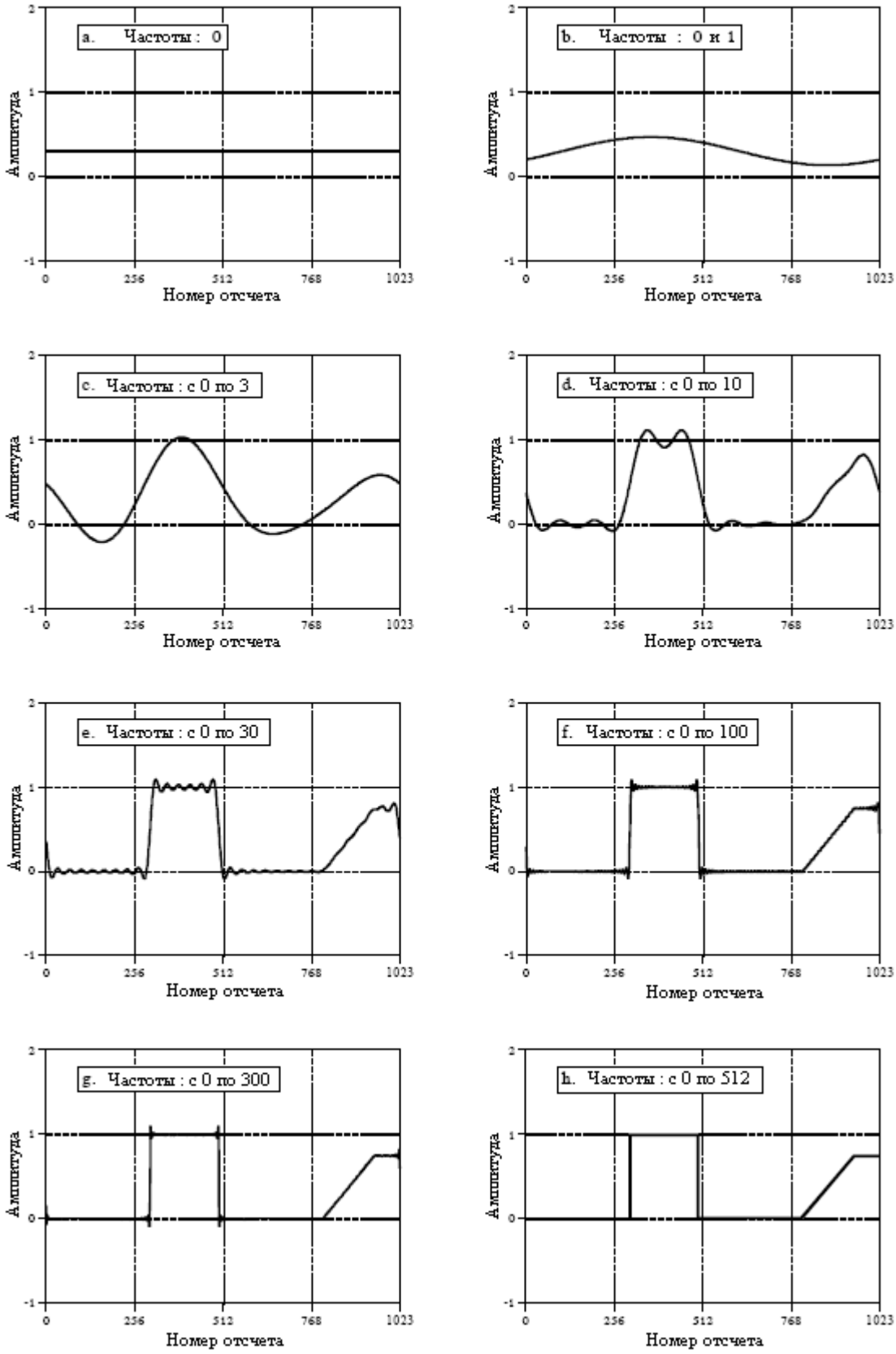


Рис. 11.6 Эффект Гиббса

Рисунок 11.7e демонстрирует тонкости гармонического анализа. Если сигнал симметричен относительно горизонтальной оси, т.е. верхние лепестки являются зеркальным отображением нижних лепестков, все четные гармоники будут иметь нулевое

значение. Как показано на рис. 11.7f, единственными частотами, содержащимися в сигнале, являются основная, третья гармоника, пятая гармоника и т.д.

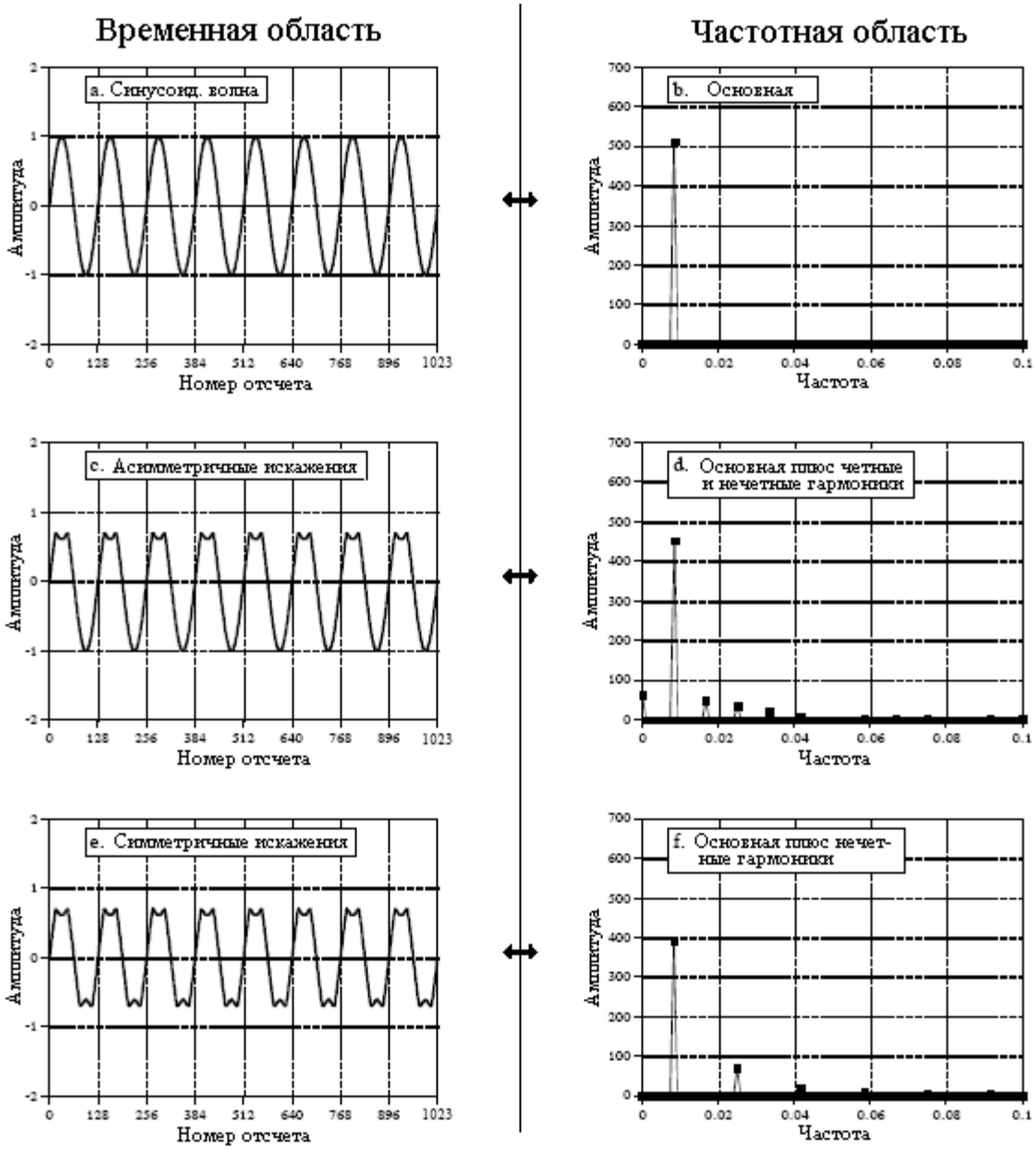


Рис. 11.7 Пример гармоник

Как было описано, все *непрерывные* периодические сигналы могут быть представлены в виде суммы гармоник. Для *дискретных* периодических сигналов существует проблема, разрушающая это простое соотношение. Как Вы возможно уже догадались, проблема заключается в *совмещении имен*. На рис. 11.8a показана синусоидальная волна, искаженная таким же образом, как и раньше, выступами в верхней части пиков. Эта форма волны выглядит гораздо менее регулярной и плавной чем в предыдущем примере, поскольку синусоидальная волна обладает здесь значительно более высокой частотой, что в результате дает меньшее число отсчетов приходящихся на один период. На рис. 11.8b показан частотный спектр этого сигнала. Как Вы, вероятно и ожидали, Вы можете выделить основную частоту и гармоники. Этот пример показывает, что гармоники могут простираться до частот больших, чем *0,5* от частоты дискретизации, и на частотах, где-нибудь между *0* и *0,5*, будет иметь место *совмещение имен*. Вы не замечаете этого на рис. 11.8b из-за слишком маленьких амплитуд. На рис. 11.8c показан

частотный спектр, вычерченный в логарифмическом масштабе для того, чтобы выявить малые амплитуды пиков совмещения имен. На первый взгляд, этот спектр напоминает случайный шум. Но это не так; это результат наложения множества гармоник из-за совмещения имен.

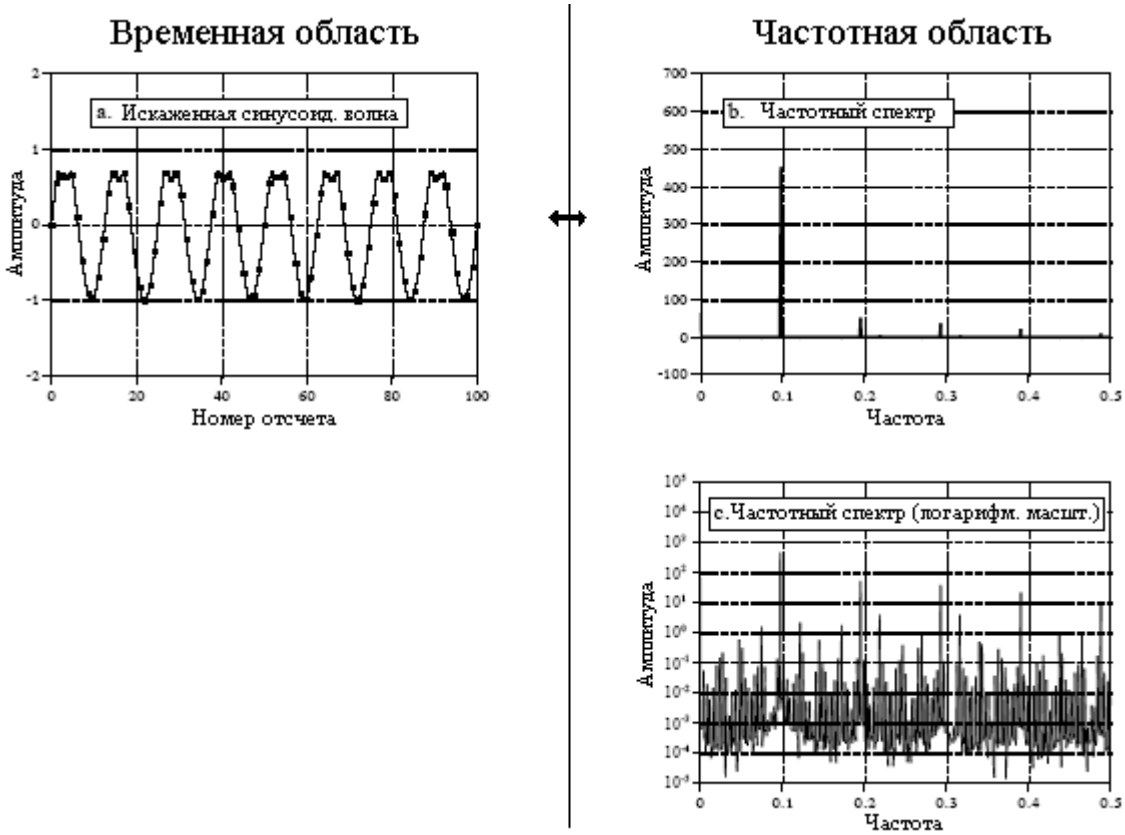


Рис. 11.8 Совмещение имен гармоник

Важно понять, что в этом примере сигнал искажается *после* того, как он был представлен в цифровой форме. Если бы эти искажения произошли в аналоговом сигнале, Вы бы удалили эти гармоники - правонарушители с помощью фильтра антисовмещения *до* оцифровки. Совмещение имен гармоник является проблемой только тогда, когда нелинейные операции выполняются непосредственно над дискретным сигналом. Часто, даже в этих случаях, амплитуда таких гармоник с совмещенными именами настолько мала, что их можно игнорировать.

Концепция гармоник полезна также и по другой причине: она объясняет, почему ДПФ рассматривает временную и частотную области как *периодические*. В частотной области N точечное ДПФ состоит из $N/2+1$, расположенных через одинаковые интервалы частот. Частоты *между* этими отсчетами, Вы можете рассматривать, как (1) имеющие значение нуля, или (2) как несуществующие. В любом случае они не вносят вклад в синтез сигнала временной области. Другими словами, *дискретный* частотный спектр состоит из *гармоник*, а не непрерывной области частот. Это требует, чтобы временная область была периодической с частотой равной частоте синусоиды с самой низкой частотой в частотной области, т.е. с основной частотой. Без учета постоянной составляющей, самая низкая частота, представленная в частотной области, совершает один полный период через каждые N отсчетов, приводя к тому, что временная область становится периодической с периодом N. Другими словами, если одна область является *дискретной*, другая область должна быть *периодической*, и наоборот. Это присуще для всех четырех членов семейства преобразований Фурье. Поскольку ДПФ рассматривает обе области как дискретные, оно

должно также рассматривать обе области как периодические. Отсчеты в каждой области представляют гармоники кратные периоду противоположной области.

Сигналы чириканья

Сигналы чириканья это остроумный способ обработки практических задач в системах эхолокации, таких как радиолокаторы и гидролокаторы. На рисунке 11.9 показан частотный отклик системы чириканья. Модуль имеет постоянное значение равное единице, тогда как фаза является параболой:

$$PhaseX[k] = \alpha k + \beta k^2 . \qquad (11.7)$$

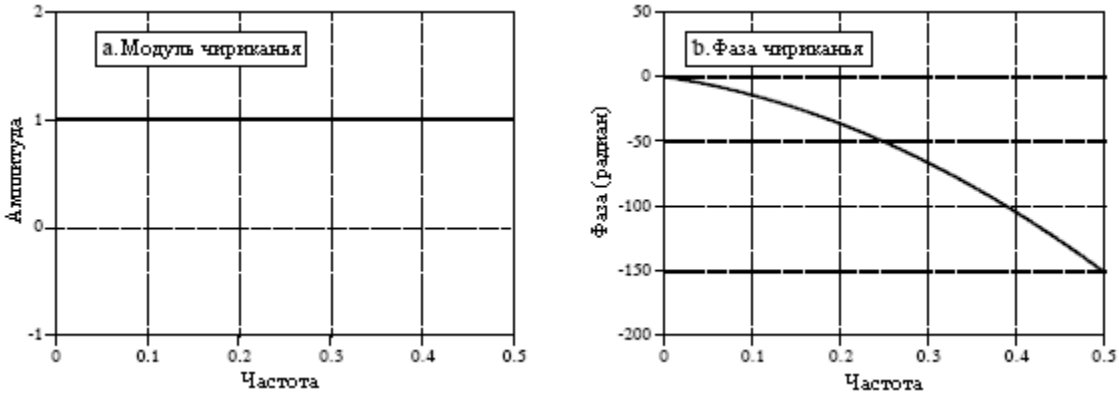


Рис. 11.9 Частотная характеристика системы чириканья

Параметр α вводит линейный наклон в фазу, то есть, он просто сдвигает, по желанию, импульсный отклик влево или вправо. Параметр β управляет *кривизной* фазы. Два этих параметра должны быть выбраны так, чтобы фаза на частоте 0,5 (т.е. $k=N/2$) была кратна 2π. Запомните, всякий раз, когда осуществляется непосредственное манипулирование фазой, частоты 0 и 0,5 обе должны иметь фазу равную нулю (или кратную 2π, что одно и тоже).

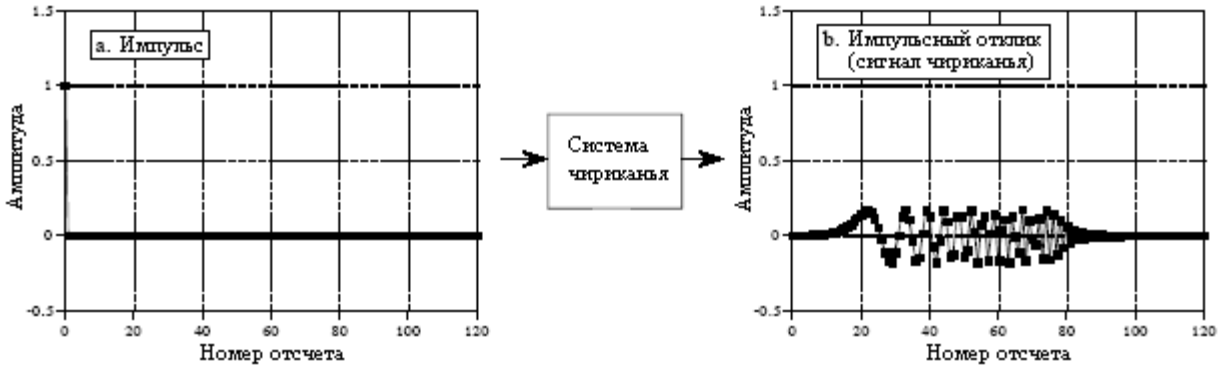


Рис. 11.10 Система чириканья

На рис. 11.10 показан импульс, входящий в систему чириканья, и импульсный отклик, покидающий систему. Импульсный отклик это пакет колебаний, который начинается с низкой частоты и с течением времени изменяется к высокой частоте. Это и

называется сигналом *чириканья* по очень простой причине: при воспроизведении его через громкоговоритель, он звучит как чириканье птички.

Ключевой характеристикой системы чириканья является то, что она полностью *обратима*. Если Вы пропустите сигнал чириканья через систему *античириканья*, сигнал снова превратится в импульс. Для этого требуется, чтобы система античириканья имела модуль равный единице, а фазу *противоположную* фазе системы чириканья. Как обсуждалось в предыдущей главе, это означает, что импульсный отклик системы античириканья находится выполнением переворота слева направо импульсного отклика системы чириканья. Интересно, но для чего это все нужно?

Рассмотрим, как работает радиолокационная система. Из направленной антенны излучается короткий всплеск энергии радио частоты. Самолет и другие объекты отражают часть этой энергии обратно к радиоприемнику, находящемуся рядом с передатчиком. Поскольку радиоволны распространяются с постоянной скоростью, конечное время между передачей и приемом сигналов дает расстояние до цели. Из этого следует первое требование к импульсу: необходимо, чтобы он был как можно короче. Например, импульс в одну микросекунду дает радио всплеск около 300 метров в длину. Это означает, что информация о расстоянии, которую мы получаем с помощью данной системы, будет иметь разрешающую способность, приблизительно такой же длинны. Если мы хотим лучшего разрешения по расстоянию, нам нужен более короткий импульс.

Второе требование очевидно: если мы хотим обнаруживать объекты на очень далеком расстоянии, нам необходимо сосредоточить как можно больше энергии в нашем импульсе. К сожалению *больше энергии* и *более короткий импульс* – требования, находящиеся в противоречии. Необходимая на создание импульса электрическая мощность равна энергии импульса, деленной на длительность импульса. Требования и *большего количества энергии* и *более короткого импульса* делает регулирование электрической мощностью сдерживающим фактором в системе. Выходной каскад радиопередатчика может регулировать только такие значения мощности, которые не приведут к его саморазрушению.

Сигналы чириканья дают способ преодоления этого ограничения. Прежде чем импульс достигнет оконечного каскада радио передатчика, он пропускается через систему чириканья. Вместо отражения от самолета-цели импульса, используется сигнал чириканья. После того как получено эхо чириканья, его пропускают через систему античириканья, возвращая сигнал к импульсу. Это позволяет той части системы, которая измеряет расстояние, видеть короткие импульсы, тогда как цепи, регулирующие мощность, видят сигналы большой длительности. Такой вид формирования сигналов является фундаментальной частью современных радиолокационных систем.

ГЛАВА
12

БЫСТРОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

Существует несколько способов вычисления дискретного преобразования Фурье (ДПФ) таких, как совместное решение линейных уравнений или корреляционный метод, описанных в Главе 8. Быстрое преобразование Фурье (БПФ), еще один способ вычисления ДПФ. Хотя он дает точно такой же результат, как и другие подходы, он невероятно более эффективен и часто снижает время вычисления в *сотни* раз. Это такое же усовершенствование, как полет на реактивном самолете по сравнению с ходьбой! Если бы БПФ не было доступно, многие из методов, описанных в этой книге, не получили бы практического применения. Хотя для БПФ требуется всего несколько дюжин строк программного кода, это один из наиболее сложных алгоритмов в ЦОС. Но не впадайте в отчаяние! Вы можете легко использовать опубликованные программы БПФ без полного понимания внутренних механизмов их работы.

Действительное ДПФ, использующее комплексное ДПФ

Надо отдать должное Дж. У. Кули и Дж. У. Тьюки за то, что они подарили миру БПФ в своей статье: "Алгоритм для машинного вычисления комплексных рядов Фурье", Вычислительная Математика, Т. 19, 1965, с. 297-301. В ретроспективе, за много лет до этого данная техника была отрыта другими. Например, великий немецкий математик Карл Фридрих Гаусс (1777-1855) использовал этот метод более чем столетие назад. Эта ранняя работа была в основном забыта, поскольку ей не доставало инструмента, чтобы применить ее на практике: *цифрового компьютера*. Кули и Тьюки удостоились чести, потому что они открыли БПФ в нужное время, в начале компьютерной революции.

БПФ основано на *комплексном ДПФ*, более утонченной версии, обсуждавшегося в четырех предыдущих главах *действительного ДПФ*. Эти преобразования получили названия по способу представления данных в каждом из них, то есть, использование комплексных чисел или использование действительных чисел. Термин *комплексное* не означает того, что это представление трудное или сложное, а означает только то, что здесь используется особый вид математики. Комплексная математика часто трудна и сложна, но названа она так не поэтому. В Главе 31 обсуждается комплексное ДПФ и приводятся основы, необходимые для понимания деталей алгоритма БПФ. Тема данной главы значительно проще: как использовать БПФ, чтобы вычислить действительное ДПФ, не утопая в трясине продвинутой математики.

Поскольку БПФ является алгоритмом для вычисления комплексного ДПФ, важно понимать, как преобразовать данные *действительного ДПФ* в формат и из формата *комплексного ДПФ*. На рис. 12.1 приводится сравнение того, как хранятся данные в действительном ДПФ и в комплексном ДПФ. Действительное ДПФ преобразует N точечный сигнал временной области в два $N/2+1$ точечных сигнала частотной области. Сигнал временной области так просто и называется: *сигнал временной области*. Два сигнала в частотной области, содержащие амплитуды косинусных волн и синусных волн, называются, соответственно, *действительной частью* и *мнимой частью*. Из предыдущих глав это должно быть Вам очень знакомо.

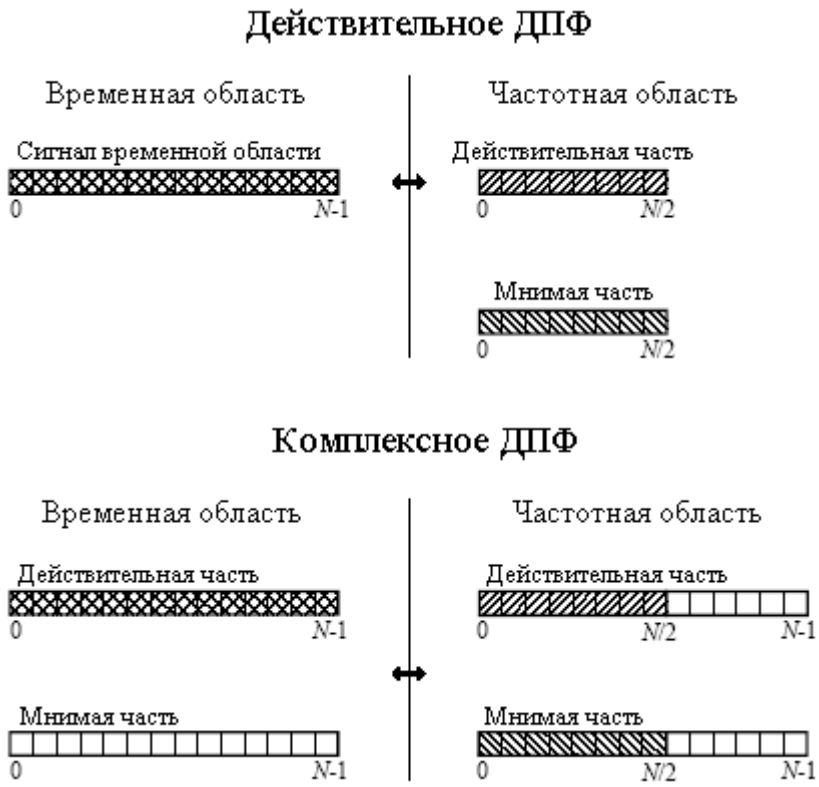


Рис. 12.1 Сравнение действительного и комплексного ДПФ

Для сравнения, комплексное ДПФ преобразует два *N* точечных сигнала временной области в два *N* точечных сигнала частотной области. Два сигнала временной области называются *действительной частью* и *мнимой частью*, так же, как и сигналы частотной области. Несмотря на их названия, все величины в этих множествах просто обычные числа. (Если Вы знакомы с комплексными числами: *j*-ы не включаются в множество значений; они являются частью *математики*. Вспомните, что оператор, *Im* ( ), возвращает действительное число.)

Предположим, что у Вас есть *N* точечный сигнал и нужно вычислить *действительное ДПФ* при помощи *комплексного ДПФ* (например, используя алгоритм БПФ). Во-первых, переместите *N* точечный сигнал в действительную часть временной области комплексного ДПФ, а затем установите значение всех отсчетов в мнимой части равным *нулю*. Вычисление комплексного ДПФ дает действительный и мнимый сигналы в частотной области, состоящие из *N* отсчетов каждый. Отсчеты этих сигналов с *0* по *N/2* соответствуют спектру действительного ДПФ.

Как обсуждалось в Главе 10, когда в рассмотрение включаются отрицательные частоты, частотная область ДПФ становится периодической (см. рис. 10.9). Выбор отдельного периода произволен; он может быть выбран между –1,0 и 0, -0,5 и 0,5, 0 и 1,0, или любым другим интервалом в одну единицу относительно частоты дискретизации. Частотный спектр комплексного ДПФ включает отрицательные частоты в расположение от 0 до 1,0. Другими словами, один полный период тянется от нулевого отсчета по отсчет с номером *N-1*, что соответствует частотам от 0 до 1,0 от частоты дискретизации. Положительные частоты располагаются между отсчетами *0* и *N/2*, что соответствует частотам от 0 до 0,5. Другие отсчеты, меду *N/2+1* и *N-1* содержат значения отрицательных частот (которые обычно игнорируются).

Вычисление *действительного обратного ДПФ* с использованием *комплексного обратного ДПФ* немного труднее. Это связано с тем, что Вам нужно подстраховаться, что отрицательные частоты загружены в надлежащем формате. Запомните, точки от *0* до *N/2* в

комплексном ДПФ точно такие же, как и в действительном ДПФ, как для действительной, так и для мнимой частей. Для действительной части, точка *N/2+1* точно такая же, как точка *N/2-1*, точка *N/2+2* точно такая же, как точка *N/2-2* и т.д. Это продолжается до точки *N-1*, которая является точно такой же, как точка *1*. Точно такой же базовый образец используется и для мнимой части, за исключением того, что меняется знак. То есть, точка *N/2+1* является отрицательной величиной точки *N/2-1*, точка *N/2+2* является отрицательной величиной точки *N/2-2* и т.д. Заметьте, что отсчеты *0* и *N/2* не имеют соответствующих точек в этой схеме дублирования. Для понимания этой симметрии используйте как путеводитель рис. 10.9. На практике Вы загружаете частотный спектр действительного ДПФ в отсчеты массива комплексного ДПФ с *0* по *N/2*, а затем используете подпрограмму для генерирования отрицательных частот между отсчетами *N/2+1* и *N-1*. Такая программа приведена в таблице 12.1. Для проверки того, что надлежащая симметрия присутствует, после взятия обратного ДПФ, посмотрите на мнимую часть временной области. Если все правильно она будет содержать все нули (за исключением шума в несколько долей на миллион при одинарной точности вычислений).

Таблица 12.1

```
6000 'NEGATIVE FREQUENCY GENERATION
6010 'This subroutine creates the complex frequency domain from the real frequency domain.
6020 'Upon entry to this subroutine, N% contains the number of points in the signals, and
6030 'REX[ ] and IMX[ ] contain the real frequency domain in samples 0 to N%/2.
6040 'On return, REX[ ] and IMX[ ] contain the complex frequency domain in samples 0 to N%-
1.
6050 '
6060 FOR K% = (N%/2+1) TO (N%-1)
6070 REX[K%] = REX[N%-K%]
6080 IMX[K%] = -IMX[N%-K%]
6090 NEXT K%
6100 '
6110 RETURN
```

Как работает БПФ

БПФ - это сложный алгоритм, и его детали обычно оставляют для тех, кто специализируются на таких вещах. В этом разделе описываются основные операции БПФ, но обходится ключевой вопрос: использование *комплексных чисел*. Если у Вас есть подготовка в комплексной математике, Вы можете читать между строк, чтобы понять истинную природу алгоритма. Не беспокойтесь, если от Вас ускользнут детали; немногие ученые и инженеры, использующие БПФ, могли бы написать программу с нуля.

В комплексной системе обозначений, временная и частотная области, каждая, содержат *один сигнал*, состоящий из *N* комплексных точек. Каждая из этих комплексных точек состоит из двух чисел, действительной части и мнимой части. Например, когда мы говорим о комплексном отсчете *X*[*42*], это относится к комбинации из *Re X*[*42*] и *Im X*[*42*]. Другими словами, каждая комплексная переменная содержит два числа. При умножении двух комплексных переменных, для того чтобы сформировать два компонента произведения (таких, как в уравнении (9.1)), должны быть объединены четыре отдельных составляющих. Последующее обсуждение по вопросу "*Как работает БПФ*" использует этот жаргон комплексной системы обозначений. То есть, термины в единственном числе: *сигнал*, *точка*, *отсчет* и *значение* относятся к *комбинации* действительной части и

мнимой части. БПФ работает за счет разложения *N* точечного сигнала временной области на *N* сигналов временной области, каждый из которых состоит из одной точки. Вторым шагом является вычисление *N* частотных спектров соответствующих этим *N* сигналам временной области. Наконец, *N* спектров объединяются в один частотный спектр.

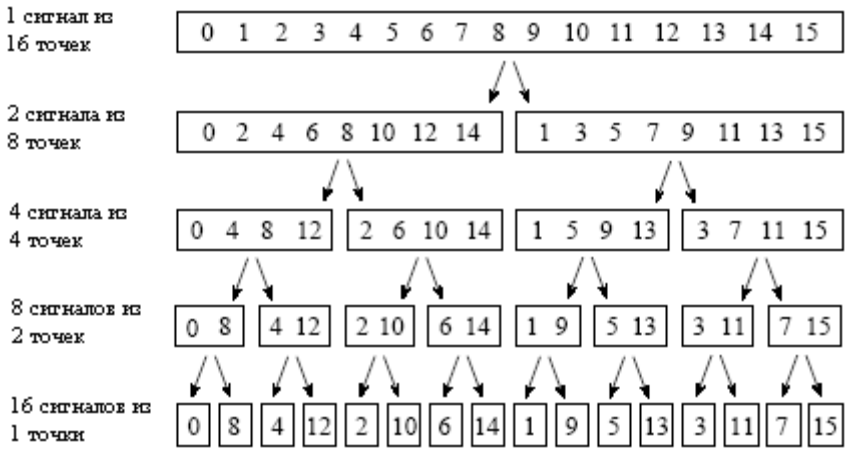


Рис. 12.2 Декомпозиция, используемая в БПФ

На рис. 12.2 показан пример декомпозиции временной области, используемый в БПФ. В этом примере 16 точечный сигнал раскладывается в четыре отдельных этапа. На первом этапе 16 точечный сигнал разбивается на два сигнала, каждый из которых состоит из 8 точек. На втором этапе данные разбиваются на 4 сигнала по 4 точки в каждом. Эта процедура продолжается до тех пор, пока не получится *N* сигналов содержащих по одной точке. Каждый раз, когда сигнал разбивается на два, используется **перекрестная декомпозиция**, то есть сигнал разделяется на отсчеты с четными и нечетными номерами. Лучшим способом понять это является разглядывание рис. 12.2 до тех пор, пока не увидишь узор. Для такой декомпозиции требуется $log_2 N$ этапов, т.е. 16 точечный сигнал (2^4) требует 4 этапа, 512 точечный сигнал (2^7) требует 7 этапов, 4096 точечный сигнал (2^{12}) требует 12 этапов и т.д. Запомните эту величину $log_2 N$; в этой главе мы будем ссылаться на нее много раз.

Номера отсчетов в обычном порядке		Номера отсчетов после переворота битов слева направо	
Десятичный	*Двоичный*	*Десятичный*	*Двоичный*
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0100
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111

Рис. 12.3 Сортировка перевернутых битов в БПФ

Теперь, когда Вы понимаете структуру декомпозиции, она может быть сильно упрощена. Декомпозиция представляет собой ничто иное, как *перестановку* отсчетов в сигнале. На рис. 12.3 показан образец требуемой перестановки. Слева приведены номера отсчетов исходного сигнала вместе с их двоичными эквивалентами. Справа приведены номера отсчетов после перестановки тоже с их двоичными эквивалентами. Важной идеей является то, что двоичные номера являются *перевернутой* слева направо копией друг друга. Например, отсчет номер 3 (0011) меняется местами с отсчетом номер 12 (1100). Аналогично, отсчет номер 14 (1110) обменивается с отсчетом номер 7 (0111) и так далее. Декомпозиция временной области БПФ обычно выполняется с помощью алгоритма **сортировки перевернутых битов**. Он включает в себя изменение порядка расположения *N* отсчетов временной области с переворотом при помощи двоичных вычислений битов номеров отсчетов слева направо (так же, как в самом правом столбце на рис. 12.3).

Следующим шагом в алгоритме БПФ является нахождение частотного спектра 1 точечного сигнала временной области. Нет ничего проще; частотный спектр 1 точечного сигнала эквивалентен *самому* сигналу. Это означает, что для выполнения этого шага не требуется *ничего*. Хотя здесь не было проделано никакой работы, не забывайте, что каждый из этих 1 точечных сигналов является теперь частотным спектром, а не сигналом временной области.

Последним шагом в БПФ является объединение *N* частотных спектров точно в обратном порядке, который имел место при декомпозиции временной области. Вот здесь алгоритм становится запутанным. К сожалению, упрощение переворотом битов номеров отсчетов не применимо, и мы должны двигаться в обратном направлении, проходя каждый раз лишь по одному этапу. На первом этапе 16 частотных спектров (по 1 точке каждый) объединяются в 8 частотных спектров (по 2 точки каждый). На втором этапе 8 частотных спектров (по 2 точки каждый) объединяются в 4 частотных спектра (по 4 точки каждый) и так далее. Последний этап дает на выходе БПФ 16 точечный частотный спектр.

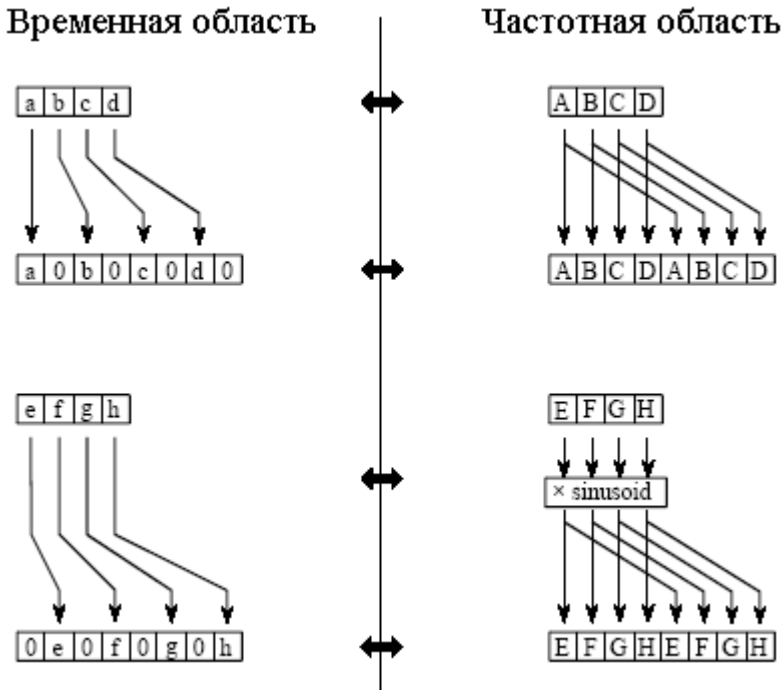


Рис. 12.4 БПФ синтез

На рис. 12.4 показано, как два частотных спектра, состоящие из 4 точек каждый, объединяются в единый частотный спектр из 8 точек. Это объединение должно *отменить* перекрестную декомпозицию, выполненную во временной области. Другими словами, операции частотной области должны соответствовать процедурам временной области по

комбинированию двух 4 точечных сигналов с помощью перекрещивания. Сравните два сигнала временной области, *abcd* и *efgh*. Восьми точечный сигнал временной области может быть сформирован за два шага: дополнить каждый 4 точечный сигнал нулями, чтобы сделать его 8 точечным сигналом, а затем сложить сигналы вместе. То есть *abcd* становится *a0b0c0d0*, а *efgh* становится *0e0f0g0h*. Сложение этих двух 8 точечных сигналов дает *abcdefgh*. Как показано на рис. 12.4, дополнение временной области нулями соответствует *удвоению* частотного спектра. Следовательно, частотные спектры объединяются в БПФ с помощью их удвоения, а затем сложения удвоенных спектров вместе.

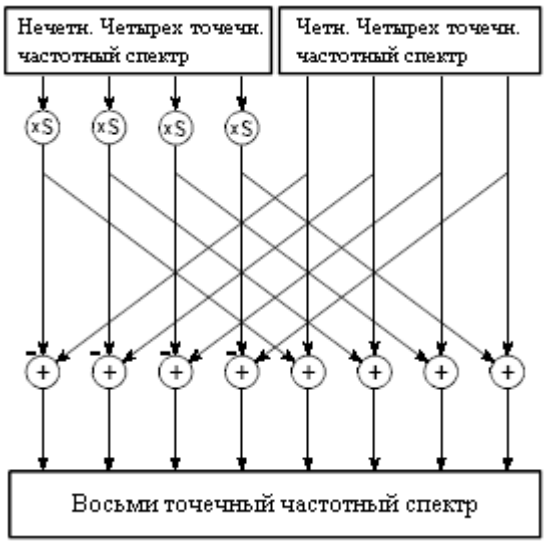


Рис. 12.5 Структурная схема БПФ синтеза

Для того чтобы добиться соответствия при суммировании, два сигнала временной области дополняются нулями немного разными способами. В одном сигнале нулевыми являются *нечетные точки*, в то время как в другом сигнале нулевыми являются *четные точки*. Другими словами, один из сигналов временной области (*0e0f0g0h* на рис. 12.4) сдвигается вправо на один отсчет. Этот сдвиг во временной области соответствует умножению спектра на *синусоиду*. Для того чтобы это увидеть, вспомните, что сдвиг во временной области эквивалентен свертке сигнала со сдвинутой дельта функцией. Это перемножает спектр сигнала со спектром сдвинутой дельта функции. Спектром сдвинутой дельта функции является синусоида (см. рис. 11.2).

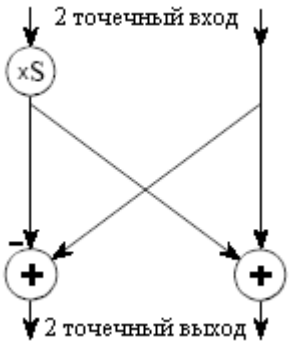


Рис. 12.6 Бабочка БПФ

На рис. 12.5 показана структурная схема объединения двух 4 точечных спектров в единый 8 точечный спектр. Для того чтобы упростить ситуацию еще больше, обратите внимание, что изображение на рис. 12.5 образовано многократным повторением базового

блока, показанного на рис. 12.6. Эта простая структурная схема из-за ее крылатого внешнего вида называется **бабочкой**. Бабочка основной вычислительный элемент БПФ, преобразующий две комплексные точки в две другие комплексные точки.

На рис. 12.7 показана структура всего БПФ. Декомпозиция временной области с помощью алгоритма сортировки перевернутых битов. Преобразование данных, подвергнутых декомпозиции, в частотную область, не включающее *ничего* и, следовательно, не отраженное на рисунке.

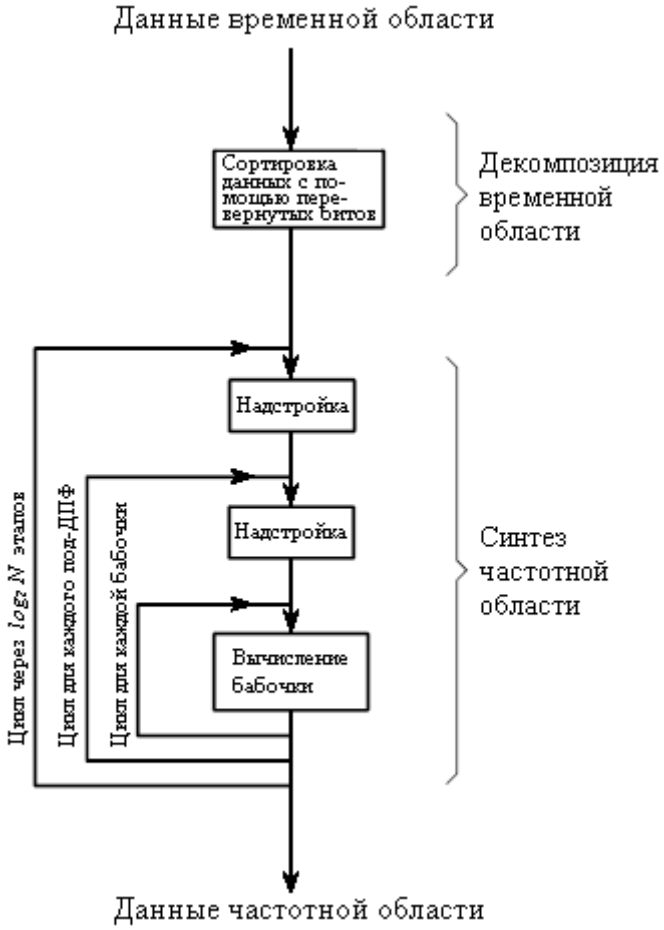


Рис. 12.7 Структурная схема БПФ

Синтез частотной области, требующий трех циклов. Внешний цикл проходит через $log_2 N$ этапов (т.е. каждый уровень, начиная с основания и двигаясь к вершине). Средний цикл проходит через каждую индивидуальную частоту спектра, обрабатываемую на конкретном этапе (т.е. каждый квадратик на одном из уровней на рис. 12.2). Внутренний цикл использует бабочку для вычисления точек в каждом частотном спектре (т.е. зацикливание на отсчетах внутри каждого квадратика на рис. 12.2). Надстроечные блоки на рис. 12.7 определяют начальные и конечные индексы для циклов, а также вычисляют необходимые в бабочках синусоиды. Теперь мы переходим к сердцу этой главы фактическим программам БПФ.

Программы БПФ

Как обсуждалось в Главе 8, *действительное ДПФ* может быть вычислено при помощи корреляции сигнала временной области с синусной и косинусной волнами (см. таблицу 8.2). В таблице 12.2 показана программа для вычисления *комплексного ДПФ* с

помощью этого же самого метода. При сравнении яблок с яблоками - это программа, в которой БПФ достаточно улучшено.

В таблицах 12.3 и 12.4 показаны две разные программы БПФ одна на Фортране и одна на Бейсике. Сначала мы посмотрим программу на Бейсике в таблице 12.4. Эта подпрограмма дает точно такой же результат, как и метод корреляции в таблице 12.2, кроме того, она делает это *гораздо быстрее*. Для идентификации различных участков этой программы может использоваться блок-схема на рис. 12.7. Данные, пропускаемые через эту подпрограмму БПФ, находятся в массивах REX[ ] и IMX[ ], каждый из которых идет от отсчета с номером *0* до отсчета с номером *N-1*. По возвращении из подпрограммы, REX[ ] и IMX[ ] перезаписаны данными частотной области. Это еще один способ высокой оптимизации БПФ; одни и те же массивы используются для входных, промежуточных, хранимых и выходных данных. Такое эффективное использование памяти является важным при проектировании быстрого аппаратного обеспечения, предназначенного для вычисления БПФ. Для описания такого использования памяти используют термин **вычисление на месте**.

Таблица 12.2

```
5000 'COMPLEX DFT BY CORRELATION
5010 'Upon entry, N% contains the number of points in the DFT, and
5020 'XR[ ] and XI[ ] contain the real and imaginary parts of the time domain.
5030 'Upon return, REX[ ] and IMX[ ] contain the frequency domain data.
5040 'All signals run from 0 to N%-1.
5050 '
5060 PI = 3.14159265                    'Set constants
5070 '
5080 FOR K% = 0 TO N%-1                 'Zero REX[ ] and IMX[ ], so they can be used
5090 REX[K%] = 0                        'as accumulators during the correlation
5100 IMX[K%] = 0
5110 NEXT K%
5120 '
5130 FOR K% = 0 TO N%-1                 'Loop for each value in frequency domain
5140 FOR I% = 0 TO N%-1                 'Correlate with the complex sinusoid, SR & SI
5150 '
5160 SR = COS(2*PI*K%*I%/N%)            'Calculate complex sinusoid
5170 SI = -SIN(2*PI*K%*I%/N%)
5180 REX[K%] = REX[K%] + XR[I%]*SR - XI[I%]*SI
5190 IMX[K%] = IMX[K%] + XR[I%]*SI + XI[I%]*SR
5200 '
5210 NEXT I%
5220 NEXT K%
5230 '
5240 RETURN
```

Хотя все программы БПФ дают один и тот же численный результат, в программировании существуют тонкие вариации, которые Вам следует отыскивать. Некоторые из этих важных различий иллюстрируются с помощью программы на Фортране приведенной в таблице 12.3. Эта программа использует алгоритм, называемый **децимация по частоте**, в то время как предварительно описанный алгоритм называется **децимация по времени**. В алгоритме децимации по частоте сортировка перевернутых битов выполняется *после* трех вложенных циклов. Существуют также программы БПФ,

которые полностью устраняют сортировку перевернутых битов. Ни одна из этих разновидностей значительно не улучшает выполнение БПФ, и Вы не должны волноваться относительно того, какую из них Вы используете.

Важные различия между алгоритмами БПФ касаются того, как данные проходят к и от подпрограмм. В программе на Бейсике данные входят и покидают подпрограмму в массивах REX[ ] и IMX[ ], каждый из которых идет от отсчета с номером *0* по отсчет с номером *N-1*. В программе на Фортране данные поступают в комплексный массив X( ), идущий от отсчета с номером *1* по отсчет с номером *N*. Поскольку это массив комплексных переменных, каждый отсчет в X( ) состоит из двух чисел, действительной части и мнимой части. На эти подпрограммы должна также поступать длина ДПФ. В программе на Бейсике для этой цели используется переменная N%. Для сравнения, программа на Фортране использует переменную M, определенную равной $log_2 N$. Например, M будет равно 8 для 256 точечного ДПФ, 12 для 4096 точечного ДПФ и т.д. Суть в том, что программист, пишущий подпрограмму БПФ, имеет большое число параметров взаимодействующих с основной программой. Массивы, идущие от *1* по *N*, такие как в программе на Фортране, особенно ухудшают ситуацию. Большинство литературы по ЦОС (включая эту книгу) объясняет алгоритмы, предполагая, что массивы идут от отсчета с номером *0* по отсчет с номером *N-1*. Например, если массивы идут от *1* по *N*, то симметрия в частотной области наблюдается вокруг точек *1* и *N/2+1*, а не вокруг точек *0* и *N/2*.

Таблица 12.3

```
      SUBROUTINE FFT(X,M)
      COMPLEX X(4096),U,S,T
      PI=3.14159265
      N=2**M
      DO 20 L=1,M
      LE=2**(M+1-L)
      LE2=LE/2
      U=(1.0,0.0)
      S=CMPLX(COS(PI/FLOAT(LE2)),-SIN(PI/FLOAT(LE2)))
      DO 20 J=1,LE2
      DO 10 I=J,N,LE
      IP=I+LE2
      T=X(I)+X(IP)
      X(IP)=(X(I)-X(IP))*U
10    X(I)=T
20    U=U*S
      ND2=N/2
      NM1=N-1
      J=1
      DO 50 I=1,NM1
      IF(I.GE.J) GO TO 30
      T=X(J)
      X(J)=X(I)
      X(I)=T
30    K=ND2
40    IF(K.GE.J) GO TO 50
      J=J-K
      K=K/2
      GO TO 40
```

```
50      J=J+K
        RETURN
        END
```

Таблица 12.4

```
1000 'THE FAST FOURIER TRANSFORM
1010 'Upon entry, N% contains the number of points in the DFT, REX[ ] and
1020 'IMX[ ] contain the real and imaginary parts of the input. Upon return,
1030 'REX[ ] and IMX[ ] contain the DFT output. All signals run from 0 to N%-1.
1040 '
1050 PI = 3.14159265                                    'Set constants
1060 NM1% = N%-1
1070 ND2% = N%/2
1080 M% = CINT(LOG(N%)/LOG(2))
1090 J% = ND2%
1100 '
1110 FOR I% = 1 TO N%-2                                 'Bit reversal sorting
1120 IF I% >= J% THEN GOTO 1190
1130 TR = REX[J%]
1140 TI = IMX[J%]
1150 REX[J%] = REX[I%]
1160 IMX[J%] = IMX[I%]
1170 REX[I%] = TR
1180 IMX[I%] = TI
1190 K% = ND2%
1200 IF K% > J% THEN GOTO 1240
1210 J% = J%-K%
1220 K% = K%/2
1230 GOTO 1200
1240 J% = J%+K%
1250 NEXT I%
1260 '
1270 FOR L% = 1 TO M%                                   'Loop for each stage
1280 LE% = CINT(2^L%)
1290 LE2% = LE%/2
1300 UR = 1
1310 UI = 0
1320 SR = COS(PI/LE2%)                                  'Calculate sine & cosine values
1330 SI = -SIN(PI/LE2%)
1340 FOR J% = 1 TO LE2%                                 'Loop for each sub DFT
1350 JM1% = J%-1
1360 FOR I% = JM1% TO NM1% STEP LE%                     'Loop for each butterfly
1370 IP% = I%+LE2%
1380 TR = REX[IP%]*UR - IMX[IP%]*UI                     'Butterfly calculation
1390 TI = REX[IP%]*UI + IMX[IP%]*UR
1400 REX[IP%] = REX[I%]-TR
1410 IMX[IP%] = IMX[I%]-TI
1420 REX[I%] = REX[I%]+TR
1430 IMX[I%] = IMX[I%]+TI
1440 NEXT I%
```

```
1450 TR = UR
1460 UR = TR*SR - UI*SI
1470 UI = TR*SI + UI*SR
1480 NEXT J%
1490 NEXT L%
1500 '
1510 RETURN
```

Использование комплексного ДПФ для вычисления действительного ДПФ имеет другое интересное преимущество. Комплексное ДПФ обладает большей симметрией между временной и частотной областями, чем действительное ДПФ. То есть **дуальность** выражена сильнее. Среди прочих вещей это означает, что обратное ДПФ почти идентично прямому ДПФ. Действительно, самый легкий способ вычислить *обратное БПФ* состоит в том, чтобы вычислить *прямое БПФ*, а затем выверить данные. В таблице 12.5 показана подпрограмма для вычисления обратного БПФ в такой манере.

Предположим, что Вы скопировали один из этих алгоритмов БПФ в свою компьютерную программу и запустили ее прогон. Как Вы узнаете, правильно ли она работает? Обычно при отладке используются два приема. Во-первых, начните с некоторого произвольного сигнала временной области, такого как числа от генератора случайных чисел, и пропустите их через БПФ. Затем, пропустите результирующий частотный спектр через обратное БПФ и сравните результат с исходным сигналом. Они должны быть *идентичными*, за исключением шума округления (несколько долей на миллион, при одинарной точности).

Вторым тестом на правильную работу служит то, что сигналы обладают правильной *симметрией*. Когда мнимая часть сигнала временной области состоит из всех нулей (нормальный случай), частотная область комплексного ДПФ будет симметрична вокруг отсчетов *0* и *N/2*, как уже описывалось раньше.

Таблица 12.5

```
2000 'INVERSE FAST FOURIER TRANSFORM SUBROUTINE
2010 'Upon entry, N% contains the number of points in the IDFT, REX[ ] and
2020 'IMX[ ] contain the real and imaginary parts of the complex frequency domain.
2030 'Upon return, REX[ ] and IMX[ ] contain the complex time domain.
2040 'All signals run from 0 to N%-1.
2050 '
2060 FOR K% = 0 TO N%-1                    'Change the sign of IMX[ ]
2070 IMX[K%] = -IMX[K%]
2080 NEXT K%
2090 '
2100 GOSUB 1000                            'Calculate forward FFT (Table 12-3)
2110 '
2120 FOR I% = 0 TO N%-1                    'Divide the time domain by N% and
2130 REX[I%] = REX[I%]/N%                  'change the sign of IMX[ ]
2140 IMX[I%] = -IMX[I%]/N%
2150 NEXT I%
2160 '
2170 RETURN
```

Аналогичным образом, когда в частотной области существует такая правильная симметрия, обратное ДПФ даст временную область, мнимая часть которой, состоит из всех нулей (плюс шум округления). Для использования БПФ такие методы отладки необходимы, хорошо познакомьтесь с ними.

Сравнения скорости и точности

Когда ДПФ вычисляется с помощью корреляции (как в таблице 12.2), программа использует два вложенных цикла, каждый из которых проходит через N точек. Это означает, что общее количество операций пропорционально N *взятому* N *раз*. Таким образом, время для завершения программы определяется как:

$$Время \quad выполнения = k_{ДПФ} N^2, \tag{12.1}$$

где N - число точек в ДПФ и $k_{ДПФ}$ – константа пропорциональности. Если синусные и косинусные величины вычисляются *при помощи* вложенных циклов, $k_{ДПФ}$ равна приблизительно 25 микросекундам для процессора Pentium с тактовой частотой 100 МГц. Если Вы *предварительно вычисляете* синусные и косинусные величины и храните их в справочной таблице, $k_{ДПФ}$ уменьшается приблизительно до 7 микросекунд. Например, на выполнение 1024 точечного ДПФ потребуется около 25 секунд, или около 25 миллисекунд на одну точку. Это медленно!

Используя туже самую стратегию, мы можем получить время выполнения для БПФ. Время, требуемое для сортировки перевернутых битов, является незначительным. На каждом из $log_2\ N$ этапов выполняется $N/2$ вычислений бабочки. Это означает, что время для выполнения программы приближается к:

$$Время \quad выполнения = k_{БПФ} N \log_2 N. \tag{12.2}$$

Значение $k_{БПФ}$ для системы Pentium при 100 МГц около 10 микросекунд. 1024 точечное БПФ потребует для выполнения около 70 миллисекунд, или 70 микросекунд на одну точку. Это более чем в 300 раз быстрее, чем ДПФ вычисленное при помощи корреляции!

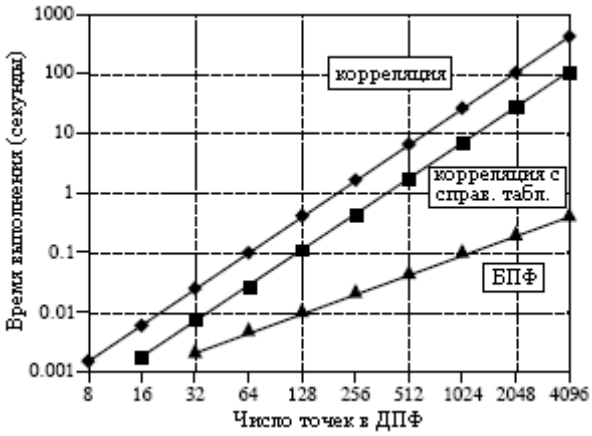


Рис. 12.8 Время выполнения вычисления БПФ

$N\ log_2\ N$ не только меньше, чем N^2, по мере увеличения N, оно еще и увеличивается существенно более медленно. Например, 32 точечное БПФ в *десять* раз быстрее, чем

корреляционный метод. Однако 4096 точечное БПФ быстрее в *одну тысячу* раз. Для малых значений *N* (скажем, от 32 до 128) использование БПФ является важным. Для больших значений *N* (1024 и больше) использование БПФ крайне необходимо. На рис. 12.8 приводится сравнение времени выполнения этих двух алгоритмов в графическом виде.

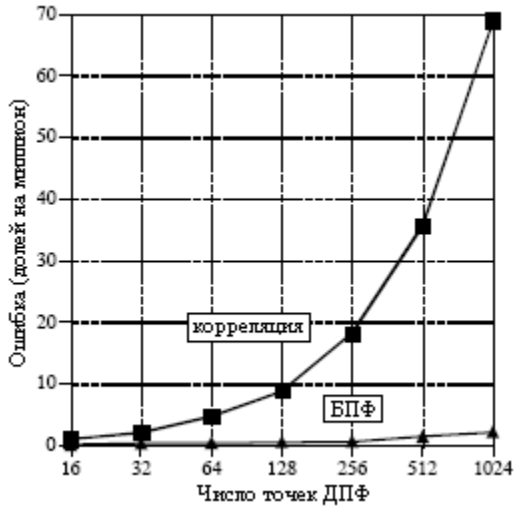


Рис. 12.9 Точность ДПФ

Помимо естественной скорости БПФ имеет еще одно преимущество. БПФ вычисляется более *точно*, поскольку меньшее количество вычислений дает меньшую ошибку округления. Это можно продемонстрировать взятием БПФ произвольного сигнала, а затем пропусканием частотного спектра через обратное БПФ. Это восстановит исходный сигнал временной области, *за исключением* добавленного от вычислений шума округления. Некоторое число, характеризующее это шум, может быть получено вычислением стандартного отклонения разницы между двумя сигналами. Для сравнения, та же самая процедура может быть повторена с использованием ДПФ, вычисленного при помощи корреляции, и соответствующего обратного ДПФ. А как шум округления БПФ сравнивается с шумом округления ДПФ, вычисленного при помощи корреляции? Посмотрите самостоятельно рис. 12.9.

Дальнейшее увеличение скорости

Существует несколько способов сделать БПФ еще быстрее; однако, улучшение составляет всего около 20-40%. В одном из этих методов, декомпозиция временной области останавливается на два этапа раньше, тогда, когда каждый сигнал состоит только из 4 точек. Вместо того чтобы осуществлять вычисления последних двух этапов, для того чтобы осуществить непосредственный переход в частотную область, применяется высоко оптимизированный код, использующий простоту четырех точечных синусных и косинусных волн.

Другой популярный алгоритм устраняет бесполезные вычисления, связанные с равной нулю мнимой частью временной области и симметричным частотным спектром. Другими словами БПФ модифицируется для вычисления *действительного ДПФ* вместо *комплексного ДПФ*. Такие алгоритмы называются **действительным БПФ** и **действительным обратным БПФ** (или подобными названиями). Ожидайте, что они будут приблизительно на 30 % быстрее, чем обычные программы БПФ. В таблицах 12.6 и 12.7 показаны программы для этих алгоритмов.

В использовании *действительного БПФ* есть два небольших недостатка. Во-первых, их программный код примерно в два раза длинней. Хотя это и не волнует Ваш компьютер, Вам, для преобразования чьей-то программы для работы на Вашем компьютере нужно будет затратить время. Во-вторых, отладка таких программ несколько труднее потому, что для проверки правильной работы Вы не сможете использовать симметрию. Такие алгоритмы принудительно *заставляют* мнимую часть временной области быть нулевой, а частотную область иметь симметрию слева - направо. При отладке проверьте, что эти программы дают тот же самый результат, что и традиционные алгоритмы БПФ.

Таблица 12.6

```
4000 'INVERSE FFT FOR REAL SIGNALS
4010 'Upon entry, N% contains the number of points in the IDFT, REX[ ] and
4020 'IMX[ ] contain the real and imaginary parts of the frequency domain running from
4030 'index 0 to N%/2. The remaining samples in REX[ ] and IMX[ ] are ignored.
4040 'Upon return, REX[ ] contains the real time domain, IMX[ ] contains zeros.
4050 '
4060 '
4070 FOR K% = (N%/2+1) TO (N%-1)          'Make frequency domain symmetrical
4080 REX[K%] = REX[N%-K%]                 '(as in Table 12-1)
4090 IMX[K%] = -IMX[N%-K%]
4100 NEXT K%
4110 '
4120 FOR K% = 0 TO N%-1                   'Add real and imaginary parts together
4130 REX[K%] = REX[K%]+IMX[K%]
4140 NEXT K%
4150 '
4160 GOSUB 3000                           'Calculate forward real DFT (TABLE 12-6)
4170 '
4180 FOR I% = 0 TO N%-1                   'Add real and imaginary parts together
4190 REX[I%] = (REX[I%]+IMX[I%])/N%       'and divide the time domain by N%
4200 IMX[I%] = 0
4210 NEXT I%
4220 '
4230 RETURN
```

На рис. 12.10 и 12.11 показано как работает действительное БПФ. На рис. 12.10a и рис. 12.10b показан сигнал временной области, состоящий из импульса в действительной части и всех нулей в мнимой части. На рис. 12.10c и рис. 12.10d показан соответствующий частотный спектр. Как описывалось ранее, действительная часть частотной области имеет *четную* симметрию вокруг отсчета с номером *0* и отсчета с номером *N/2*, в то время как мнимая часть имеет *нечетную* симметрию вокруг тех же самых точек.

Теперь рассмотрим рис. 12.11, где импульс находится в мнимой части временной области, а вся действительная часть нулевая. Симметрия в частотной области меняется на *противоположную*, действительная часть становится нечетной, в то время как мнимая часть четной. Эта ситуация будет обсуждена в Главе 31. А сейчас, примите как очевидное, что комплексное ДПФ ведет себя именно так.

А что если сигнал присутствует в *обеих частях* временной области? Благодаря аддитивности частотная область будет *суммой* двух частотных спектров. А теперь ключевой элемент: частотный спектр, состоящий из этих двух видов симметрии, может

быть идеально разделен на две составляющих сигнала. Это достигается при помощи *четной/нечетной декомпозиции*, обсуждаемой в Главе 5. Другими словами, два действительных ДПФ могут быть вычислены по цене одного БПФ. Один из сигналов помещается в действительную часть временной области, а другой сигнал помещается в мнимую часть. После вычисления комплексного ДПФ (конечно через БПФ), используя четную/нечетную декомпозицию, спектры разделяются. Когда через БПФ нужно пропустить два или более сигналов, данная техника снижает время выполнения примерно на 40%. Улучшения точно в два раза не происходит потому, что для выполнения четной/нечетной декомпозиции требуется время на вычисления. Эта относительно простая техника с небольшим количеством ловушек, ничто по сравнению с написанием программы БПФ с нуля.

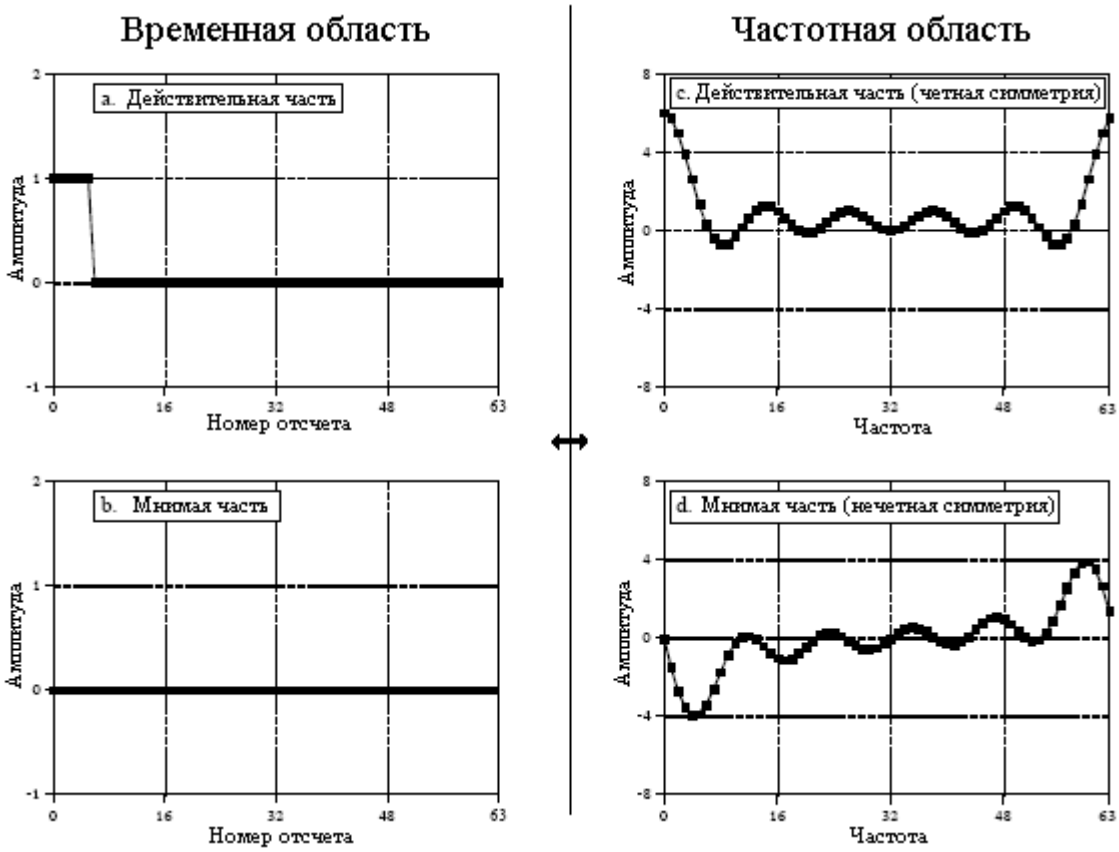


Рис. 12.10 Симметрия действительной части ДПФ

Следующим шагом является модификация алгоритма для более быстрого вычисления *одного* ДПФ. Это некрасиво, но вот как это делается. Входной сигнал разбивается пополам с помощью перекрестной декомпозиции. *N/2* четных точек помещаются в действительную часть сигнала временной области, тогда как *N/2* нечетных точек отправляются в мнимую часть. Затем вычисляется *N/2* точечное БПФ, требующее приблизительно половину времени от времени *N* точечного БПФ. Затем результирующая частотная область разделяется с помощью четной/нечетной декомпозиции, давая в результате частотные спектры двух перекрестных сигналов временной области. После чего эти два частотных спектра объединяются в один спектр, так же, как и в последнем этапе синтеза БПФ.

Для того чтобы закончить эту главу, считайте, что *БПФ* для *цифровой обработки сигнала* - это то же самое, что и *транзистор* для *электроники*. Это основа данной технологии, каждый в этой области знает его характеристики и как его использовать.

Однако только небольшое число специалистов действительно понимают детали его внутреннего механизма.

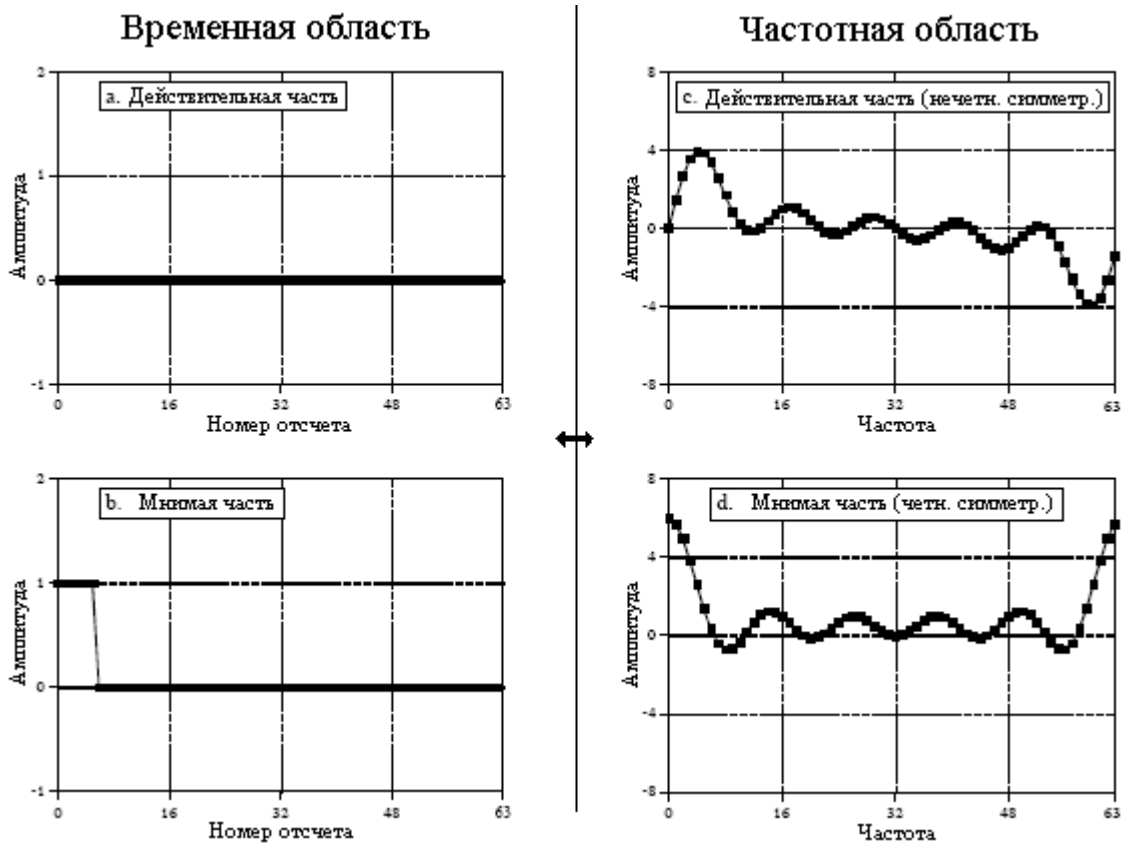


Рис. 12.11 Симметрия мнимой части ДПФ

Таблица 12.7

```
3000 'FFT FOR REAL SIGNALS
3010 'Upon entry, N% contains the number of points in the DFT, REX[ ] contains
3020 'the real input signal, while values in IMX[ ] are ignored. Upon return,
3030 'REX[ ] and IMX[ ] contain the DFT output. All signals run from 0 to N%-1.
3040 '
3050 NH% = N%/2-1                       'Separate even and odd points
3060 FOR I% = 0 TO NH%
3070 REX(I%) = REX(2*I%)
3080 IMX(I%) = REX(2*I%+1)
3090 NEXT I%
3100 '
3110 N% = N%/2                          'Calculate N%/2 point FFT
3120 GOSUB 1000                         '(GOSUB 1000 is the FFT in Table 12-3)
3130 N% = N%*2
3140 '
3150 NM1% = N%-1                        'Even/odd frequency domain decomposition
3160 ND2% = N%/2
3170 N4% = N%/4-1
3180 FOR I% = 1 TO N4%
3190 IM% = ND2%-I%
```

```
3200 IP2% = I%+ND2%
3210 IPM% = IM%+ND2%
3220 REX(IP2%) = (IMX(I%) + IMX(IM%))/2
3230 REX(IPM%) = REX(IP2%)
3240 IMX(IP2%) = -(REX(I%) - REX(IM%))/2
3250 IMX(IPM%) = -IMX(IP2%)
3260 REX(I%) = (REX(I%) + REX(IM%))/2
3270 REX(IM%) = REX(I%)
3280 IMX(I%) = (IMX(I%) - IMX(IM%))/2
3290 IMX(IM%) = -IMX(I%)
3300 NEXT I%
3310 REX(N%*3/4) = IMX(N%/4)
3320 REX(ND2%) = IMX(0)
3330 IMX(N%*3/4) = 0
3340 IMX(ND2%) = 0
3350 IMX(N%/4) = 0
3360 IMX(0) = 0
3370 '
3380 PI = 3.14159265                    'Complete the last FFT stage
3390 L% = CINT(LOG(N%)/LOG(2))
3400 LE% = CINT(2^L%)
3410 LE2% = LE%/2
3420 UR = 1
3430 UI = 0
3440 SR = COS(PI/LE2%)
3450 SI = -SIN(PI/LE2%)
3460 FOR J% = 1 TO LE2%
3470 JM1% = J%-1
3480 FOR I% = JM1% TO NM1% STEP LE%
3490 IP% = I%+LE2%
3500 TR = REX[IP%]*UR - IMX[IP%]*UI
3510 TI = REX[IP%]*UI + IMX[IP%]*UR
3520 REX[IP%] = REX[I%]-TR
3530 IMX[IP%] = IMX[I%]-TI
3540 REX[I%] = REX[I%]+TR
3550 IMX[I%] = IMX[I%]+TI
3560 NEXT I%
3570 TR = UR
3580 UR = TR*SR - UI*SI
3590 UI = TR*SI + UI*SR
3600 NEXT J%
3610 RETURN
```

ГЛАВА
13

ОБРАБОТКА НЕПРЕРЫВНЫХ СИГНАЛОВ

Обработка непрерывных сигналов - это область параллельная ЦОС, поэтому большинство методов используемых здесь и в ЦОС идентичны. Например, как обработка непрерывных сигналов, так и ЦОС, обе базируются на линейности, декомпозиции, свертке и Фурье анализе. Не ожидайте, что Вы найдете в этой главе компьютерные программы, поскольку непрерывные сигналы, нельзя непосредственно представить в компьютере. В основе обработки непрерывных сигналов лежит *математика*; сигналы представляются в виде уравнений, а системы преобразуют одни уравнения в другие. Также как *цифровые компьютеры* являются первичным инструментом, используемым в ЦОС, в обработке непрерывных сигналов, используемым первичным инструментом являются *исчисления*. Эти методы использовались в течение столетий, задолго до появления компьютеров.

Дельта функция

Таким же образом, как это было сделано с дискретными сигналами, непрерывные сигналы могут быть разложены на смасштабированные и сдвинутые *дельта функции*. Разница заключается в том, что непрерывная дельта функция значительно более сложна и математически более абстрактна, чем ее дискретный аналог. Вместо того чтобы давать определение непрерывной дельта функции через то, чем она *является*, мы будем определять ее через характеристики, которые у *нее есть*.

Как это все работает, покажет мысленный эксперимент. Представьте электронную схему, содержащую линейные компоненты, такие как резисторы, конденсаторы и индуктивности. Подсоединим к ее входу генератор сигналов, вырабатывающий короткие *импульсы* различной формы. Выход схемы соединим с осциллографом, показывающим форму волны, вырабатываемую этой схемой в ответ на каждый входной импульс. Вопрос, на который мы хотим найти ответ, состоит в следующем: *каким образом связана форма выходных импульсов с характеристиками входных импульсов*? Для упрощения исследований будем использовать только такие входные импульсы, длительность которых значительно меньше, чем выходного сигнала. Например, если длительность отклика системы составляет миллисекунды, мы должны использовать входной импульс длительностью всего несколько микросекунд.

После проведения большого числа измерений, мы приходим к трем заключениям. Первое, *форма* входного импульса не влияет на форму выходного сигнала. Это иллюстрируется на рис. 13.1, где входные импульсы разной формы дают выходной импульс одной и той же формы. Второе, форма волны выходного сигнала полностью определяется характеристиками системы, т.е. значением и конфигурацией соединения между собой резисторов, конденсаторов и индуктивностей. Третье, *амплитуда* выходного импульса прямопропорциональна *площади* входного импульса. Например, выходной сигнал будет иметь одну и ту же амплитуду для входных прямоугольных импульсов: амплитудой 1 вольт и длительностью 1 микросекунда, амплитудой 10 вольт и длительностью 0,1 микросекунда, амплитудой 1000 вольт и длительностью 1 наносекунда

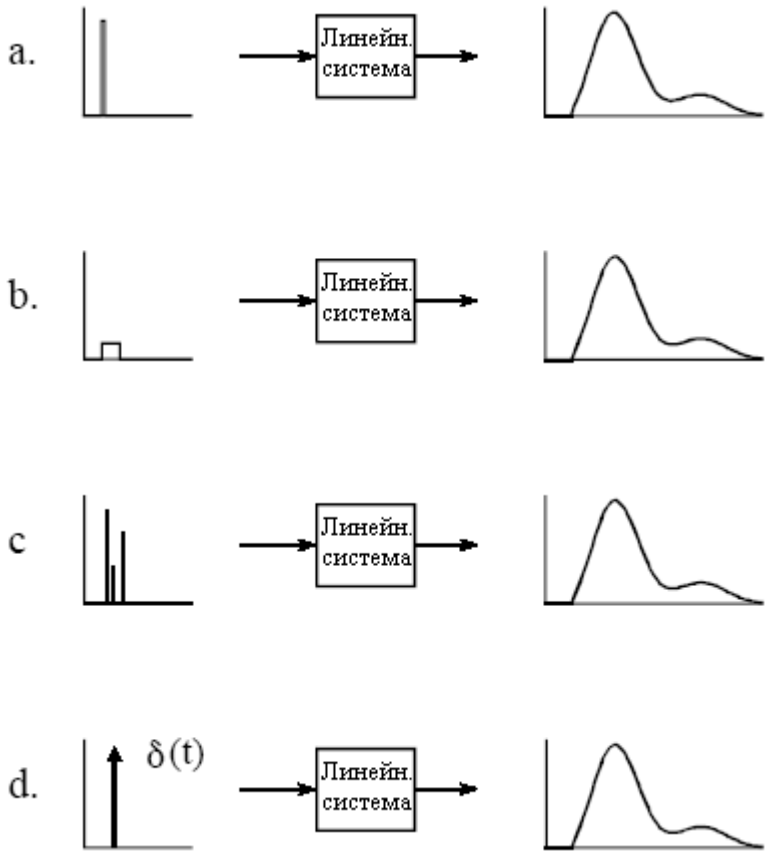


Рис. 13.1 Непрерывная дельта функция

и т.д. Это соотношение допустимо так же и для входных импульсов с *отрицательными* площадями. Например, представьте комбинацию импульса амплитудой 2 вольта и длительностью 2 микросекунды, за которым мгновенно следует импульс амплитудой минус 1 вольт и длительностью 4 микросекунды. Общая площадь входного сигнала равна *нулю*, в результате на выходе *ничего* не произойдет.

Сигналы достаточно короткие, для того чтобы иметь все три этих свойства, называют **импульсами**. Другими словами, импульс это некоторый сигнал, который везде, кроме короткого *всплеска* произвольной формы, равен нулю. Например, импульс для микроволнового передатчика может быть *пикосекундной* длительности, поскольку отклик электроники *наносекундный*. Для сравнения, вулкан, извергающийся *годами*, может быть великолепным импульсом для геологических изменений, происходящих *тысячелетиями*.

Математикам не нравится, когда их ограничивают рамками конкретной системы, и они обычно используют термин *импульс* для обозначения сигнала, достаточно короткого, для того чтобы быть импульсом для *любой возможной* системы. То есть сигнал, который имеет *бесконечно малую* ширину. **Непрерывная дельта функция** является нормализованной версией импульса этого типа. Конкретно, непрерывная дельта функция математически определяется тремя идеализированными характеристиками: (1) сигнал должен иметь бесконечно малую ширину, (2) импульс должен начинаться в момент времени равный нулю, (3) импульс должен иметь площадь равную единице.

Так как дельта функция определена как имеющая бесконечно малую ширину *и* фиксированную площадь, подразумевается, что ее амплитуда равна *бесконечности*. Не беспокойтесь об этом, это абсолютно не важно. Поскольку амплитуда является частью *формы* импульса, Вы никогда не столкнетесь с проблемой, где существует какая-нибудь разница, бесконечна амплитуда или нет. Дельта функция это математическая

конструкция, а не сигнал из реального мира. Сигналы в реальном мире, *действующие* как дельта функция, всегда будут иметь конечную длительность и амплитуду.

Так же, как и в случае дискретных сигналов, непрерывная дельта функция обозначается математическим символом $\delta(\ )$. Аналогично, выходной сигнал непрерывной системы в ответ на дельта функцию называется **импульсным откликом** и часто обозначается как $h(\ )$. Заметьте, что для обозначения непрерывных сигналов используются круглые скобки ( ), в отличие от квадратных скобок [ ], используемых для дискретных сигналов. Такое обозначение используется в этой книге и, наверное, где-нибудь еще в ЦОС, но оно не является универсальным. На графиках дельта импульсы показываются в виде вертикальных стрелок (см. рис. 13.1d) с *длинной* отражающей *площадь* импульса.

Для лучшего понимания дельта импульсов реального мира, найдите в ночном небе *планету* или *звезду*, например Марс или Сириус. Для невооруженного глаза обе выглядят примерно одной яркости и размера. Причина такой одинаковости не очевидна, поскольку геометрия рассматриваемых объектов радикально отличается. Марс около 6000 километров в диаметре и находится на расстоянии около 60 миллионов километров от Земли. Для сравнения, Сириус приблизительно в 300 раз больше и почти в миллион раз дальше. Такие размеры должны сделать Марс видимым в более чем *три тысячи* раз, огромнее, чем Сириус. Как же это возможно, что они выглядят одинаковыми?

Эти объекты выглядят одинаково потому, что они достаточно малы, как раз для того, чтобы стать *дельта импульсами* для системы зрения человека. Воспринимаемый образ это импульсный отклик глаза, а не действительное изображение звезды или планеты. Это становится очевидным, когда два объекта наблюдаются через небольшой телескоп: Марс выглядит как туманный диск, в то время как Сириус все еще виден как яркий дельта импульс. Это является также и причиной того, что звезды мерцают, а планеты нет. Изображение звезды достаточно мало, так что оно может быть кратковременно блокировано пылинками или турбулентностью в атмосфере, в то время как большее изображение планеты намного меньше подвергается такому воздействию.

Свертка

Так же, как и в случае дискретных сигналов, свертка непрерывных сигналов может быть рассмотрена со стороны *входного сигнала* или со стороны *выходного сигнала*. Точка зрения со стороны входа является наилучшим *концептуальным* описанием того, как работает свертка. Для сравнения, точка зрения со стороны выхода описывает *математику*, которую необходимо использовать. Эти описания фактически идентичны тем, что представлены в Главе 6 для дискретных сигналов.

Рис. 13.2 показывает, как свертка рассматривается со стороны входа. Входной сигнал $x(t)$ проходит через систему, характеризуемую импульсным откликом $h(t)$, производя выходной сигнал $y(t)$. Это может быть описано знакомым математическим уравнением: $y(t) = x(t)*h(t)$. Входной сигнал делится на узкие столбцы, настолько узкие, чтобы воздействовать на систему как *импульс*. Другими словами, входной сигнал раскладывается на бесконечное число смасштабированных и сдвинутых дельта функций. Каждый из этих импульсов производит смасштабированную и сдвинутую версию импульсного отклика в выходном сигнале. Тогда результирующий выходной сигнал эквивалентен эффекту объединения, т.е. сумме всех индивидуальных откликов.

Для такой схемы функционирования ширина столбцов должна быть намного короче, чем отклик системы. Конечно, математики доводят это до крайности, делая ширину входных сегментов *бесконечно малой*, поворачивая ситуацию к проблеме исчисления. В такой манере, точка зрения со стороны входа описывает то, как отдельная точка (или узкая область) во входном сигнале воздействует на большую часть выходного сигнала.

Для сравнения, точка зрения со стороны выхода рассматривает, как отдельная точка в выходном сигнале определяется различными значениями из входного сигнала. Так же, как и в случае дискретных сигналов, каждое мгновенное значение в выходном сигнале является результатом воздействия секции входных сигналов взвешенных посредством импульсного отклика перевернутого слева направо. В дискретном случае сигналы умножаются и *суммируются*. В непрерывном случае сигналы умножаются и *интегрируются*. В форме уравнения это выглядит как:

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau . \qquad (13.1)$$

Данное уравнение называют интегралом свертки, являющимся близнецом сверточной суммы (уравнение 6.1), используемой с дискретными сигналами. Рис. 13.3 показывает, как это уравнение может быть понято. Целью здесь является нахождение выражения для вычисления значения выходного сигнала в произвольное время *t*. Первым шагом является замена независимой переменной используемой для продвижения вдоль входного сигнала и импульсного отклика. То есть мы заменяем *t* на τ (прописная греческая буква тау). Это приводит к тому, что $x(t)$ и $h(t)$ становятся соответственно $x(\tau)$ и $h(\tau)$. Эта замена имен переменных необходима, поскольку *t* уже используется для представления точки в рассчитываемом выходном сигнале. Следующим шагом является переворот импульсного отклика слева направо, превращая его в $h(-\tau)$. Сдвиг перевернутого импульсного отклика к местоположению *t* приведет к тому, что выражение станет $h(t-\tau)$. Затем, входной сигнал взвешивается посредством перевернутого и сдвинутого импульсного отклика при помощи перемножения обоих, т.е. $x(\tau)h(t-\tau)$. После чего, интегрированием от минус до плюс бесконечности этого взвешенного входного сигнала в соответствии с уравнением 13.1, находится значение выходного сигнала.

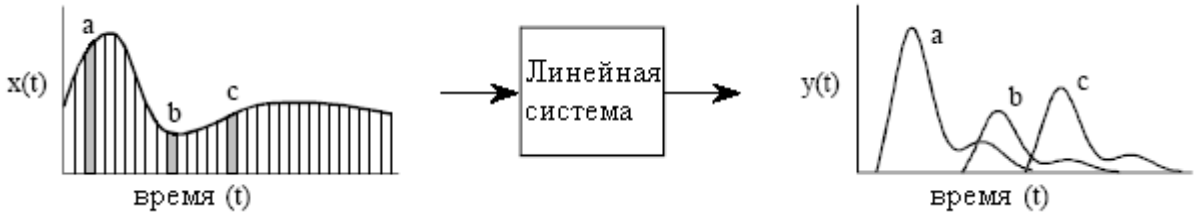


Рис. 13.2 Свертка, рассматриваемая со стороны входа

Если Вы испытываете трудности в понимании того, как все это работает, вернитесь назад и просмотрите туже концепцию для дискретных сигналов в Главе 6. Рис. 13.3 еще один путь описания машины свертки, приведенной на рис. 6.8. Единственное отличие состоит в том, что вместо суммирования используется интегрирование. Рассматривайте это не как что-то новое, а как расширение того, что Вы уже знаете.

Следующий пример проиллюстрирует то, как непрерывная свертка используется в проблемах реального мира, и какая при этом требуется математика. Рис. 13.4 показывает простую непрерывную линейную систему: электронный низкочастотный фильтр, состоящий из простого резистора и простого конденсатора. Как показано на рисунке импульс, входящий в эту систему, вызывает на выходе сигнал, который совершает резкий скачек до некоторого значения, а затем экспоненциально падает до нуля. Другими словами импульсный отклик этой простой электронной схемы это *односторонняя экспонента*. Математически импульсный отклик этой системы разбивается на две части, каждая из которых представляется уравнением:

$$h(t) = 0 \text{ для } t < 0;$$

$$h(t) = \alpha e^{-\alpha t} \text{ для } t \geq 0,$$

где $\alpha = 1/RC$ (R в омах, C в фарадах и t в секундах). Так же, как и в дискретном случае, непрерывный импульсный отклик содержит полную информацию о системе, т.е. о том, как она будет реагировать на все возможные сигналы. Продолжая этот пример дальше, рис. 13.5 показывает входящий в систему прямоугольный импульс математически описываемый как:

$$x(t) = 1 \text{ для } 0 \leq t \leq 1;$$

$$x(t) = 0 \text{ в противном случае.}$$

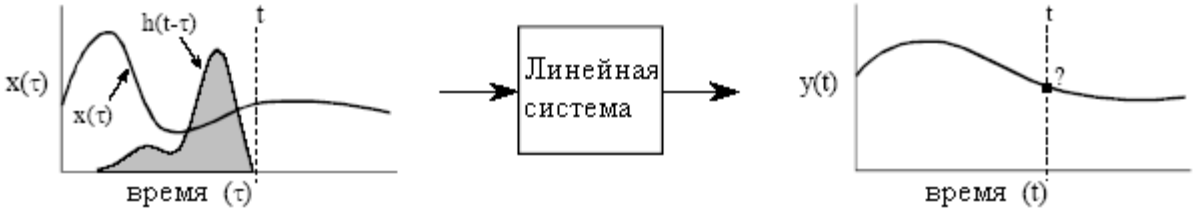


Рис. 13.3 Свертка, рассматриваемая со стороны выхода

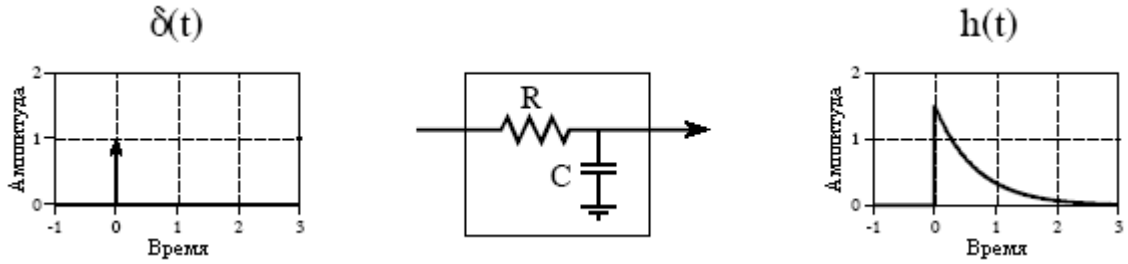


Рис. 13.4 Пример непрерывной линейной системы

Поскольку оба и входной сигнал, и импульсный отклик как математические выражения полностью известны, выходной сигнал $y(t)$ может быть вычислен оценкой интеграла свертки в соответствии с уравнением 13.1. Задача усложняется тем, что оба сигнала заданы *кусочно*, а не отдельным математическим выражением. Это очень типично для непрерывной обработки сигнала. Обычно немаловажным бывает графически изобразить то, как два сигнала смещаются друг относительно друга при различных значениях t. В этом примере, рис. 13.6а показывает, что два сигнала при $t<0$ не перекрываются вообще. Это означает, что произведение этих сигналов равно нулю в любой точке оси τ и результирующий сигнал:

$$y(t) = 0 \qquad \text{для } t<0.$$

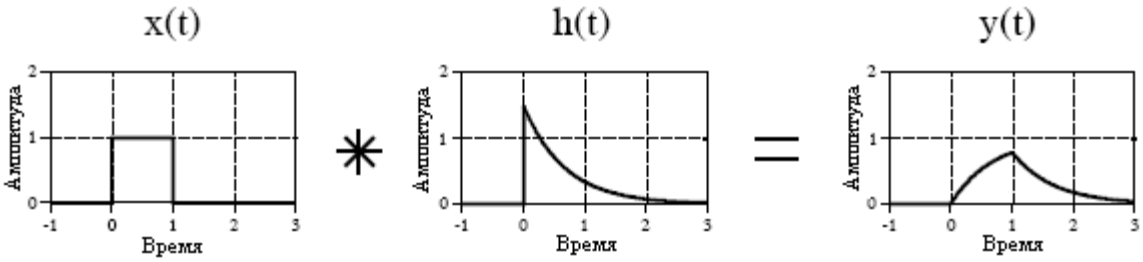


Рис. 13.5 Пример непрерывной свертки

Рис. 13.6 Кусочное вычисление свертки

Второй случай, где *t* находится между *0* и *1*, показан на рис. 136b. Здесь два сигнала частично перекрываются, это выражается в том, что их произведение между $\tau=0$ и $\tau=t$ имеет ненулевые значения. Поскольку это единственная ненулевая область, она является единственным участком, где необходимо вычислить интеграл. Для $0 \le t \le 1$ это дает выходной сигнал описываемый как:

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau \quad \text{(берем уравнение 13.1);}$$

$$y(t) = \int_{0}^{t} 1 \cdot \alpha e^{-\alpha(t-\tau)} d\tau \quad \text{(вставляем сигналы);}$$

$$y(t) = e^{-\alpha t}\left(e^{\alpha\tau}\right)\Big|_{0}^{t} \quad \text{(вычисляем интеграл);}$$

$$y(t) = e^{-\alpha t}\left(e^{\alpha t} - 1\right) \quad \text{(упрощаем) ;}$$

$$y(t) = 1 - e^{-\alpha t} \quad \text{(для } 0 \le t \le 1\text{).}$$

Рис. 13.6c показывает вычисление третьего участка выходного сигнала, где $t>1$. Здесь перекрытие имеет место между $\tau=0$ и $\tau=1$, изменяя пределы интегрирования, проводим вычисления таким же образом, как и для второго участка:

$$y(t)=\int_0^1 1\cdot\alpha e^{-\alpha(t-\tau)}d\tau$$ (вставляем сигналы в уравнение 13.1);

$$y(t)=e^{-\alpha t}\left(e^{\alpha\tau}\right)\Big|_0^1$$ (вычисляем интеграл);

$$y(t)=\left(e^{\alpha}-1\right)e^{-\alpha t}$$ для $t>1$.

Форма волны на каждом из этих трех сегментов вполне согласуется с Вашими знаниями из электроники: (1) Выходной сигнал должен быть равен нулю до тех пор, полка входной сигнал не станет отличным от нуля. (2) Когда на входе появляется ступенчатый сигнал, сигнал на выходе *RC* цепи экспоненциально нарастает в соответствии с уравнением: $y(t)=1-e^{-\alpha t}$. (3) Когда сигнал на входе возвращается к значению нуля, выходной сигнал экспоненциально падает до нуля согласно уравнению: $y(t)=ke^{-\alpha t}$ (где $k=e^{\alpha}-1$ - напряжение на конденсаторе перед тем, как начался его разряд).

Подобным же образом могут быть обработаны и более сложные формы волн, хотя математическая сложность может быстро стать неуправляемой. При столкновении с неприятной задачей непрерывной свертки Вы должны потратить значительное время, оценивая *стратегии* решения задачи. Если Вы начинаете вслепую оценивать интегралы, Вы, вероятно, закончите математическим беспорядком. Обычной стратегией будет разбить один из сигналов на простые суммарные составляющие, которые *индивидуально* могут быть подвергнуты операции свертки. Для нахождения ответа на первоначально поставленную задачу, используя принципы линейности, результирующие формы волн затем могут быть сложены.

На рис. 13.7 показана другая стратегия: модифицируйте один из сигналов некоторым линейным образом, выполните операцию свертки, а затем отмените первоначальную модификацию. В этом примере модификация это *производная*, а ее отмена это *интегрирование*. Производная от прямоугольного импульса единичной амплитуды - это два коротких *импульса бесконечно малой длительности*, первый в области единицы и второй в области минус единицы. Для того чтобы понять это, подумайте о противоположном процессе, взятии интеграла от этих двух импульсов. По мере того как Вы проводите интегрирование, после первого импульса интеграл быстро увеличивается от нуля до единицы, т.е. ступенчатая функция. После прохождения отрицательного импульса интеграл сигнала быстро возвращается от единицы обратно к нулю, завершая прямоугольный импульс.

Взятие производной упрощает эту задачу, поскольку свертка выполняется легче, когда один из сигналов состоит из импульсов бесконечно малой длительности. Вклад каждого из двух импульсов в $x'(t)$ - это смасштабированная и сдвинутая версия импульсного отклика в производной выходного сигнала $y'(t)$. То есть, из непосредственного рассматривания рис. 13.7, видно, что: $y'(t)=h(t)-h(t-1)$. Тогда выходной сигнал $y(t)$ может быть найден подстановкой сюда точного уравнения для $h(t)$ и интегрированием выражения.

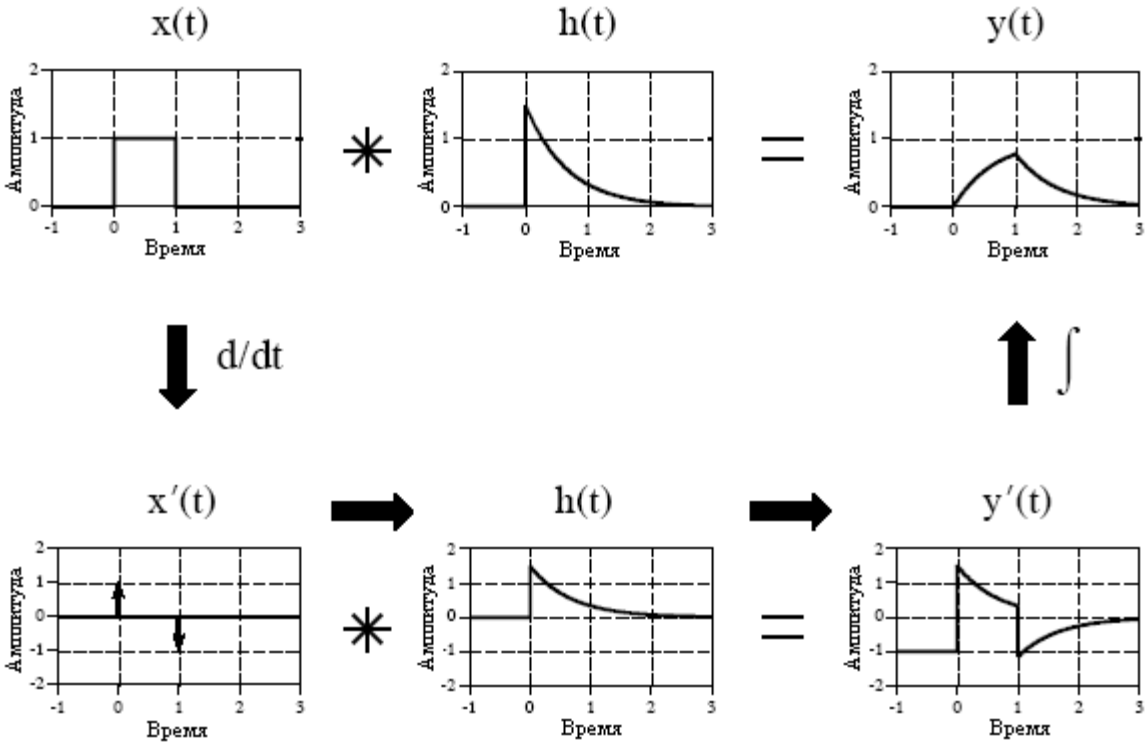


Рис. 13.7 Стратегия свертки сигналов

Небольшой неприятностью в этой процедуре является то, что значение постоянной составляющей входного сигнала при взятии производной теряется. Это может дать ошибку в значении постоянной составляющей вычисленного выходного сигнала. Математика отражает данный факт с помощью произвольной константы, которая должна быть добавлена во время интегрирования. Систематического пути идентификации такой ошибки нет, но обычно она может быть скорректирована из непосредственного рассматривания проблемы. Например, в примере на рис. 13.7 ошибки постоянной составляющей нет. Это очевидно, поскольку вычисленный выходной сигнал, при *t* становящимся очень большим, имеет правильное значение постоянной составляющей. Если в конкретной проблеме присутствует ошибка, для завершения вычислений соответствующая постоянная составляющая вручную добавляется к выходному сигналу.

Этот метод работает так же с сигналами, которые могут быть приведены к коротким импульсам с бесконечно малой длительностью *многократным* взятием производной. На жаргоне области такие сигналы называются *кусочными полиномами*. После выполнения операции свертки многократное взятие производной отменяется многократным интегрированием. Единственной уловкой является то, что потерянное значение постоянной составляющей должно быть найдено на каждом этапе за счет отыскания правильного значения постоянной интегрирования.

Перед тем как приступить к проблеме трудной непрерывной свертки, существует еще один подход, который Вам следует рассмотреть. Задайте себе вопрос: *Действительно ли для выходного сигнала необходимо математическое выражение или достаточно графика формы волны*? Если график адекватен, Вам может быть выгоднее решить проблему с помощью *дискретных* методов. То есть, аппроксимировать непрерывный сигнал отсчетами, с которыми операция свертки может быть выполнена непосредственно при помощи компьютерной программы. Хотя это математически не строго, это может быть значительно проще.

Преобразование Фурье

Преобразование Фурье для непрерывных сигналов делится на две категории: одна для *периодических* сигналов, другая для *непериодических*. Для периодических сигналов используется вариант преобразования Фурье, называемый **ряды Фурье**, который обсуждается в следующем разделе. Преобразование Фурье, используемое для непериодических сигналов, просто называют **преобразование Фурье**. Эта глава описывает данные техники Фурье, пользуясь только *действительной* математикой, точно так же, как это было сделано для дискретных сигналов в нескольких предыдущих главах. Использование более мощной *комплексной* математики будет оставлено для Главы 31.

На рис. 13.8 показан пример непрерывного непериодического сигнала и его частотный спектр. Во временной области сигнал простирается от минус бесконечности до плюс бесконечности, в то время как в каждой из частотных областей сигнал простирается от нуля до плюс бесконечности. Этот частотный спектр показан в прямоугольных координатах (действительная и мнимая части), однако для непрерывных сигналов используются также и полярные координаты (амплитуда и фаза). Так же, как и в дискретном случае, **уравнение синтеза** содержит рецепт построения сигнала во временной области по данным частотной области. В математической форме:

$$x(t) = \frac{1}{\pi} \int_{0}^{+\infty} \mathrm{Re}\, X(\omega)\cos(\omega t) - \mathrm{Im}\, X(\omega)\sin(\omega t)\, d\omega . \qquad (13.2)$$

Выражаясь словами, сигнал во временной области формируется за счет суммирования (с помощью интеграла) бесконечного числа смасштабированных синусоидальных и косинусоидальных волн. Действительная часть частотной области состоит из масштабирующих коэффициентов косинусоидальных волн, в то время как мнимая часть состоит из масштабирующих коэффициентов синусоидальных волн. Так же, как и в случае с дискретными сигналами, уравнение синтеза обычно записывается с использованием *отрицательных* синусоидальных волн. Хотя в этом обсуждении отрицательный знак не имеет значения, он необходим, чтобы сделать систему обозначений совместимой с комплексной математикой описанной в Главе 29. Главное, о чем нужно помнить - то, что некоторые авторы ставят этот отрицательный знак в уравнение, тогда как другие нет. Заметим также, что частота обозначена символом ω строчная греческая буква омега. Как Вы помните, это обозначение называется **естественной частотой** и измеряется в радианах в секунду. То есть $\omega = 2\pi f$, где f – частота в колебаниях в секунду (герц). Обозначение естественной частоты нравится математикам и всем, кто использует обработку сигналов для *решения уравнений* в силу того, что для записи обычно требуется меньшее количество символов.

Уравнения анализа для непрерывных сигналов следуют такой же стратегии, как и в случае с дискретными сигналами: *корреляция* с синусоидальными и косинусоидальными волнами. Эти уравнения следующие:

$$\mathrm{Re}\, X(\omega) = \int_{-\infty}^{+\infty} x(t)\cos(\omega t)\, dt$$

$$\mathrm{Im}\, X(\omega) = -\int_{-\infty}^{+\infty} x(t)\sin(\omega t)\, dt . \qquad (13.3)$$

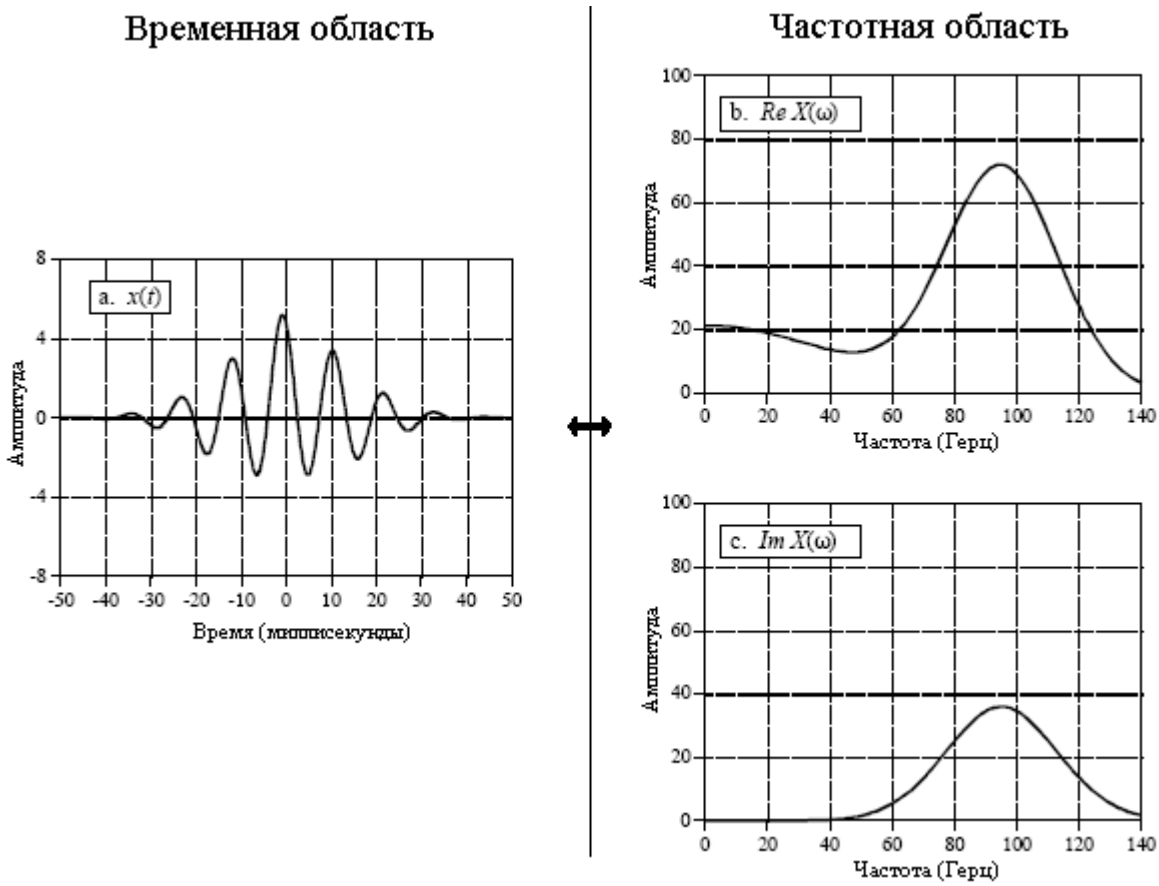


Рис. 13.8 Пример преобразования Фурье

В качестве примера использования уравнений анализа найдем частотный отклик *RC* фильтра низких частот. Это может быть сделано, если взять преобразование Фурье от импульсного отклика *RC* фильтра низких частот, ранее показанного на рис. 13.4 и описываемого как:

$$h(t)=0 \qquad \text{для} \qquad t<0$$

$$h(t)=\alpha e^{-\alpha t} \qquad \text{для} \qquad t\geq 0.$$

Частотный отклик находится подстановкой импульсного отклика в уравнения анализа. Прежде всего, действительная часть:

$$\mathrm{Re}\,H(\omega)=\int_{-\infty}^{+\infty} h(t)\cos(\omega t)dt \qquad \text{(начните с уравнения 13.3)}$$

$$\mathrm{Re}\,H(\omega)=\int_{0}^{+\infty} \alpha e^{-\alpha t}\cos(\omega t)dt \qquad \text{(подставьте сигнал)}$$

$$\mathrm{Re}\,H(\omega) = \frac{\alpha e^{-\alpha t}}{\alpha^2 + \omega^2}\left[-\alpha\cos(\omega t) + \omega\sin(\omega t)\right]\Big|_0^{+\infty}$$ (вычислите)

$$\mathrm{Re}\,H(\omega) = \frac{\alpha}{\alpha^2 + \omega^2}.$$

Используя тот же самый подход, вычисленная мнимая часть частотного отклика будет:

$$\mathrm{Im}\,H(\omega) = \frac{-\omega\alpha}{\alpha^2 + \omega^2}.$$

Так же, как и в случае с дискретными сигналами представление частотной области в прямоугольной системе координат великолепно для математических преобразований, но тяжело для человеческого восприятия. Ситуация может быть исправлена преобразованием в полярную систему координат с помощью стандартных соотношений:

$$MagH(\omega) = \sqrt{\mathrm{Re}\,H(\omega)^2 + \mathrm{Im}\,H(\omega)^2}$$

и

$$PhaseH(\omega) = \arctan\left(\mathrm{Im}\,H(\omega) \Big/ \mathrm{Re}\,H(\omega)\right).$$

Проделывание алгебраических преобразований дает частотный отклик *RC* фильтра низких частот как амплитуду и фазу (т.е. в полярном виде):

$$MagH(\omega) = \frac{\alpha}{\sqrt{\alpha^2 + \omega^2}}$$

$$PhaseH(\omega) = \arctan\left(-\frac{\omega}{\alpha}\right)$$

На рис. 13.9 показаны графики этих кривых для частоты среза *1000* герц (т.е. $\alpha=2\pi 1000$).

Ряды Фурье

Этот материал приведет нас к последнему члену семейства преобразований Фурье – *рядам Фурье*. Сигнал, используемый в рядах Фурье, во временной области является *периодическим* и *непрерывным*. На рис. 13.10 показано несколько примеров непрерывных форм волн, которые повторяют сами себя от минус до плюс бесконечности (обратите внимание на то, что *A* на рис. 13.10b, рис. 13.10c, рис. 13.10d представляет собой размах колебания, а не амплитуду, как это чаще всего принято – прим. перев.). В Главе 11 было показано, что периодические сигналы имеют частотный спектр, состоящий из **гармоник**.

Например, если сигнал во временной области повторяется с частотой 1000 герц (период 1 миллисекунда), частотный спектр будет содержать первую гармонику с частотой 1000 герц, вторую гармонику с частотой 2000 герц, третью гармонику с частотой 3000 герц и т.д. Первую гармонику, т.е. частоту, с которой повторяется сигнал во временной области, еще называют **основной частотой**. Это означает, что частотный спектр может быть рассмотрен двумя способами: (1) частотный спектр является *непрерывным*, но равен нулю на всех частотах кроме частот гармоник, или (2) частотный спектр *дискретный* и *определен* только на частотах гармоник. Другими словами члены с частотами между гармониками могут рассматриваться, как имеющие нулевую амплитуду или просто несуществующими. Важно то, что они не вносят вклада в формирование сигнала во временной области.

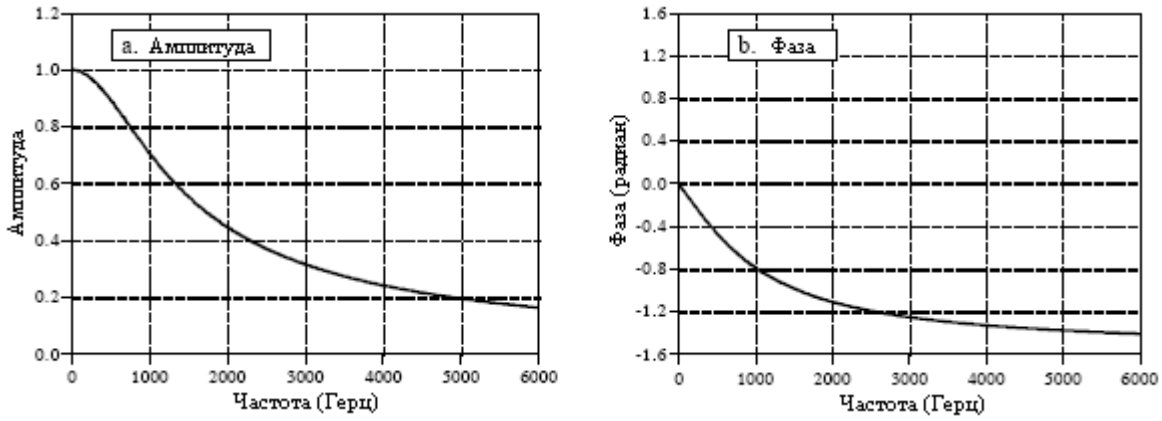


Рис. 13.9 Частотная характеристика низкочастотного *R-C* фильтра

Уравнение синтеза рядов Фурье создает непрерывный периодический сигнал с основной частотой *f*, суммируя смасштабированные косинусные и синусные волны с частотами: *f*, *2f*, *3f*, *4f* и т.д. Амплитуды косинусных волн содержатся в переменных: a_1, a_2, a_3, a_4, и т.д., в то время как амплитуды синусных волн содержатся в b_1, b_2, b_3, b_4, и т.д. Другими словами, коэффициенты "*a*" и "*b*" – это соответственно действительная и мнимая части частотного спектра. Кроме того коэффициент a_0 используется для хранения значения постоянной составляющей сигнала во временной области. Этот коэффициент может рассматриваться как амплитуда косинусной волны с нулевой частотой (постоянная величина). Иногда a_0 объединяется с другими коэффициентами "*a*", но чаще всего он обрабатывается отдельно, поскольку он требует специальных вычислений. Коэффициента b_0 нет, так как синусная волна с нулевой частотой имеет постоянное значение равное нулю, и он был бы совсем бесполезным. Уравнение синтеза записывается как:

$$x(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(2\pi f t n) - \sum_{n=1}^{\infty} b_n \sin(2\pi f t n). \qquad (13.4)$$

Соответствующие **уравнения анализа** для рядов Фурье обычно записываются в терминах периода, обозначаемого как *T*, а не в терминах основной частоты *f* (где $f=1/T$). Поскольку сигнал во временной области является периодическим, корреляцию синусной и косинусной волн нужно оценить только на одном единственном периоде, т.е. от $-T/2$ до $T/2$, или от 0 до T, или от $-T$ до 0 и т.д. Выбор разных пределов разнообразит математику, но конечный ответ всегда одинаков. Уравнения анализа рядов Фурье следующие:

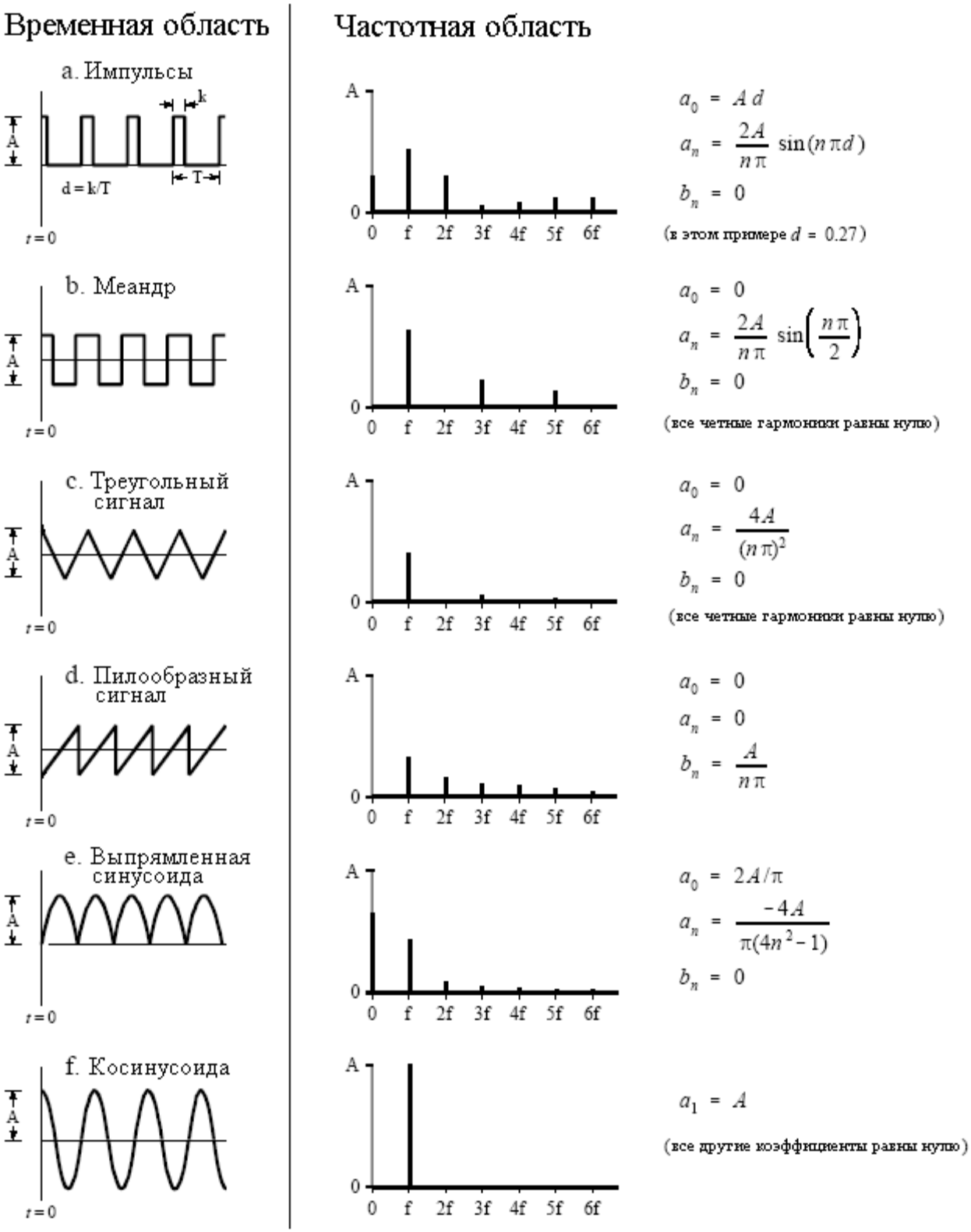


Рис. 13.10 Примеры рядов Фурье

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} x(t)dt;$$

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} x(t)\cos\left(\frac{2\pi tn}{T}\right)dt \qquad (13.5)$$

$$b_n = \frac{-2}{T}\int_{-T/2}^{T/2} x(t)\sin\left(\frac{2\pi tn}{T}\right)dt$$

На рис. 13.11 показан пример для вычисления ряда Фурье с использованием этих уравнений. Анализируемый сигнал во временной области представляет собой *последовательность импульсов* прямоугольной формы с неодинаковой длительностью его верхних и нижних уровней. На протяжении одного периода форма волны описывается как:

$x(t)=A$ для $-k/2 \le t \le k/2;$

$x(t)=0$ в противном случае.

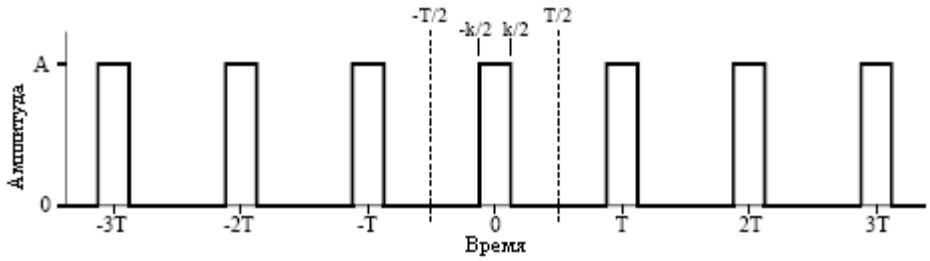


Рис. 3.11 Пример для вычисления ряда Фурье

Таким образом, к*оэффициент заполнения* формы волны (доля периода, на которой импульс находится в состоянии высокого уровня) определяется как $d=k/T$. Коэффициенты ряда Фурье могут быть найдены через вычисления уравнений 13.5. Прежде всего, найдем постоянную составляющую a_0:

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} x(t)dt \qquad \text{(начните с уравнений 13.5)}$$

$$a_0 = \frac{1}{T}\int_{-k/2}^{k/2} A\,dt \qquad \text{(подставьте сигнал)}$$

$$a_0 = \frac{Ak}{T} \qquad \text{(вычислите интеграл)}$$

$$a_0 = Ad \qquad \text{(замените: } d=k/T\text{)}$$

Этот результат нужно интуитивно почувствовать: постоянная составляющая является просто средним значением сигнала. Подобный анализ дает и коэффициенты "*a*".

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} x(t)\cos\left(\frac{2\pi tn}{T}\right)dt \qquad \text{(начните с уравнений 13.5)}$$

$$a_n = \frac{2}{T}\int_{-k/2}^{k/2} A\cos\left(\frac{2\pi tn}{T}\right)dt \qquad \text{(подставьте сигнал)}$$

$$a_n = \frac{2A}{T}\left[\frac{T}{2\pi n}\sin\left(\frac{2\pi tn}{T}\right)\right]\Bigg|_{-k/2}^{k/2} \qquad \text{(вычислите интеграл)}$$

$$a_n = \frac{2A}{n\pi}\sin(\pi nd) \qquad \text{(упростите)}$$

Коэффициенты "*b*" вычисляются таким же образом, однако в данном примере все они равны *нулю*. Другими словами данную форму волны можно построить, используя только косинусные волны, в синусных волнах необходимости нет.

Если форма волны смещается во временной области влево или вправо, коэффициенты "*a*" и "*b*" будут меняться. Например, коэффициенты "*b*" в этом примере будут равны нулю *только в том случае*, когда центр одного из импульсов находится на линии *t=0*. Рассуждайте об этом следующим образом. Если форма волны *четная* (т.е. симметрична относительно *t=0*), она будет состоять исключительно из четных синусоид, т.е. косинусоид. Это делает равными нулю все коэффициенты "*b*". Если форма волны *нечетная* (т.е. симметрична, но противоположна по знаку относительно *t=0*), она будет состоять из нечетных синусоид, т.е. синусоидальных волн. Это приводит к тому, что коэффициенты "*a*" становятся равными нулю. Если коэффициенты преобразовать к полярным обозначениям (скажем к коэффициентам M_n и Θ_n), сдвиг сигнала во временной области оставит амплитуду неизмененной, но добавит линейную компоненту к фазе.

Чтобы завершить этот пример, представьте существующую в электронной схеме последовательность импульсов с частотой 1 кГц, амплитудой один вольт и

коэффициентом заполнения *0,2*. В таблице на рис. 13.12 приведены амплитуды каждой гармоники, содержащейся в волне такой формы. На рис. 13.12 показана также форма волны, синтезированная с использованием только *первых четырнадцати* из этих гармоник. Даже с таким числом гармоник реконструкция не очень хороша. На математическом жаргоне ряд Фурье *сходится* очень *медленно*. Это просто другой способ выражения того, что крутые фронты в форме волны во временной области приводят к очень высоким частотам в спектре. Наконец, убедитесь в существовании и посмотрите на выбросы на крутых фронтах, то есть, на эффект Гиббса, обсужденный в Главе 11.

Одним из важных применений рядов Фурье является электронное **умножение частоты**. Предположим, что Вы хотите построить очень стабильный синусоидальный генератор на 150 МГц. Это могло бы понадобиться, например, для радиопередатчика работающего на этой частоте. Высокая стабильность требует, чтобы частота в схеме задавалась *кварцевым резонатором*. То есть частота генератора задается резонирующим кристаллом кварца, являющимся частью схемы. Проблема заключается в том, что кварцевые кристаллы работают только на частотах приблизительно до 10 МГц. Решение заключается в том, чтобы построить генератор с частотой, задаваемой пьезокристаллом, работающий где-то между 1 и 10 МГц, а затем *умножить* частоту до нужного Вам значения. Это достигается *искажением* синусоидальной волны, например, ограничением ее пиков с помощью диода или пропусканием волны через цепь, формирующую прямоугольную форму. Затем с помощью полосового фильтра из искаженной формы волны выделяют гармоники. Это позволяет удвоить, утроить частоту или умножить ее на более высокие целые числа. Наиболее обычным способом для генерирования требуемого умножения частоты является использование не просто однокаскадных, а последовательных многокаскадных удвоителей и утроителей. Ряды Фурье очень важны для проектов такого типа, поскольку они описывают зависимость амплитуды умноженного сигнала от вида искажения и отобранной гармоники.

Частота	Амплитуда (вольты)
Пост. составл	0.20000
1 kHz	0.37420
2 kHz	0.30273
3 kHz	0.20182
4 kHz	0.09355
5 kHz	0.00000
6 kHz	-0.06237
7 kHz	-0.08649
8 kHz	-0.07568
9 kHz	-0.04158
10 kHz	0.00000
11 kHz	0.03402
12 kHz	0.05046
⋮	
123 kHz	0.00492
124 kHz	0.00302
125 kHz	0.00000
126 kHz	-0.00297
⋮	
803 kHz	0.00075
804 kHz	0.00046
805 kHz	0.00000
806 kHz	-0.00046

Рис. 13.12 Пример синтеза по ряду Фурье

ГЛАВА
14

ВВЕДЕНИЕ В ЦИФРОВЫЕ ФИЛЬТРЫ

Использование цифровых фильтров преследует две основные цели: (1) разделение смешанных сигналов, и (2) восстановление сигналов, которые некоторым образом были искажены. Аналоговые (электронные) фильтры могут использоваться для решения тех же самых задач, однако цифровые фильтры позволяют достигать более значительных результатов. В следующих семи главах описываются и сравниваются наиболее популярные цифровые фильтры. Эта вводная глава описывает параметры, которые Вы желаете определить при изучении каждого из этих фильтров.

Базис фильтров

Цифровые фильтры являются очень важной частью ЦОС. Фактически их экстраординарная работа является одной из ключевых причин, по которой ЦОС стала так популярна. Как отмечалось во введении, существует два применения фильтров: *выделение* сигнала и *восстановление* сигнала. Выделение сигнала необходимо тогда, когда сигнал был искажен интерференцией, шумом или другими сигналами. Например, представьте прибор для измерения электрической активности сердца ребенка (ЭКГ), находящегося в утробе матери. Исходный сигнал, вероятно, будет искажен дыханием и сердцебиением матери. Фильтр мог бы быть использован для разделения этих сигналов таким образом, чтобы они могли быть проанализированы индивидуально.

Восстановление сигнала используется тогда, когда сигнал некоторым образом был искажен. Например, звукозапись, сделанная на оборудовании низкого качества, может быть отфильтрована для наилучшего представления звука в том виде, в котором он извлекался. Другим примером является устранение размытости изображения, возникающего при ненадлежащим образом сфокусированной линзе или дрожащей камере.

Эти проблемы могут быть устранены как с помощью аналоговых, так и с помощью цифровых фильтров. Какой же из них лучше? Аналоговые фильтры дешевле, быстрее и имеют большой динамический диапазон, как по амплитуде, так и по частоте. Цифровые фильтры в сравнении намного превосходят по уровню достижимых характеристик. Например, низкочастотный цифровой фильтр представленный в Главе 16 обладает усилением порядка 1±0,0002 от постоянной составляющей до 1000 герц и усилением менее чем 0,0002 для частот свыше 1001 герц. Весь переход происходит в пределах всего в 1 герц. Не ждите этого от схем на операционных усилителях! Цифровые фильтры могут достигать в *тысячи* раз лучших характеристик, чем аналоговые. Это создает поразительную разницу в подходах к проблемам фильтрации. С аналоговыми фильтрами акценты направлены на разрешение ограничений, связанных с электроникой, таких как точность и стабильность резисторов и конденсаторов. Для сравнения, цифровые фильтры настолько хороши, что характеристики фильтра часто игнорируется. Акценты смещаются к ограничениям *сигналов* и *теоретическим* проблемам, связанным с их обработкой.

Для ЦОС совершенно типично говорить, что входной и выходной сигналы фильтра находятся во *временной области*. Это оттого, что сигналы обычно создаются с помощью снятия отсчетов через равные промежутки *времени*. Но это не единственный способ, с

помощью которого может быть осуществлено снятие отсчетов. Вторым наиболее типичным способом снятия отсчетов является снятие отсчетов через равные интервалы в *пространстве*. Например, представьте снятие одновременных показаний от ряда тензодатчиков, установленных через один сантиметр вдоль крыла самолета. Возможны и множества других областей, однако, время и пространство являются наиболее типичными. Когда в ЦОС Вы встречаете термин *временная область*, помните, что это может фактически относиться к взятым во времени отсчетам, или это может быть общая ссылка на любую область, в которой осуществляется снятие отсчетов.

Как показано на рис. 14.1, каждый линейный фильтр обладает **импульсной характеристикой**, **переходной характеристикой** и **частотной характеристикой**. Каждая из этих характеристик содержит полную информацию о фильтре, но в различной форме. Если одна из этих трех характеристик определена, то две другие являются заданными и могут быть непосредственно вычислены. Все три этих представления являются важными, поскольку они описывают то, какова будет реакция фильтра при различных обстоятельствах.

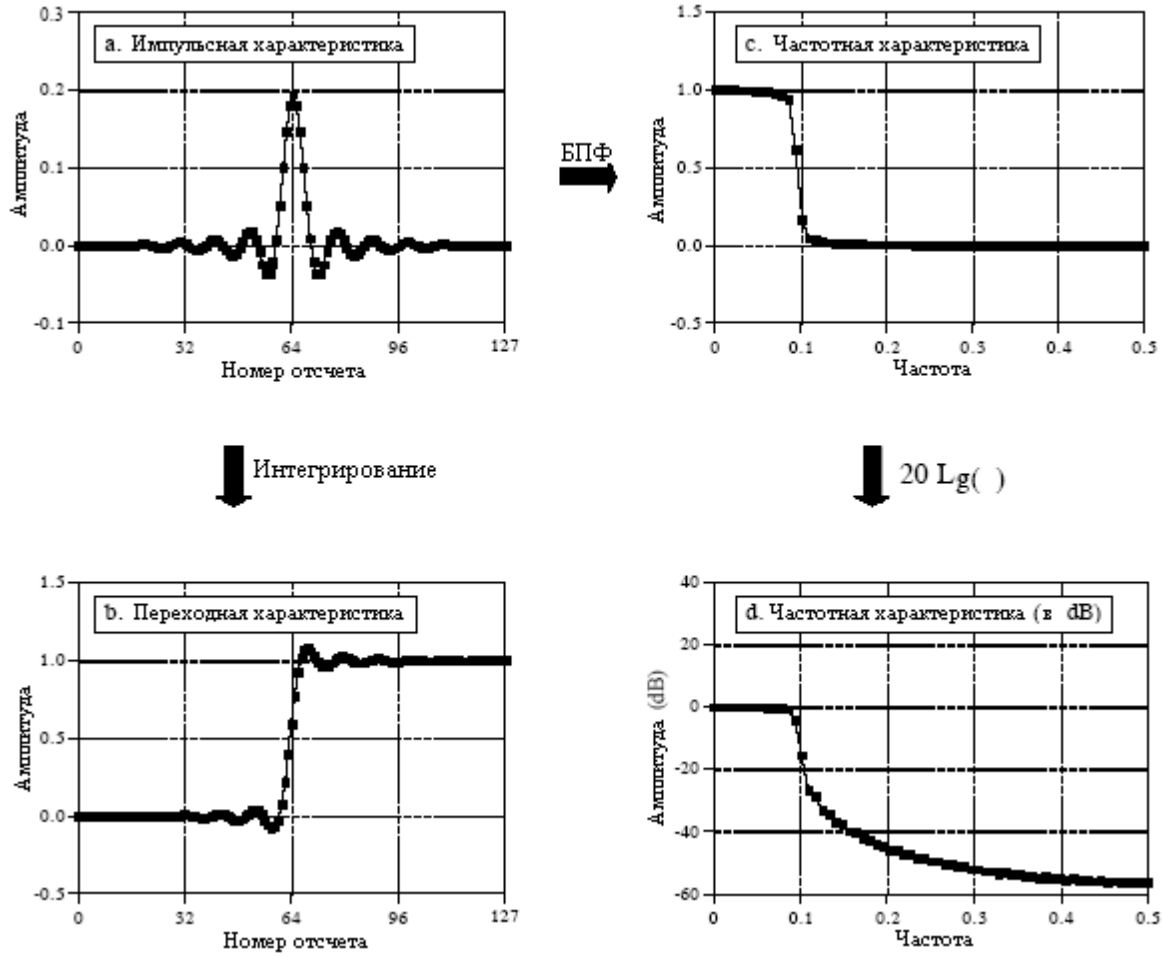


Рис. 14.1 Параметры фильтров

Наиболее прямым способом реализации цифрового фильтра является *свертка* входного сигнала с *импульсной характеристикой* цифрового фильтра. Таким манером могут быть сделаны все возможные фильтры. (Это должно быть очевидным. Если это не так, то вероятно у Вас нет достаточной подготовки для понимания этого раздела по проектированию фильтров. Попробуйте снова пересмотреть предыдущие разделы по основам ЦОС.) Когда импульсная характеристика используется в такой манере, разработчики фильтров дают ей специальное название: **ядро фильтра**.

Существует также другой способ создания цифровых фильтров, называемый **рекурсией**. Когда фильтр реализуется с помощью свертки, каждый отсчет в выходном сигнале вычисляется с помощью *взвешивания* входных отсчетов и сложения их вместе. Рекурсивные фильтры являются расширением этого, использующим кроме точек из *входного* сигнала ранее вычисленные значения *выходного* сигнала. Вместо использования ядра фильтра, рекурсивные фильтры определяются при помощи набора **рекурсивных коэффициентов**. Детально этот метод будет обсужден в Главе 19. А сейчас важным моментом является то, что все линейные фильтры обладают импульсной характеристикой, даже в том случае, если Вы не используете ее для реализации фильтра. Для нахождения импульсной характеристики рекурсивного фильтра просто подайте на вход импульс и посмотрите, что появится на выходе. Импульсные характеристики рекурсивных фильтров состоят из экспоненциально затухающих по амплитуде синусоид. В принципе это делает их импульсные характеристики *бесконечно протяженными*. Однако амплитуда, в конечном счете, снижается ниже шума округления системы, и остающиеся отсчеты могут быть игнорированы. По причине такой характеристики рекурсивные фильтры также называют фильтрами с **бесконечной импульсной характеристикой** или **БИХ-**фильтрами. Для сравнения, фильтры, реализуемые с помощью свертки, называются фильтрами с **конечной импульсной характеристикой** или **КИХ**-фильтрами.

Как Вы знаете, *импульсной характеристикой* системы является сигнал на ее выходе тогда, когда на ее вход подан *импульс*. Таким же образом *переходная характеристика* является сигналом на выходе, когда на вход подан *ступенчатый* сигнал (именуемые также соответственно *откликом на фронт* и *фронтом*). Поскольку ступенчатый сигнал является интегралом импульсного сигнала, переходная характеристика является интегралом импульсной характеристики. Это дает возможность находить переходную характеристику двумя способами: (1) подавать на фильтр волну ступенчатой формы и наблюдать, что поступает с выхода, или (2) проинтегрировать импульсную характеристику. (Чтобы быть математически точными: *интегрирование* используется с непрерывными сигналами, тогда как с дискретными сигналами используется *дискретное интегрирование*, т.е. бегущая сумма.) Частотная характеристика может быть найдена взятием ДПФ (используя алгоритм БПФ) от импульсной характеристики. Это будет рассмотрено позже в данной главе. Графически частотная характеристика может быть показана на линейной вертикальной оси, как показано на рис. 14.1c, или в логарифмическом масштабе (децибелы), как показано на рис. 14.1d. Линейная шкала является наилучшей для изображения пульсаций в полосе пропускания и завала частотной характеристики, тогда как шкала децибел необходима для изображения ослабления в полосе подавления.

Не помните децибелы? Вот краткий обзор. **Бел** (в честь Александра Грэма Бела) означает, что мощность изменилась на *множитель десять*. Например, электронная схема, имеющая усиление в 3 бела, вырабатывает выходной сигнал 10×10×10 = 1000 в тысячу раз мощнее, чем входной. **Децибел** (**dB**) - это одна десятая от бела. Следовательно, значения в децибелах: -20dB, -10dB, 0dB, 10dB, 20dB означают соответственно отношения мощностей 0,01, 0,1, 1, 10, и 100. Другими словами, каждые *десять* децибел означают, что мощность изменилась на множитель десять.

А вот и ловушка: обычно Вы хотите работать с *амплитудой* сигналов, а не с их *мощностью*. Например, представьте усилитель с коэффициентом усиления 20dB. По определению это означает, что мощность сигнала увеличилась на множитель 100. Поскольку амплитуда пропорциональна корню квадратному от мощности, амплитуда на выходе отличается от амплитуды на входе в 10 раз. Хотя по мощности 20dB означает множитель 100, это всего означает множитель 10 по амплитуде. Каждые *двадцать* dB означают, что амплитуда изменилась на множитель десять. В форме уравнения:

$$dB = 10\log_{10}\frac{P_2}{P_1};$$
$$dB = 20\log_{10}\frac{A_2}{A_1}. \quad (14.1)$$

Приведенные выше выражения используют логарифм по основанию десять, однако, многие языки программирования обеспечивают только функцией логарифма по основанию *e* (натуральный логарифм записываемый как $log_e\ x$ или $ln\ x$). Натуральный логарифм может быть использован для модификации выше приведенных выражений: dB=4,342945 $log_e\ (P_2/P_1)$ и dB=8,685890 $log_e\ (A_2/A_1)$.

Поскольку децибелы являются способом выражения отношения между двумя сигналами, они идеальны для описания усиления системы, т.е. отношения между выходным и входным сигналом. Однако инженеры используют децибелы также для определения амплитуды (или мощности) *одного единственного* сигнала, относя его к некоторому стандарту. Например, термин **dBV** означает, что сигнал соотносится с сигналом со среднеквадратичным значением в 1 вольт. Аналогично, **dBm** указывает на соотносимый сигнал, вырабатывающий 1 мВт на нагрузке в 600 Ом (среднеквадратичное значение около 0,78 вольт).

Если Вы уже больше ничего не понимаете о децибелах, запомните две вещи. Во-первых, -3dB означает, что амплитуда уменьшилась до 0,707 (и следовательно мощность уменьшилась до 0,5). Во-вторых, запомните следующие преобразования между децибелами и отношениями *амплитуды*:

60dB=1000;
40dB=100;
20dB=10;
0dB=1;
-20dB=0,1;
-40dB=0,01;
-60dB=0,001.

Как информация представлена в сигналах

Наиболее важной частью любой задачи ЦОС является понимание того, каким образом в сигнале, с которым Вы работаете, содержится *информация*. Существует большое число способов, которыми информация может содержаться в сигнале. Это особенно верно для искусственных сигналов. Например, взглянете на все изобретенные схемы модуляции: амплитудная модуляция, частотная модуляция, однополосная модуляция, импульсно-кодовая модуляция, широтно-импульсная модуляция и т.д. Список можно продолжать и продолжать. К счастью, существуют только два типичных способа, которыми информация может быть представлена в естественно встречающихся сигналах. Мы будем называть их: **информация представлена во временной области** и **информация представлена в частотной области**.

Информация представленная во временной области описывает, в какой момент происходит нечто и какова амплитуда происходящего. Например, представьте эксперимент по изучению светоотдачи от солнца. Один раз в течение каждой секунды измеряется и записывается светоотдача. Каждый отсчет в сигнале показывает то, что происходит в это мгновение и величину события. Если происходит солнечная вспышка,

сигнал непосредственно обеспечивает информацией о времени, когда она произошла, ее продолжительности, развитии во времени и т.д. Каждый отсчет содержит информацию, интерпретируемую безотносительно к любому другому отсчету. Если даже у Вас есть всего один отсчет из этого сигнала, Вы все же знаете, кое-что о том, что Вы измеряете. Это самый простой способ для информации содержаться в сигнале.

Напротив, информация представленная в частотной области является более косвенной. Большое число вещей в нашей вселенной показывает периодическое движение. Например, бокал для вина, по которому постучали ногтем, будет вибрировать, издавая звенящий звук; маятник дедушкиных часов раскачивается взад и вперед; звезды и планеты вращаются вокруг своих осей и друг относительно друга и так далее. Измеряя частоту фазу и амплитуду этих периодических движений, очень часто может быть получена информация о системе производящей движение. Предположим, мы осуществляем снятие отсчетов звука производимого звенящим винным бокалом. Основная частота и гармоники периодической вибрации соотносятся с массой и эластичностью материала. Один единственный отсчет сам по себе не содержит информации о периодическом движении и, следовательно, никакой информации о винном бокале. Информация содержится во *взаимосвязи* между многими отсчетами в сигнале.

Это показывает нам важность переходной и частотной характеристик. *Переходная характеристика* описывает то, как информация, представленная во *временной области*, изменяется системой. Напротив, *частотная характеристика* показывает, как изменяется информация, представленная в *частотной области*. Это различие является абсолютно критичным при конструировании фильтров, поскольку невозможно оптимизировать фильтр одновременно для обоих приложений. Хорошая работа во временной области приводит к плохой работе в частотной области и наоборот. Если Вы проектируете фильтр для удаления шума из сигнала ЭКГ (информация представлена во временной области), переходная характеристика является важным параметром, а частотная характеристика не представляет интереса. Если Вашей задачей является разработка цифрового фильтра для слухового аппарата (с информацией представленной в частотной области), частотная характеристика очень важна, тогда как переходная характеристика не имеет значения. А теперь давайте посмотрим, что делает фильтр оптимальным для приложений временной области или частотной области.

Параметры временной области

Может быть неочевидно, почему переходная характеристика является такой важной в фильтрах временной области. Вас может удивлять, почему импульсная характеристика не является важным параметром. Ответ заключается в способе, которым человеческий мозг понимает и обрабатывает информацию. Запомните, что переходная, импульсная и частотная характеристики все содержат идентичную информацию только в различной форме. Переходная характеристика полезна в анализе временной области, потому что она соответствует способу, которым люди рассматривают информацию, содержащуюся в сигналах.

Например, предположим, что Вам дан сигнал некоторого неизвестного источника и Вас просят его проанализировать. Первая вещь, которую Вы сделаете – это разделите сигнал на участки с похожими характеристиками. Вы не сможете остановиться, чтобы не сделать этого, Ваш мозг сделает это автоматически. Некоторые области могут быть гладкими, другие могут иметь пики большой амплитуды, еще одни могут быть зашумлены. Такая сегментация выполняется при помощи идентификации точек, которые отделяют участки. Вот где в дело вступает ступенчатая функция. Ступенчатая функция наиболее чистый способ представления деления между двумя разнородными областями. Она может отмечать то, когда начинается событие или то, когда событие заканчивается. Она говорит Вам, что-то, что находится *слева* некоторым образом отличается от того, что

находится *справа*. Вот каким образом человеческий мозг рассматривает информацию временной области: группа ступенчатых функций, которая делит информацию на сегменты с похожими характеристиками. Переходная характеристика, в свою очередь, является важной, потому что она описывает, как разделительные линии изменяются фильтром.

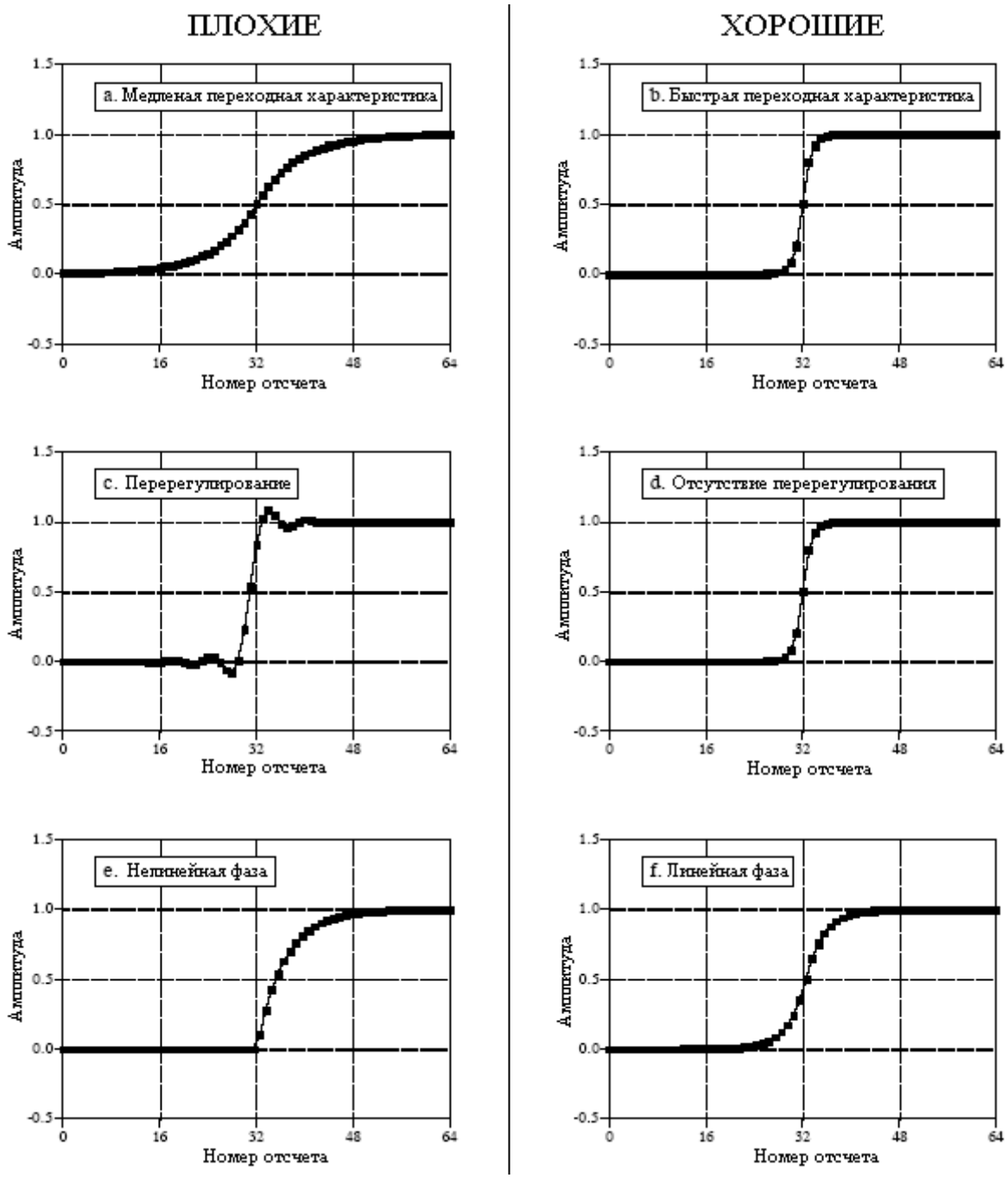


Рис. 14.2 Параметры для оценки характеристик во временной области

Параметры переходной характеристики являющиеся, важными при проектировании, показаны на рис. 14.2. Для того чтобы различать события в сигнале, длительность переходной характеристики должна быть короче, чем интервал между событиями. Это требует того, что переходная характеристика должна быть настолько *быстрой* (жаргон ЦОС), насколько это возможно. Это показано на рис. 14.2a и рис. 14.2b. Самый простой способ определения **времени нарастания** (опять жаргон) состоит в том, чтобы оценить число отсчетов между 10 % и 90 % уровнями амплитуд. Почему очень

быстрое время нарастания не всегда возможно? На это есть множество причин: уменьшение шума, внутренние ограничения системы сбора данных, предотвращение совмещения имен и т.д.

Рис. 14.2c и рис. 14.2d показывают следующий важный параметр: **перерегулирование** в переходной характеристике. Вообще перерегулирование должно быть устранено, потому что оно изменяет амплитуду отсчетов в сигнале; это основное искажение информации содержащейся во временной области. Все это можно сконцентрировать в одном вопросе: Происходит ли перерегулирование, которое Вы наблюдаете в сигнале, от вещей, которые Вы пытаетесь измерить, или от фильтра, который Вы применили?

Наконец, часто очень желательно, чтобы верхняя половина переходной характеристики была симметрична с нижней половиной, что иллюстрируется рис. 14.2e и рис. 14.2f. Эта симметрия необходима, для того чтобы сделать *нарастающий фронт* выглядящим точно так же, как и *падающий фронт*. Эта симметрия называется **линейной фазой**, потому что частотная характеристика имеет фазу, которая является прямой линией (обсуждается в Главе 19). Убедитесь, что Вы понимаете эти три параметра; они являются ключом к оценке фильтров временной области.

Параметры частотной области

На рис. 14.3 показаны четыре основных частотных характеристики. Цель этих фильтров позволить некоторым частотам проходить без изменений при полном блокировании других частот. **Полоса пропускания** относится к тем частотам, которые пропускаются, тогда как **полоса заграждения** содержит те частоты, которые блокированы. **Полоса перехода** находится между ними. **Быстрый завал** означает, что полоса перехода очень узкая. Раздел между полосой пропускания и полосой перехода называется **частотой среза**. При проектировании аналогового фильтра, частота среза обычно определяется там, где амплитуда снижается до 0,707 (т.е. -3dB) (частота среза представляет собой частоту, на которой квадрат модуля передаточной функции равен 0,5 – прим. перев.). Цифровые фильтры менее стандартизированы, увидеть здесь уровни амплитуд в 99 %, 90 %, 70,7 % и 50%, определенные как частота среза, обычное явление.

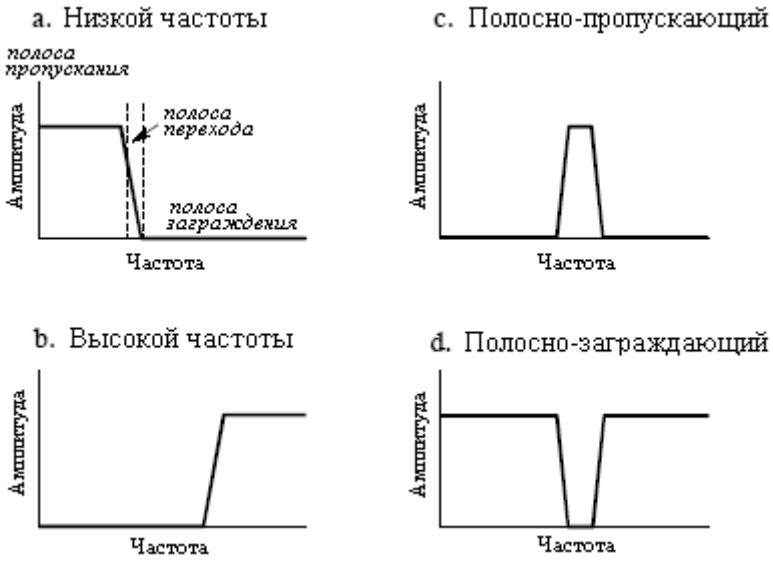


Рис. 14.3 Четыре типичных частотных характеристики

На рис. 14.4 показаны три параметра, которые измеряют то, насколько хорошо работает фильтр в частотной области. Для разделения близко расположенных частот

фильтр должен иметь **быстрый завал**, что иллюстрируется рис. 14.4a и рис. 14.4b. Для того чтобы частоты полосы пропускания проходили через фильтр без изменений, не должно быть **пульсаций полосы пропускания**, как показано на рис. 14.4c и рис. 14.4d. Наконец, чтобы адекватно блокировать частоты полосы заграждения, необходимо иметь хорошее **ослабление полосы заграждения**, показанное на рис. 14.4e и рис. 14.4f.

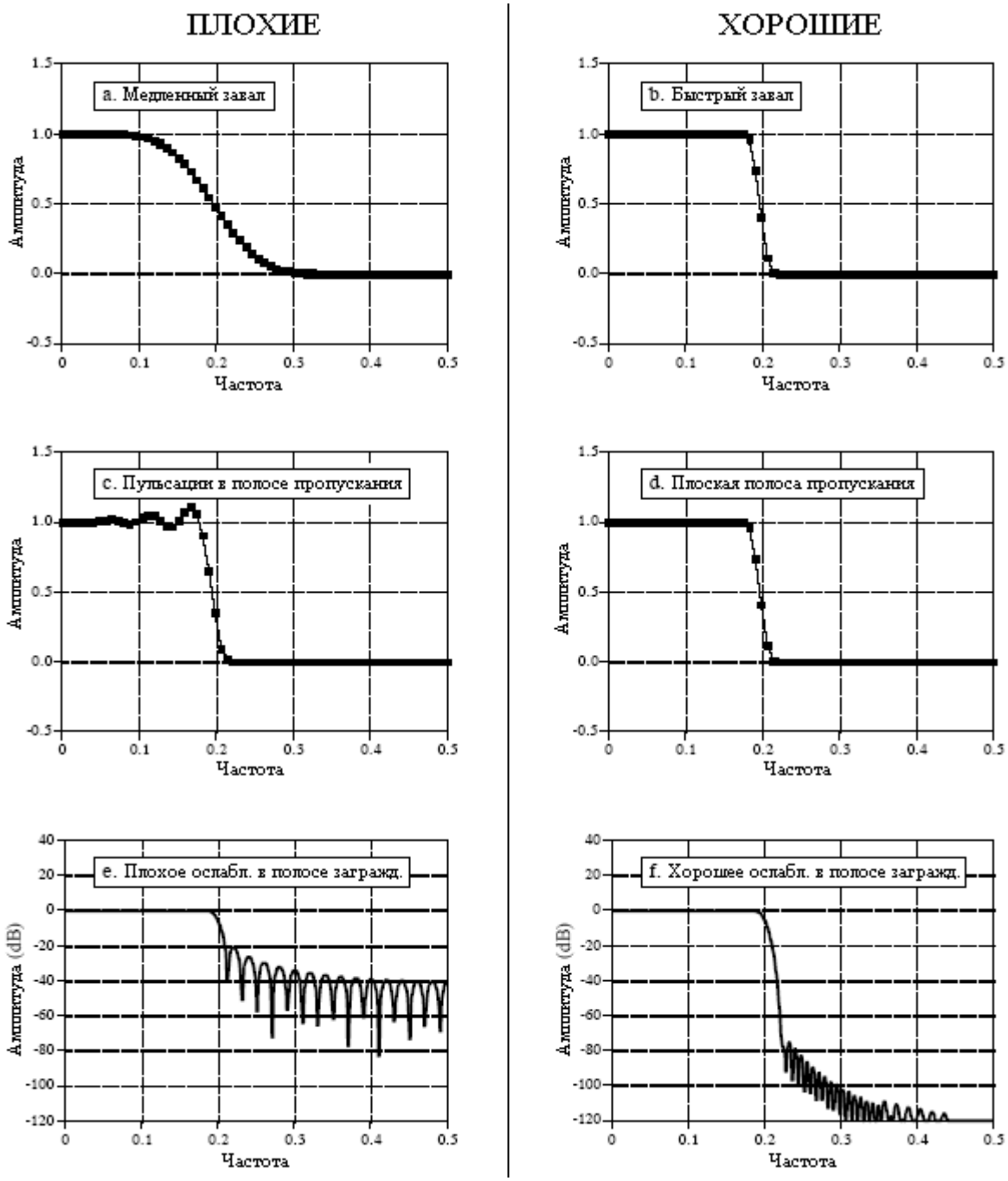


Рис. 14.4 Параметры для оценки характеристик в частотной области

Почему в этих параметрах нет ничего о *фазе*? Во-первых, в большинстве приложений частотной области фаза не важна. Например, фаза аудио сигнала является почти полностью случайной, и содержит немного полезной информации. Во-вторых, если фаза важна, очень легко сделать цифровой фильтр с *идеальной* фазовой характеристикой, т.е. все частоты проходят через фильтр с нулевым фазовым сдвигом (также обсуждается в Главе 19). Для сравнения, аналоговые фильтры в этом отношении ужасны.

Предыдущие главы описывали, как ДПФ преобразует импульсную характеристику системы в ее частотную характеристику. Вот краткий обзор этого. Наиболее быстрый способ вычисления ДПФ – это вычисление посредством алгоритма БПФ, представленного в Главе 12. Начиная с ядра фильтра длиной *N* отсчетов, БПФ вычисляет частотный спектр, состоящий из *N* точечной *действительной части* и *N* точечной *мнимой части*. Полезную информацию содержат только отсчеты БПФ с *0* по *N/2*; оставшиеся точки являются дубликатами (отрицательные частоты) и их можно игнорировать. Так как действительная и мнимая части трудны для человеческого понимания, они обычно преобразуются в полярные обозначения так, как описано в Главе 8. Это дает сигналы модуля и фазы, каждый из которых изменяется от отсчета *0* до отсчета *N/2* (т.е. *N/2+1* отсчет в каждом сигнале). Например, импульсная характеристика из *256* точек даст в результате частотную характеристику, изменяющуюся от точки *0* до точки *128*. Отсчет *0* представляет постоянную составляющую, т.е. нулевую частоту. Отсчет *128* представляет половину частоты дискретизации. Помните, никакие частоты выше чем половина частоты дискретизации появиться в дискретных данных не могут.

Число отсчетов, используемых для представления импульсной характеристики, может быть произвольно большим. Например, предположим Вы хотите найти частотную характеристику ядра фильтра, которая состоит из *80* точек. Поскольку БПФ работает только с сигналами, число отсчетов которых кратно двум в соответствующей степени, Вам необходимо добавить к сигналу *48* нулей, чтобы довести его длину до *128* отсчетов. Это *дополнение нулями* не изменяет импульсной характеристики. Чтобы понять, почему это так, подумайте о том, что случится с этими добавленными нулями, когда осуществляется свертка входного сигнала с импульсной характеристикой системы. Добавленные нули просто *исчезают* в свертке и не влияют на результат.

Принимая это во внимание, в качестве следующего шага Вы могли бы добавить *много-много* нулей к импульсной характеристике, делая ее, скажем 256, 512, или 1024 точек длиной. Важной идеей является то, что более длинная импульсная характеристика дает в результате более близко расположенные точки данных в частотной характеристике. То есть, между постоянной составляющей и половиной частоты дискретизации располагается большее число отсчетов. Доводя это до крайности, если импульсная характеристика дополнена *бесконечным* числом нулей, то точки данных в частотной характеристике располагаются бесконечно близко друг к другу, т.е. становятся непрерывной линией. Другими словами, частотная характеристика фильтра является действительно *непрерывным* сигналом между постоянной составляющей и половиной частоты дискретизации. Результат ДПФ является *отсчетами* этой непрерывной линии. Какой длины импульсную характеристику Вам следует использовать при вычислении частотной характеристики фильтра? Прежде всего, попробуйте *N=1024* и не бойтесь изменить его, если нужно (например, при недостаточном разрешении или чрезмерном времени вычисления).

Не забывайте, что “хорошие” и “плохие” параметры, обсуждаемые в этой главе, являются только обобщениями. Многие сигналы не укладываются аккуратно в категории. Например, рассмотрим сигнал ЭКГ, искаженный помехами в 60 Гц. Информация закодирована во *временной области*, а помеха наилучшим образом связана с *частотной областью*. Лучший проект для этого приложения связан с наличием обмена между областями и может идти против традиционной мудрости этой главы. Запомните правило номер один для образования: *Параграф в книге не дает Вам лицензию на то, чтобы перестать думать*.

Высокочастотные, полосно-пропускающие и полосно-заграждающие фильтры

Проектирование высокочастотных, полосно-пропускающих и полосно-заграждающих фильтров начинается с низкочастотного фильтра и его преобразования к

желаемой характеристике. По этой причине в большинстве обсуждений по проектированию фильтров приводятся примеры проектирования только фильтров нижних частот. Существуют два способа преобразования фильтров нижних частот в фильтры верхних частот: **инверсия спектра** и **обращение спектра**. Оба способа одинаково полезны.

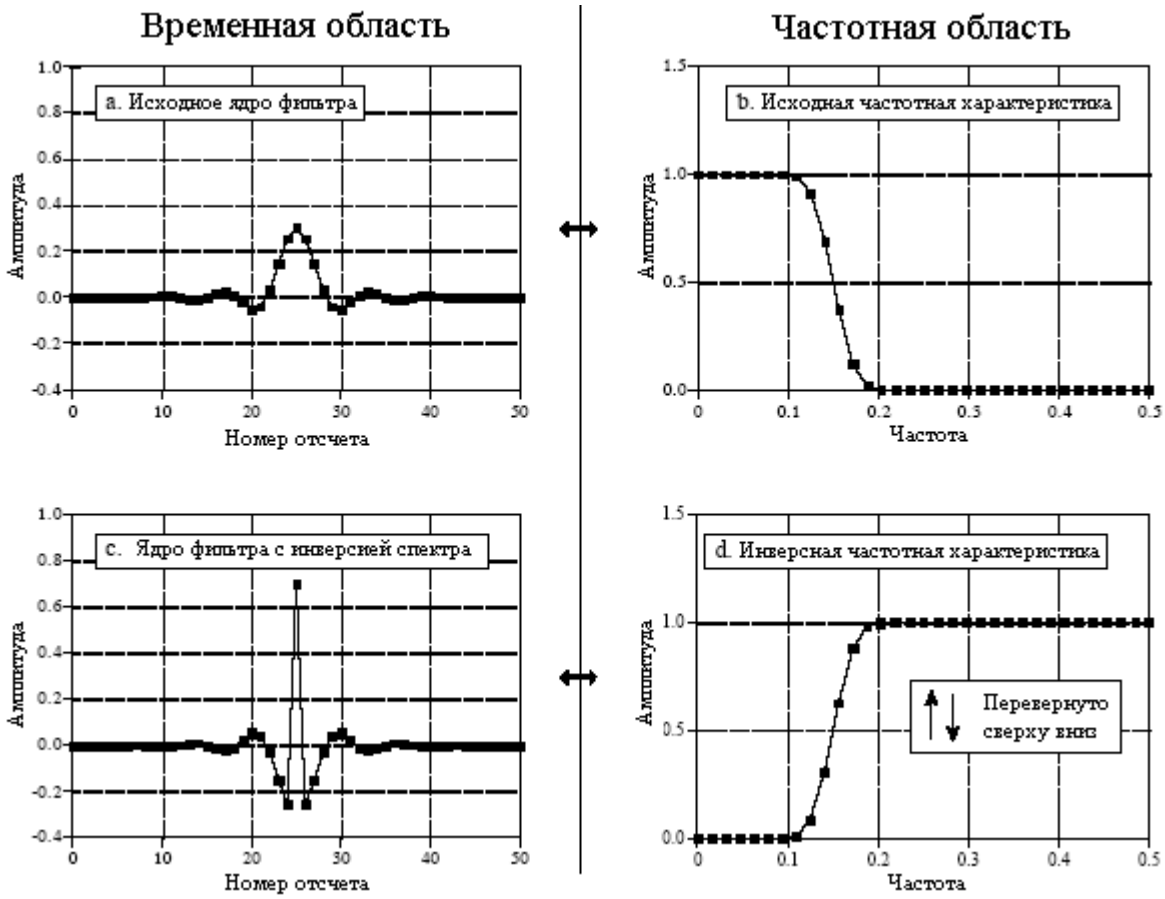


Рис. 14.5 Пример инверсии спектра

Пример *инверсии спектра* показан на рис. 14.5. На рис. 14.5a показано ядро фильтра низкой частоты, называемое синк функцией ограниченной окном (тема Главы 16). Это ядро фильтра в *51* точку длиной, хотя многие из отсчетов имеют настолько маленькое значение, что они кажутся на этом графике нулевыми. Соответствующая частотная характеристика, найденная при помощи добавления *13* нулей к ядру фильтра и взятия *64* точечного БПФ, показана на рис. 14.5b. Для преобразования ядра фильтра низкой частоты в ядро фильтра высокой частоты должны быть сделаны две вещи. Во-первых, изменен знак каждого отсчета в ядре фильтра. Во-вторых, прибавлена *единица* к отсчету в центре симметрии. Это даст ядро фильтра высокой частоты, показанное на рис. 14.5c, с частотной характеристикой, показанной на рис. 14.5d. Инверсия спектра *переворачивает* частотную характеристику *сверху вниз*, преобразуя полосу пропускания в полосу заграждения и полосу заграждения в полосу пропускания. Другими словами, она преобразует фильтр из низкочастотного в высокочастотный, из высокочастотного в низкочастотный, из полосно-пропускающего в полосно-заграждающий или из полосно-заграждающего в полосно-пропускающий.

Рисунок 14.6 показывает, почему эта двух шаговая модификация временной области дает в результате перевернутый частотный спектр. На рис. 14.6a входной сигнал $x[n]$ подается на две параллельно соединенные системы. Одна из этих систем является низкочастотным фильтром с импульсной характеристикой, заданной как $h[n]$. Другая

система *ничего* не делает с сигналом и, следовательно имеет импульсную характеристику, представляющую собой дельта функцию $\delta[n]$. Общий выход $y[n]$ эквивалентен выходу все пропускающей системы *минус* выход низкочастотной системы. Поскольку низкочастотные компоненты вычитаются из исходного сигнала, на выходе появляются только высокочастотные компоненты. Таким образом, сформирован высокочастотный фильтр.

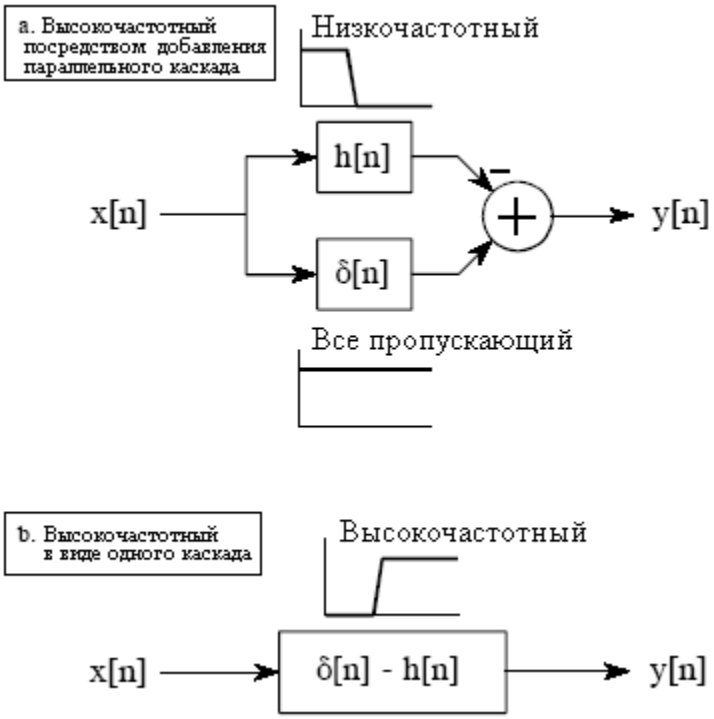


Рис. 14.6 Блок-схема инверсии спектра

В компьютерной программе это может быть выполнено в виде двух шаговой операции: пропустить сигнал через низкочастотный фильтр, а затем вычесть отфильтрованный сигнал из исходного. Однако вся операция за счет объединения двух ядер фильтров может быть выполнена и за один шаг. Как описано в Главе 7, параллельные системы с суммированием выходов могут быть объединены в один каскад посредством сложения их импульсных характеристик. Как показано на рис. 14.6b, для высокочастотного фильтра ядро фильтра задается как: $\delta[n]$-$h[n]$. То есть изменить знак всех отсчетов, а затем добавить единицу к отсчету в центре симметрии.

Для того чтобы эта техника работала, низкочастотные компоненты, выходящие из низкочастотного фильтра должны иметь точно такую же фазу, как и низкочастотные компоненты, выходящие из системы пропускающей всё. В противном случае полное вычитание не может иметь место. Это накладывает на метод два ограничения: (1) исходное ядро фильтра должно иметь симметрию слева направо (т.е. нулевую или линейную фазу), и (2) к центру симметрии должен быть добавлен дельта импульс.

Второй способ преобразования низкочастотного фильтра в высокочастотный, *обращение спектра*, иллюстрируется на рис. 14.7. Так же, как и прежде, ядро низкочастотного фильтра на рис. 14.7a соответствует частотной характеристике на рис. 14.7b. Ядро высокочастотного фильтра на рис. 14.7c формируется при помощи *изменения знака каждого последующего отсчета* на рис. 14.7a. Как показано на рис. 14.7d, это переворачивает частотную характеристику *слева направо*: *0* становится *0,5* и *0,5* становится *0*. Частота среза низкочастотного фильтра в данном примере *0,15* дает в результате частоту среза высокочастотного фильтра *0,35*.

Изменение знака каждого последующего образца эквивалентно умножению ядра фильтра на синусоиду с частотой *0,5*. Как обсуждалось в Главе 10, это дает эффект *сдвига* частотной области на *0,5*. Взгляните на рис. 14.7b и вообразите отрицательные частоты между *-0,5* и *0*, являющиеся зеркальным отображением частот между *0* и *0,5*. Частоты, которые появились на рис. 14.7d, представляют собой отрицательные частоты с рис. 14.7b, сдвинутые на *0,5*.

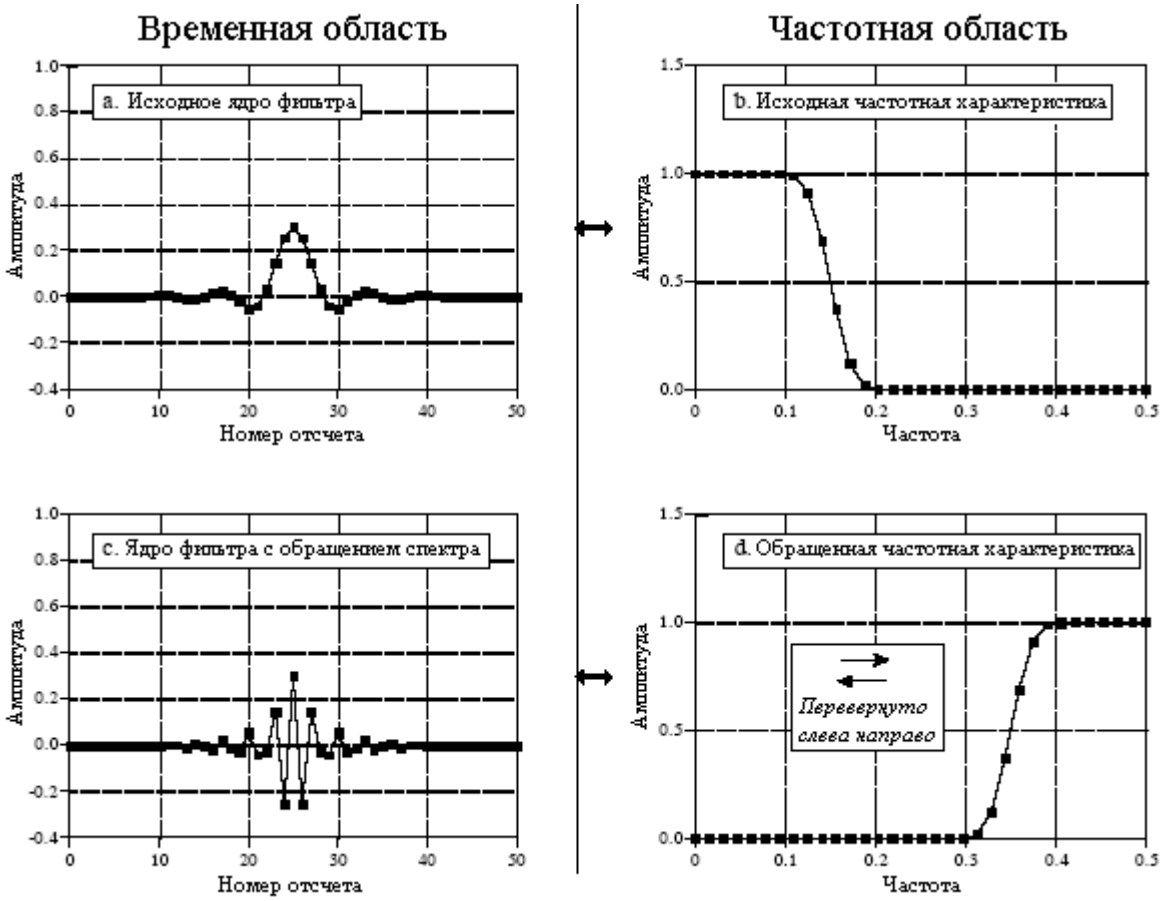


Рис. 14.7 Пример обращения спектра

Наконец, на рис. 14.8 и рис. 14.9 показано как низкочастотное и высокочастотное ядра фильтров могут быть объединены для формирования полосно-пропускающего и полосно-заграждающего фильтров. Вкратце, *сложение* ядер фильтров дает *полосно-заграждающий* фильтр, в то время как *свертка* ядер фильтров дает *полосно-пропускающий* фильтр. Как обсуждалось в Главе 7, в основе этого лежит параллельное и последовательное соединения объединяемых систем. Может также использоваться многократная комбинация этих методов. Например, полосно-пропускающий фильтр может быть спроектирован суммированием двух ядер фильтров формирующих полосно-заграждающий фильтр, после чего может быть использована *инверсия спектра* или *обращение спектра*, как это было первоначально описано. Все эти методы с небольшими сюрпризами работают очень хорошо.

Классификация фильтров

Таблица 14.1 подытоживает то, как классифицируются цифровые фильтры по их *применению* и *исполнению*. Применение цифровых фильтров может быть разбито на три категории: *временная область*, *частотная область* и *заказные*. Как было описано ранее, фильтры временной области применяются тогда, когда информация закодирована в форме

кривой сигнала. Фильтрация временной области используется для выполнения таких действий, как сглаживание, удаление постоянной составляющей, формирование формы волны и т.д. Напротив, фильтры частотной области используются тогда, когда информация содержится в амплитуде, частоте и фазе составляющих сигнал синусоид. Целью этих фильтров является отделение одной полосы частот от другой. Заказные фильтры используются тогда, когда от фильтра требуется выполнение специальных действий, что-то более сложное, чем четыре основных характеристики (низкочастотный, высокочастотный, полосно-пропускающий и полосно-заграждающий). Например, Глава 17 описывает, как заказные фильтры могут использоваться для *устранения свертки*, способ противодействия возникновению нежелательной свертки.

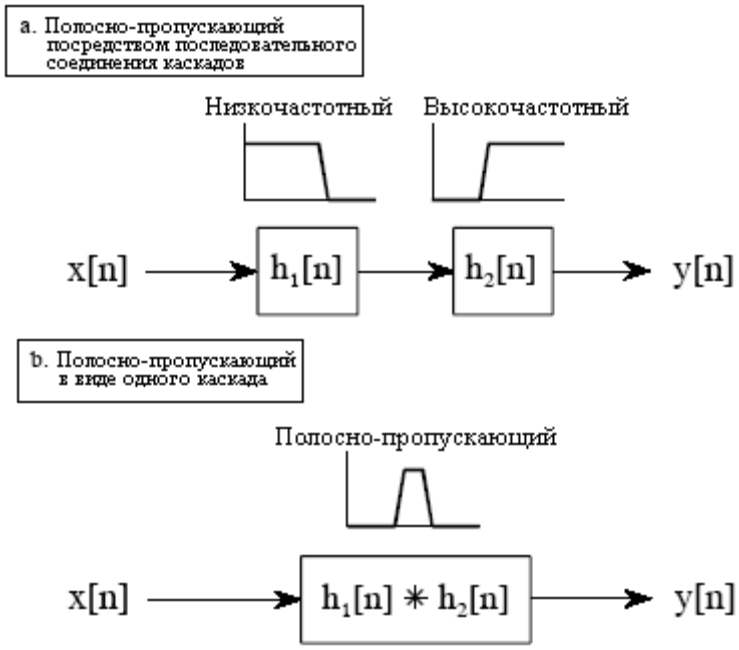


Рис. 14.8 Проектирование полосно-пропускающего фильтра

Цифровые фильтры могут быть реализованы двумя способами, при помощи *свертки* (называемые также фильтрами с *конечной импульсной характеристикой* или *КИХ*-фильтрами) и при помощи *рекурсии* (называемые также фильтрами с *бесконечной импульсной характеристикой* или *БИХ*-фильтрами). Фильтры, реализованные при помощи свертки, могут работать гораздо лучше, чем фильтры, использующие рекурсию, но работают значительно медленнее.

В следующих шести главах цифровые фильтры описываются в соответствии с классификацией приведенной в таблице 14.1. Сначала мы посмотрим на фильтры, реализованные при помощи свертки. Скользящее среднее (Глава 15) используется во временной области, *синк функция ограниченная окном* (Глава 16) используется в частотной области, а *заказные КИХ*-фильтры (Глава 17) используются, когда необходимо что-то особенное. Чтобы завершить обсуждение КИХ-фильтров в Главе 18 представляется метод, называемый БПФ сверткой. Это алгоритм для увеличения скорости выполнения свертки, позволяющий КИХ-фильтрам работать быстрее.

Далее мы рассмотрим рекурсивные фильтры. Рекурсивные фильтры с *одним полюсом* (Глава 19) используются во временной области, тогда как фильтры *Чебышева* (Глава 20) используются в частотной области. Рекурсивные фильтры, имеющие заказные характеристики, проектируются при помощи *итеративных методов*. По этой причине мы отложим их обсуждение до Главы 26, в которой они будут представлены совместно с другими типами итеративных процедур: нейронными сетями.

Как показано в таблице 14.1, *свертка* и *рекурсия* являются альтернативными методами и для конкретных приложений Вы должны использовать тот или другой. А как

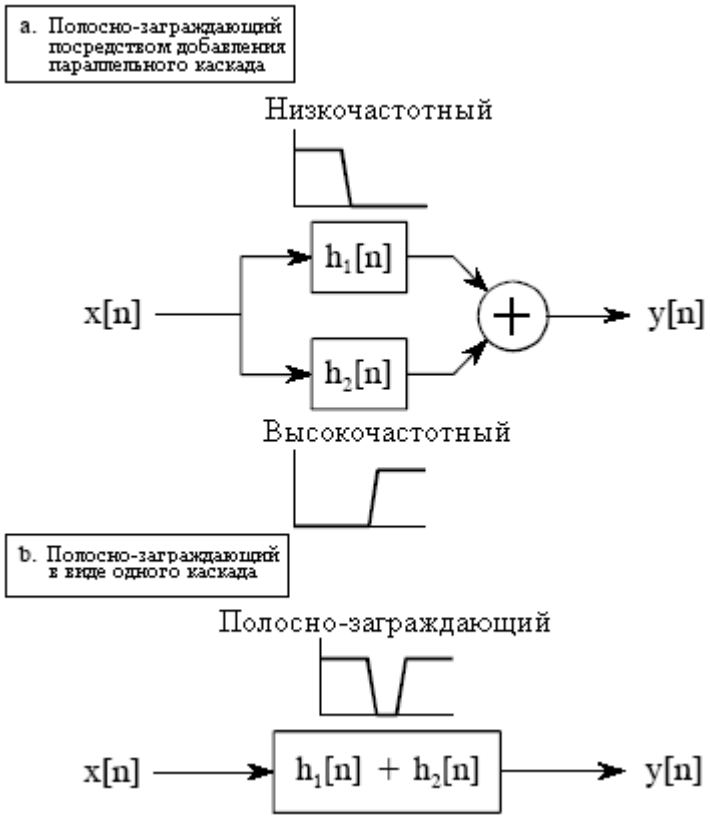


Рис. 14.9 Проектирование полосно-заграждающего фильтра

же Вам сделать выбор? Глава 21 представляет точное сравнение двух методов, как во временной, так и в частотной областях.

Таблица 14.1

ФИЛЬТР ИСПОЛЬЗУЕТСЯ ДЛЯ:	ФИЛЬТР РЕАЛИЗОВАН ПРИ ПОМОЩИ: Свертка *конечная импульсная характеристика (КИХ)*	Рекурсия *бесконечная импульсная характеристика (БИХ)*
Временная область *(сглаживание, удаление постоянной составляющей)*	Скользящее среднее (Гл. 15)	Один полюс (Гл. 19)
Частотная область *(отделение частот)*	Синк функция ограниченная окном (Гл. 16)	Чебышева (Гл. 20)
Заказные *(устранение свертки)*	Заказной КИХ (Гл. 17)	Итеративное проектирование (Гл. 26)

ГЛАВА
15

ФИЛЬТРЫ СКОЛЬЗЯЩЕГО СРЕДНЕГО

Фильтр скользящего среднего - наиболее часто встречающийся в ЦОС, главным образом потому, что это - самый простой цифровой фильтр для понимания и использования. Несмотря на его простоту, фильтр скользящего среднего является *оптимальным* для общей задачи снижения случайного шума при сохранении крутой переходной характеристики. Это делает его первым фильтром для сигналов, закодированных во временной области. Однако скользящее среднее *наихудший* фильтр, с небольшой способностью отделять одну полосу частот от другой, для сигналов, закодированных в частотной области. Сородичами скользящего среднего являются Гауссов фильтр, фильтр Блэкмена и скользящего среднего с многократным проходом. Они слегка лучше работают в частотной области за счет увеличенного времени вычисления.

Реализация посредством свертки

Как подразумевает название, фильтр скользящего среднего работает посредством усреднения нескольких точек из входного сигнала для получения каждой точки выходного сигнала. В форме уравнения это записывается как:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j], \qquad (15.1)$$

где $x[\ ]$ – входной сигнал, $y[\ ]$ – выходной сигнал, а M – число точек в среднем. В *5* точечном фильтре скользящего среднего точка *80* в выходном сигнале будет определяться как:

$$y[80] = \frac{y[80] + y[81] + y[82] + y[83] + y[84]}{5}.$$

В качестве альтернативы, группа точек из входного сигнала может быть выбрана *симметрично* относительно выходной точки:

$$y[80] = \frac{y[78] + y[79] + y[80] + y[81] + y[82]}{5}.$$

Это соответствует замене суммы от *j=0* до *j=M-1* в уравнении (15.1) на сумму от *j=-(M-1)/2* до *j=(M-1)/2*. Например, в *11* точечном фильтре скользящего среднего индекс *j* может изменяться от *0* до *10* (одностороннее усреднение) или от *–5* до *5* (симметричное усреднение). Симметричное усреднение требует, чтобы *M* было *нечетным* числом. Когда точки располагаются только с одной стороны программировать немного легче, однако это приводит к относительному сдвигу между входным и выходным сигналами.

Вы должны признать, что фильтр скользящего среднего является *сверткой*, использующей очень простое ядро фильтра. Например, *5* точечный фильтр имеет ядро фильтра: …, *0*, *0*, *1/5*, *1/5*, *1/5*, *1/5*, *1/5*, *0*, *0*, …. То есть, фильтр скользящего среднего представляет собой свертку входного сигнала с *прямоугольным импульсом*, имеющим площадь равную единице. В таблице 15.1 показана программа для реализации фильтра скользящего среднего.

Таблица 15.1

```
100 'MOVING AVERAGE FILTER
110 'This program filters 5000 samples with a 101 point moving
120 'average filter, resulting in 4900 samples of filtered data.
130 '
140 DIM X[4999]                    'X[ ] holds the input signal
150 DIM Y[4999]                    'Y[ ] holds the output signal
160 '
170 GOSUB XXXX                     'Mythical subroutine to load X[ ]
180 '
190 FOR I% = 50 TO 4949            'Loop for each point in the output signal
200 Y[I%] = 0                      'Zero, so it can be used as an accumulator
210 FOR J% = -50 TO 50             'Calculate the summation
220 Y[I%] = Y[I%] + X(I%+J%]
230 NEXT J%
240 Y[I%] = Y[I%]/101              'Complete the average by dividing
250 NEXT I%
260 '
270 END
```

Снижение шума против переходной характеристики

Многие ученые и инженеры испытывают чувство вины относительно использования фильтра скользящего среднего. Часто, поскольку это очень просто, фильтр скользящего среднего - это первая вещь, которую пытаются применить тогда, когда сталкиваются с проблемой. Даже если проблема полностью решена, все же остается чувство, что следовало бы сделать что-то еще. Эта ситуация действительно ироническая. Для многих приложений фильтр скользящего среднего не только очень хорош, он *оптимален* для решения общей проблемы, снижения уровня случайного белого шума при сохранении самой крутой переходной характеристики.

На рис. 15.1 показано, как это все работает. Сигнал на рис. 15.1a представляет собой импульс, погруженный в случайный шум. На рис. 15.1b и рис. 15.1c сглаживающее действие фильтра скользящего среднего уменьшает амплитуду случайного шума (хорошо), но также уменьшает крутизну фронтов (плохо). Из всех возможных линейных фильтров, которые можно было бы использовать, скользящее среднее для заданной крутизны фронтов дает самый низкий шум. Величина снижения шума равна корню квадратному из числа точек в среднем. Например, *100* точечный фильтр скользящего среднего снижает шум в *10* раз.

Для того чтобы понять, почему скользящее среднее является наилучшим решением, представьте, что мы хотим спроектировать фильтр с фиксированной крутизной фронтов. Предположим, что мы фиксируем крутизну фронта, определяя, что в нарастании переходной характеристики находится *11* точек. Это требует того, чтобы в ядре фильтра находилось *11* точек. Вопросом оптимизации является: как нам выбрать одиннадцать

значений ядра фильтра, чтобы минимизировать шум в выходном сигнале? Поскольку шум, который мы пытаемся уменьшить, является случайным, ни одна из входных точек не является особенной; каждая столь же зашумлена, как и соседняя. Следовательно, бесполезно отдавать предпочтение какой-либо из входных точек, приписывая ей больший коэффициент в ядре фильтра. Самый низкий уровень шума получается тогда, когда все входные отсчеты обрабатываются одинаково, т.е. фильтр скользящего среднего. (Позже в этой главе мы покажем, что другие фильтры по существу *также* хороши. Дело состоит в том, что нет фильтра *лучше*, чем простое скользящее среднее.)

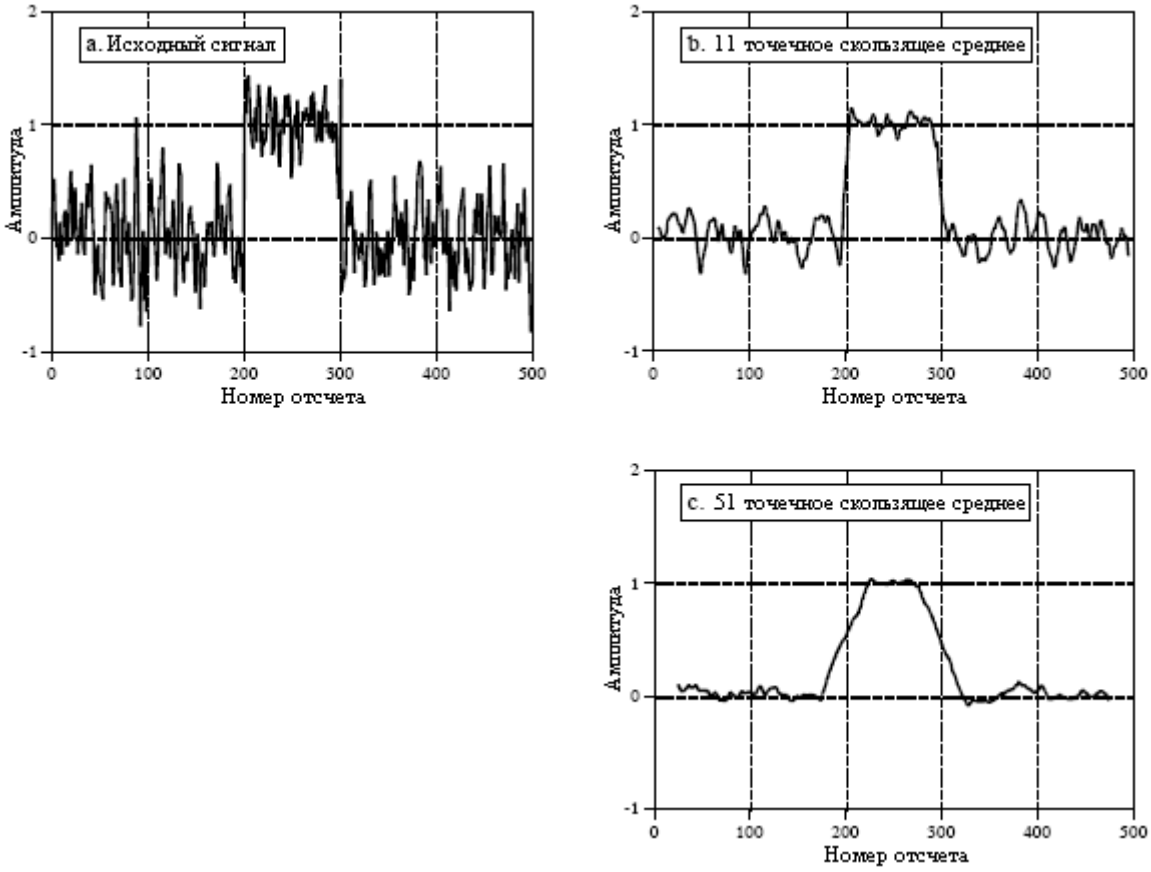


Рис. 15.1 Пример фильтра скользящего среднего

Частотная характеристика

На рис. 15.2 показана частотная характеристика фильтра скользящего среднего. Математически она описывается с помощью преобразования Фурье прямоугольного импульса, что обсуждалось в Главе 11:

$$H[f] = \frac{\sin(\pi f M)}{M \sin(\pi f)}. \qquad (15.2)$$

Завал очень медленный, а ослабление в полосе заграждения просто ужасно. Ясно, что фильтр скользящего среднего не может отделить одну полосу частот от другой. Запомните, хорошая работа во временной области приводит к плохой работе в частотной области, и наоборот. Короче говоря, скользящее среднее является исключительно

хорошим *сглаживающим фильтром* (работа во временной области), но исключительно плохим *фильтром нижних частот* (работа в частотной области).

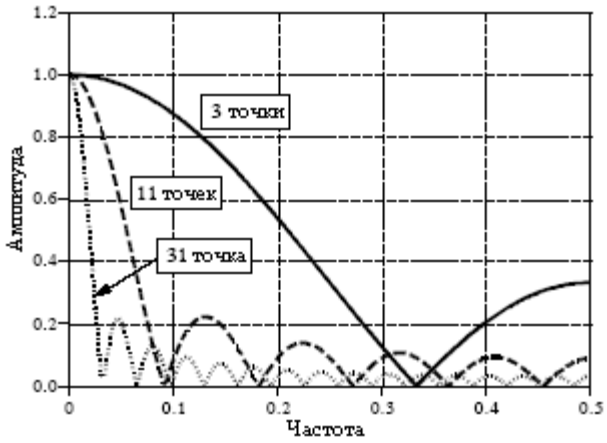


Рис. 15.2 Частотная характеристика фильтра скользящего среднего

Сородичи фильтра скользящего среднего

В идеальном мире разработчикам фильтров пришлось бы иметь дело с информацией закодированной *либо* только во временной области, *либо* только в частотной, но никогда со смесью обоих в одном и том же сигнале. К сожалению, существует ряд приложений, где обе области одновременно важны. Например, в эту противную категорию попадает телевизионный сигнал. Видео информация кодируется во временной области, т.е. конфигурация формы волны соответствует образцам яркости в изображении. Однако во время передачи видео сигнал обрабатывается в соответствии с его частотным составом, таким как общая полоса пропускания, способ добавления несущих звука и цвета, удаление и восстановление постоянной составляющей и т.д. В качестве другого примера, даже если информация в сигнале закодирована во временной области, электромагнитные помехи наилучшим образом понятны в частотной области. Например, на наблюдения температуры в научном эксперименте могла быть наложена помеха 60 Гц от промышленной сети, 30 кГц от преобразователей электропитания или 1320 кГц от местной АМ радиостанции. В таких смешанных приложениях и могут быть полезны сородичи фильтра скользящего среднего, работающие в частотной области лучше.

Фильтры скользящего среднего с многократным проходом заключаются в пропускании входного сигнала через фильтр скользящего среднего два или более раза. Рисунок 15.3a показывает общее ядро фильтра, получающееся после одного, двух и четырех проходов. Два прохода эквивалентны использованию *треугольного* ядра фильтра (свертка прямоугольного ядра фильтра с самим собой). После четырех или более проходов эквивалентное ядро фильтра выглядит как *Гауссиан* (вспомните центральную граничную теорему). Как показано на рис. 15.3b, многократные проходы, по сравнению с прямой линией после одного прохода, придают переходной характеристике “s”-образную форму. Частотные характеристики, показанные на рис. 15.3c и рис. 15.3d, определяемые в соответствии с уравнением (15.2), на каждом проходе *умножаются* сами на себя. То есть, каждая свертка во временной области приводит к умножению в частотной области.

На рис. 15.4 показаны частотные характеристики двух других сородичей фильтра скользящего среднего. Когда в качестве ядра фильтра используется чистый **Гауссиан**, как обсуждалось в Главе 11, частотная характеристика также представляет собой Гауссиан.

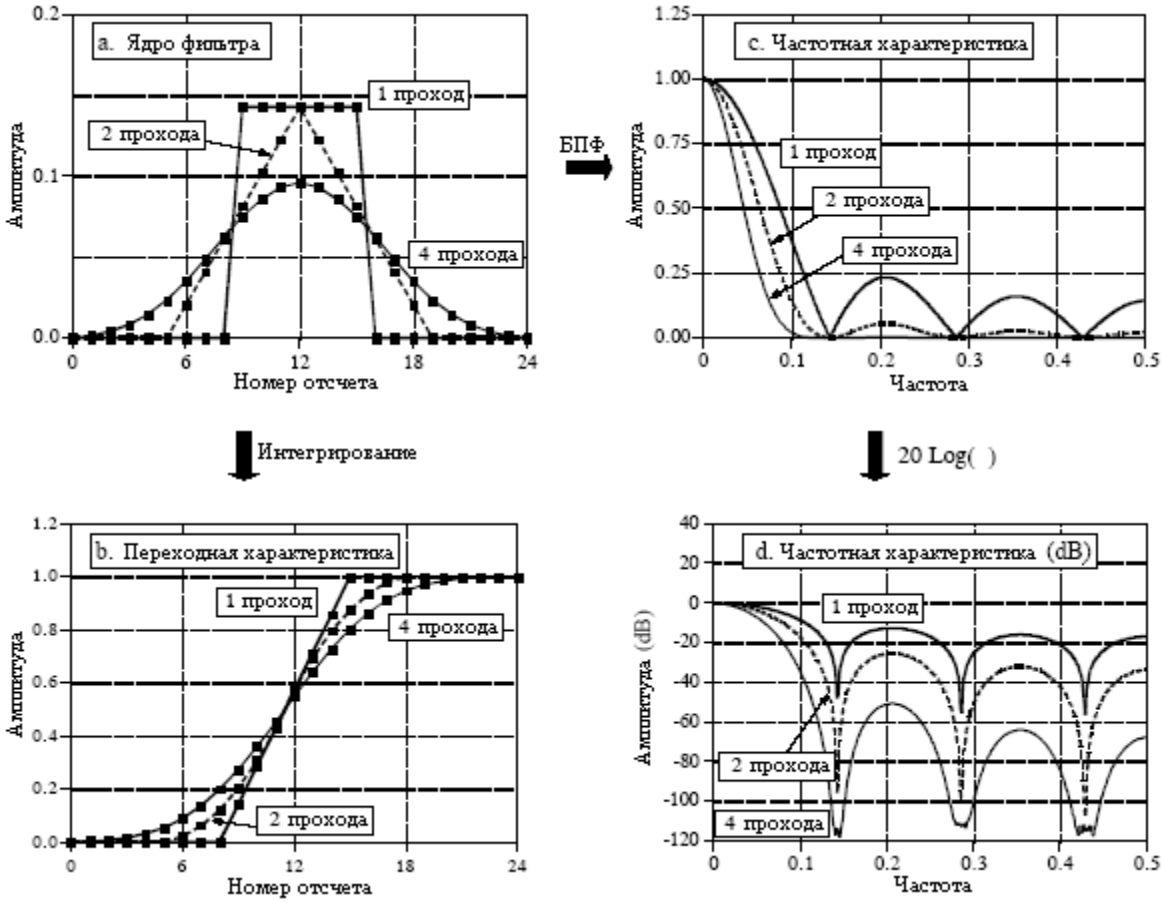


Рис. 15.3 Характеристики фильтров скользящего среднего с многократным проходом

Гауссиан является важным, поскольку он представляет собой импульсную характеристику многих естественных и искусственных систем. Например, короткий импульс света, входящий в длинную волоконно-оптическую линию передачи, из-за различных путей, пройденных фотонами внутри волокна, выйдет в виде Гауссова импульса. Ядро Гауссова фильтра также широко используется в *обработке изображений*, поскольку обладает уникальными свойствами, допускающими быструю двумерную свертку (см. Главу 24). Вторая частотная характеристика на рис. 15.4 соответствует использованию в качестве ядра фильтра **окна Блэкмена** (Термин *окно* здесь не несет какого-либо значения, это просто часть принятого названия данной кривой.). Точная форма окна Блэкмена дается в Главе 16 (уравнение (16.2), рисунок 16.2); однако оно выглядит очень похожим на Гауссиан.

По каким причинам эти сородичи фильтра скользящего среднего, лучше, чем сам фильтр скользящего среднего? По трем: Первая и наиболее важная, эти фильтры имеют лучшее *ослабление в полосе заграждения*, чем фильтр скользящего среднего. Вторая, ядра фильтра *сужаются* к концам до небольших амплитуд. Вспомните, что каждая точка в выходном сигнале является взвешенной суммой группы отсчетов на входе. Если ядро фильтра сужается к концам, отсчеты во входном сигнале, находящиеся далеко отсюда, дают меньший вес, чем те которые ближе. Третья, переходная характеристика представляет собой *плавную* кривую в отличие от резкой прямой линии скользящего среднего. Эти последние две причины обычно дают небольшую выгоду, хотя Вы можете найти приложения, где они обладают настоящим преимуществом.

Фильтр скользящего среднего и его сородичи все почти одинаковы, снижают случайный шум при одновременном поддержании крутой переходной характеристики. Неоднозначность заключается в том, как измеряется *время нарастания* переходной характеристики. Если время нарастания измеряется от *0%* до *100%* перехода, то, как было

показано ранее, фильтр скользящего среднего является наилучшим из того, что Вы можете сделать. Для сравнения, измерение времени нарастания от *10%* до *90%* делает окно Блэкмена *лучше*, чем фильтр скользящего среднего. Дело в том, что это просто теоретические споры; рассматривайте данные фильтры по этому параметру как одинаковые.

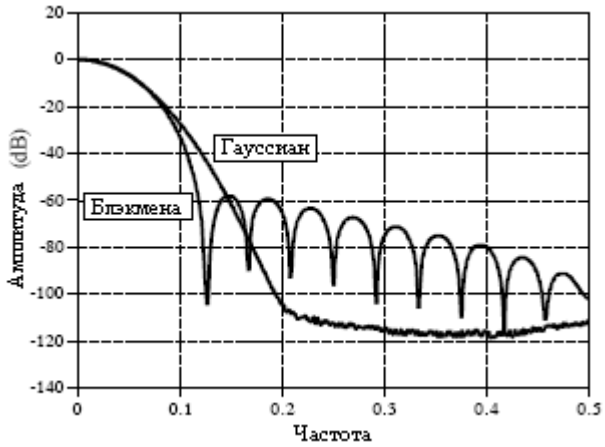


Рис. 15.4 Частотная характеристика ядер фильтров с окном Блэкмена и Гауссианом

Самым большим различием у этих фильтров является *скорость выполнения*. Используя рекурсивные алгоритмы (описывается далее), фильтр скользящего среднего будет исполняться на вашем компьютере подобно молнии. Фактически, это - *самый быстрый* доступный цифровой фильтр. Фильтр скользящего среднего с многократным проходом будет соответственно медленнее, но все еще очень быстрым. Для сравнения Гауссов фильтр и фильтр Блэкмена мучительно медленны, поскольку они должны использовать свертку. Надо полагать, в 10 умноженное на число точек в ядре фильтра раз (в основе лежит умножение, которое примерно раз в десять медленнее, чем сложение). Например, ждите, что *100* точечный Гауссов фильтр будет в *1000* раз медленнее, чем скользящее среднее, использующее рекурсию.

Рекурсивная реализация

Огромное преимущество фильтра скользящего среднего состоит в том, что он может быть реализован при помощи очень быстрого алгоритма. Для того чтобы понять этот алгоритм, представьте входной сигнал *x*[ ], проходящий через, формирующий выходной сигнал *y*[ ], семи точечный фильтр скользящего среднего. Теперь посмотрите на то, как вычисляются две смежные выходные точки *y*[*50*] и *y*[*51*]:

$$y[50]=x[47]+x[48]+x[49]+x[50]+x[51]+x[52]+x[53];$$

$$y[51]=x[48]+x[49]+x[50]+x[51]+x[52]+x[53]+x[54].$$

Эти вычисления почти одинаковы: для *y*[*50*] должны быть просуммированы точки с *x*[*48*] по *x*[*53*], для *y*[*51*] они снова должны быть просуммированы. Если точка *y*[*50*] уже была вычислена, то самый *эффективный* способ вычислить *y*[*51*] следующий:

$$y[51]=y[50]+x[54]-x[47].$$

Если, используя *y*[*50*], была найдена точка *y*[*51*], то из отсчета *y*[*51*] может быть найдена точка *y*[*52*] и т.д. После того, как в *y*[ ] вычислена первая точка, все другие точки

могут быть найдены при помощи всего одного сложения и одного вычитания на точку. Это можно выразить уравнением:

$$y[i] = y[i-1] + x[i+p] - x[i-q], \qquad (15.3)$$

где $p=(M-1)/2$ и $q=p+1$.

Заметьте, что для вычисления каждой выходной точки это уравнение использует два источника данных: точки с входа *и* ранее вычисленные точки с выхода. Это называется **рекурсивным уравнением**, означая, что результат одного вычисления используется в *будущих* вычислениях. (Термин "рекурсивный" имеет также и другие значения, особенно в компьютерных науках.) Более подробно разнообразие рекурсивных фильтров обсуждается в Главе 19. Следует знать, что рекурсивный фильтр скользящего среднего очень отличается от обычных рекурсивных фильтров. В частности, большинство рекурсивных фильтров имеют бесконечно длинную импульсную характеристику (БИХ), составленную из синусоид и экспонент. Импульсная характеристика скользящего среднего представляет собой прямоугольный импульс (конечную импульсную характеристику или КИХ).

Этот алгоритм быстрее, чем другие цифровые фильтры, по нескольким причинам. Во-первых, несмотря на длину ядра фильтра, на одну точку нужно всего два вычисления. Во-вторых, необходимы только математические операции сложения и вычитания, тогда как для большинства цифровых фильтров требуется умножение, поглощающее много времени. В- третьих, очень проста схема индексации. Каждый индекс в уравнении (15.3) находится с помощью суммирования или вычитания целых констант, которые могут быть вычислены до начала фильтрации (т.е. p и q). В-четвертых, весь алгоритм может быть выполнен с числами, представленными в форме целых. В зависимости от используемого оборудования целые числа могут быть более чем на порядок быстрее по величине, чем числа с плавающей запятой.

Удивительно, в дополнение к *быстроте*, представление в виде целого с этим алгоритмом работает *лучше*, чем представление с плавающей запятой. Ошибка округления от арифметики с плавающей запятой, если Вы не осторожны, может дать неожиданный результат. Например, представьте сигнал из *10000* отсчетов, фильтруемый с помощью этого метода. Последний отсчет в отфильтрованном сигнале содержит накопленную ошибку *10000* сложений и вычитаний. Это проявляется в выходном сигнале в виде дрейфующего смещения. С целыми такой проблемы не возникает, поскольку здесь в арифметике нет ошибки округления. Если Вы все же должны применять с этим алгоритмом плавающую запятую, программа в таблице 15.2 показывает, как использовать аккумулятор двойной точности чтобы устранить этот дрейф.

Таблица 15.2

```
100 'MOVING AVERAGE FILTER IMPLEMENTED BY RECURSION
110 'This program filters 5000 samples with a 101 point moving
120 'average filter, resulting in 4900 samples of filtered data.
130 'A double precision accumulator is used to prevent round-off drift.
140 '
150 DIM X[4999]                  'X[ ] holds the input signal
160 DIM Y[4999]                  'Y[ ] holds the output signal
170 DEFDBL ACC                   'Define the variable ACC to be double precision
180 '
190 GOSUB XXXX                   'Mythical subroutine to load X[ ]
200 '
210 ACC = 0                      'Find Y[50] by averaging points X[0] to X[100]
```

```
220 FOR I% = 0 TO 100
230 ACC = ACC + X[I%]
240 NEXT I%
250 Y[50] = ACC/101
260 '                                   'Recursive moving average filter (Eq. 15.3)
270 FOR I% = 51 TO 4949
280 ACC = ACC + X[I%+50] - X[I%-51]
290 Y[I%] = ACC/101
300 NEXT I%
310 '
320 END
```

ГЛАВА
16

ФИЛЬТРЫ С ОГРАНИЧЕННОЙ ОКНОМ СИНК ФУНКЦИЕЙ

Фильтры с ограниченной окном синк функцией используются для отделения одной полосы частот от другой. Они очень устойчивы, преподносят мало сюрпризов и могут быть доведены до невероятных уровней работы. Такие исключительные характеристики в частотной области получены за счет плохой работы во временной области, включая чрезмерные пульсации и перерегулирование переходной характеристики. При реализации с помощью стандартной свертки фильтры с ограниченной окном синк функцией легко программируются, но медленно исполняются. Глава 18 показывает, как может использоваться БПФ для того, чтобы основательно улучшить вычислительную скорость этих фильтров.

Стратегия ограничения синк функции окном

Рисунок 16-1 иллюстрирует идею, лежащую в основе фильтра с ограниченной окном синк функцией. На рис. 16.1a показана частотная характеристика *идеального* фильтра нижних частот. Все частоты ниже частоты среза f_C пропускаются с единичной амплитудой, тогда как все более высокие частоты блокируются. Полоса пропускания идеально плоская, ослабление в полосе заграждения бесконечно, а переход между ними чрезвычайно мал.

Взятие обратного преобразования Фурье от этой идеальной частотной характеристики дает идеальное ядро фильтра (импульсную характеристику), показанное на рис. 16.1b. Как обсуждалось ранее (см. Главу 11, уравнение (11.4)) - это общая форма кривой *sin(x)/x*, называемой **синк функцией** и задаваемой как:

$$h[i] = \frac{\sin(2\pi f_c i)}{i\pi}.$$

Свертка входного сигнала с этим ядром фильтра дает *идеальный* фильтр низких частот. Проблема заключается в том, что синк функция, не опускаясь до нулевой амплитуды, продолжается в обе стороны отрицательную и положительную бесконечно. Хотя эта бесконечная длина не представляет проблемы для *математики*, она является стопором для работы *компьютеров*.

Чтобы обойти эту проблему, мы сделаем две модификации синк функции на рис. 16.1b, которые дадут форму волны, показанную на рис. 16.1c. Прежде всего, она усечена до *M+1* точек, симметрично выбранных вокруг основного лепестка, где *M* – четное число. Все отсчеты за пределами этих *M+1* точек установлены в нулевое значение или просто игнорируются. Во-вторых, вся последовательность сдвигается вправо таким образом, чтобы она изменялась от *0* до *M*. Это позволяет представить ядро фильтра, пользуясь только *положительными* индексами. Хотя многие языки программирования позволяют применять *отрицательные* индексы, они неудобны для использования. Единственным эффектом от такого сдвига на *M/2* в ядре фильтра является сдвиг выходного сигнала на туже самую величину.

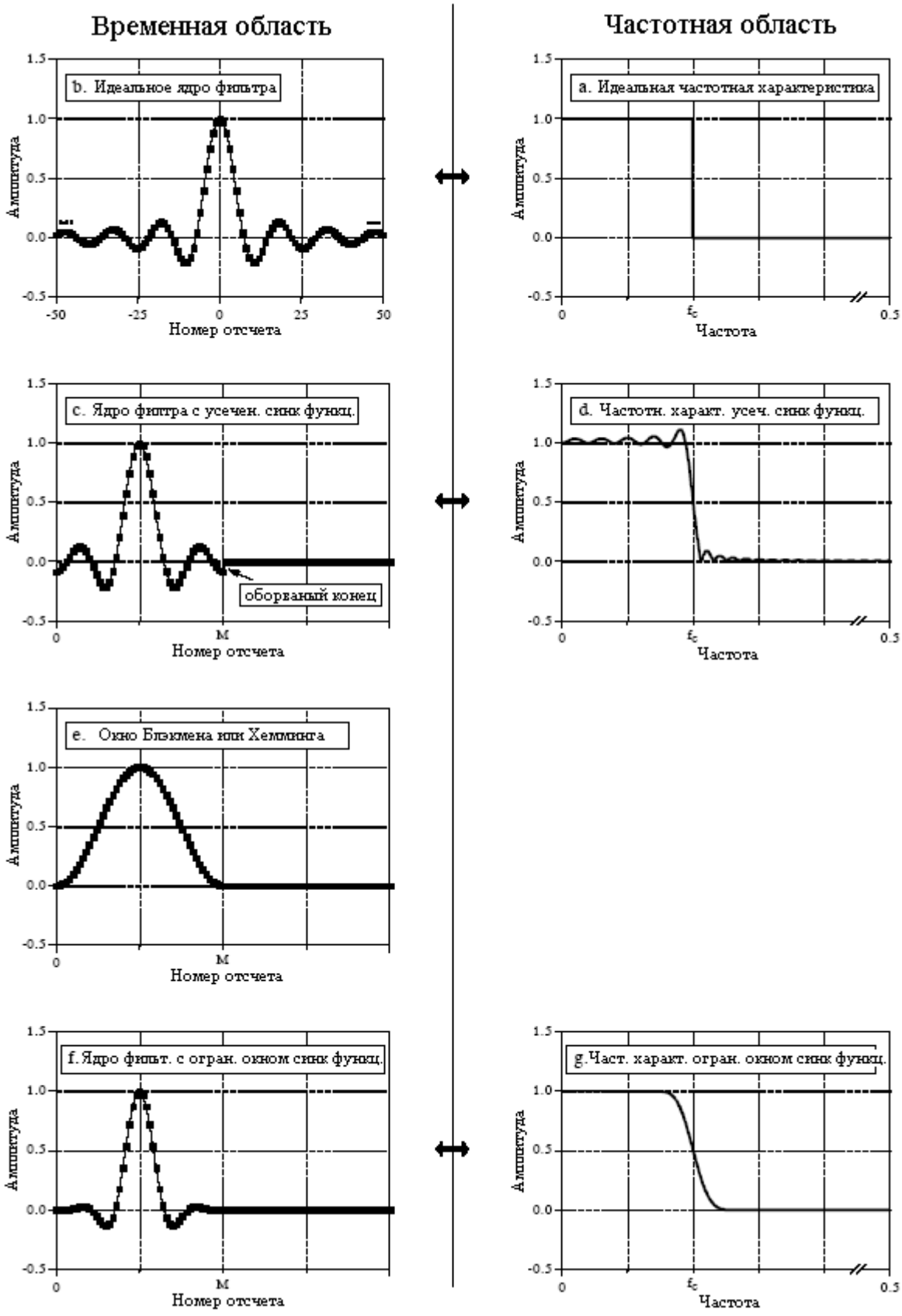


Рис. 16.1 Фильтр с ограниченной окном синк функцией

Поскольку модифицированное ядро фильтра представляет собой всего лишь аппроксимацию идеального ядра фильтра, оно не будет обладать идеальной частотной

характеристикой. Для того чтобы найти полученную частотную характеристику, необходимо взять преобразование Фурье от сигнала представленного на рис. 16.1c, что даст в результате кривую на рис. 16.1d. Это - беда! В полосе пропускания присутствует лишняя пульсация и слабое ослабление в полосе заграждения (вспомните эффект Гиббса обсуждаемый в Главе 11). Эти проблемы возникают из-за резкого разрыва на концах усеченной синк функции. Увеличение длины ядра фильтра не снижает эти проблемы; разрыв существенен в независимости от того, какова длина *M*.

К счастью существует простой метод улучшения этой ситуации. На рис. 16.1e показана плавно сужающаяся кривая, называемая **окном Блэкмена**. Умножение усеченной синк функции (рис. 16.1c) на окно Блэкмена (рис. 16.1e) дает ядро фильтра **ограниченной окном синк функции,** показанное на рис. 16.1f. Идея заключается в том, чтобы уменьшить обрывистость усеченных концов и, таким образом, улучшить частотную характеристику. Рис. 16.1g показывает это улучшение. Теперь полоса пропускания плоская, а ослабление в полосе заграждения настолько хорошее, что его нельзя даже увидеть на этом рисунке.

Существует несколько разных окон, большинство из которых названо в честь их первоначальных разработчиков в 1950-х. Но только два из них стоит использовать **окно Хемминга** и **окно Блэкмена**. Они задаются с помощью:

$$w[i] = 0{,}54 - 0{,}46\cos(2\pi i / M), \text{ для } i=0 \text{ до } M \text{ для всех } M+1 \text{ точек;} \qquad (16.1)$$

$$w[i] = 0{,}42 - 0{,}5\cos(2\pi i / M) + 0{,}08\cos(4\pi i / M). \qquad (16.2)$$

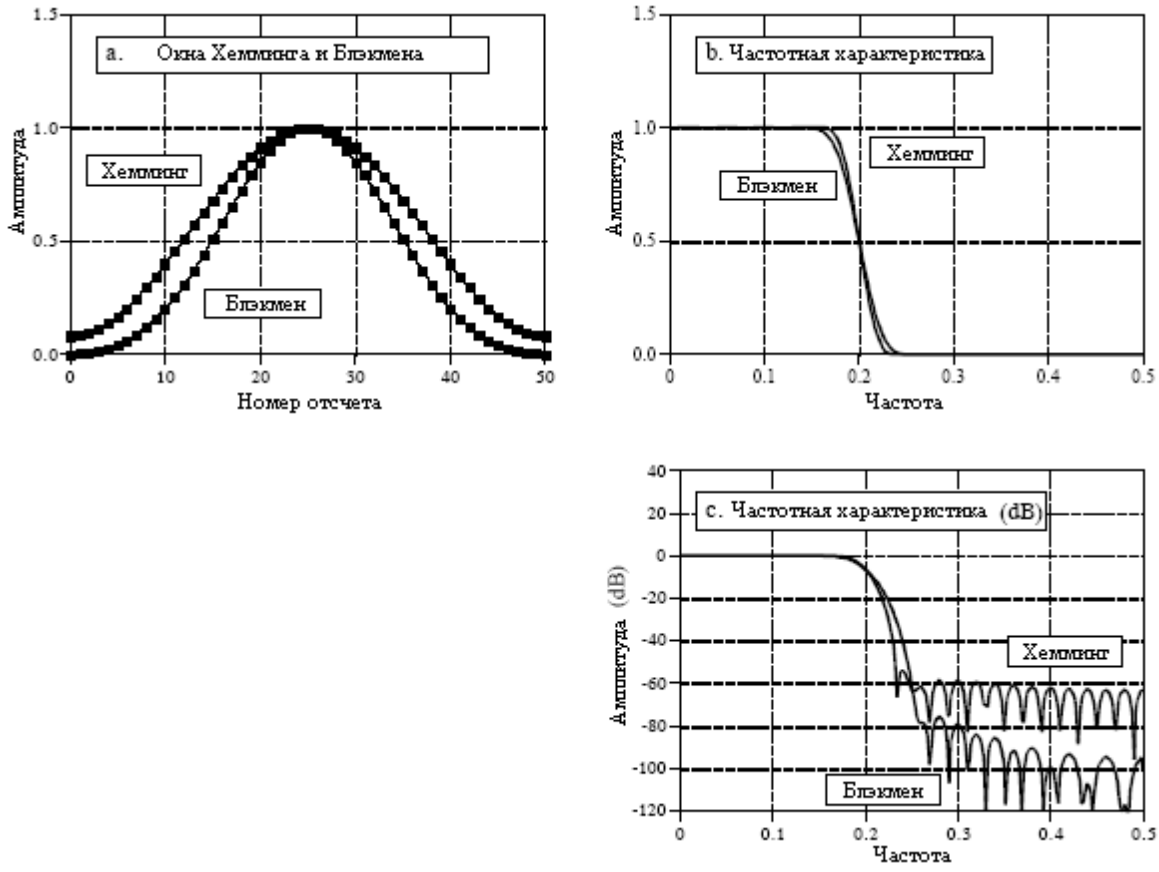


Рис. 16.2 Характеристики окон Блэкмена и Хемминга

На рис. 16.2 показаны кривые этих двух окон для $M=50$ (т.е. всего *51* точка в кривой). Какое из этих двух окон использовать Вам? Это компромисс между параметрами. Как показано на рис. 16.2b, окно Хемминга имеет приблизительно на *20%* более быстрый *завал*, чем окно Блэкмена. Однако, на рис. 16.1c показано, что окно Блэкмена имеет лучшее ослабление в *полосе заграждения*. Чтобы быть точным ослабление в полосе заграждения для окна Блэкмена составляет *–74* dB (*-0,02%*), в то время как у окна Хемминга всего *-53*dB (*-0.2%*). Хотя на этом графике этого и нельзя увидеть, окно Блэкмена имеет *пульсации в полосе пропускания* порядка приблизительно *0,02%*, в то время как для окна Хемминга типично значение *0,2%*. Вообще, Вашим первым выбором должно быть окно Блэкмена; медленный завал обрабатывать легче, чем слабое ослабление в полосе заграждения.

Существуют и другие окна, о которых Вы, возможно, слышали, хотя они и теряют по сравнению с окнами Блэкмена и Хемминга. **Окно Бартлетта** является треугольником, использующим для сужения прямые линии. **Окно Ханнинга**, называемое также **окно поднятого косинуса**, задается как: $w[i]=0,5-0,5cos(2\pi i/M)$. Эти два окна имеют примерно одинаковую скорость завала, как и у окна Хемминга, но более плохое ослабление в полосе заграждения (Окно Бартлетта: *-25* dB или *5,6%*, окно Ханнинга: *-44* dB или *0,63%*). Вы, может быть, также слышали о **прямоугольном окне**. Это то же самое, что и отсутствие какого-либо окна, только лишь усечение концов (так, как на рис. 16.1c). Хотя завал в 2,5 раза быстрее, чем у окна Блэкмена, ослабление в полосе заграждения всего *–21* dB (*8,9%*).

Проектирование фильтра

Для проектирования ограниченной окном синк функции должны быть выбраны два параметра: частота среза f_C и длина ядра фильтра M. Частота среза выражается в долях частоты дискретизации и, следовательно должна лежать между *0* и *0,5*. Величина M устанавливает *скорость завала* в соответствии со следующей аппроксимацией:

$$M \approx \frac{4}{BW}, \qquad (16.3)$$

где BW – является шириной полосы перехода измеренной от того места, где кривая едва-едва покидает уровень единицы до того места, где она почти достигает нуля (скажем от 99% до 1% кривой). Полоса перехода также выражается в долях частоты дискретизации и должна лежать между *0* и *0,5*. На рис. 16.3 показан пример того, как используется эта аппроксимация. Приведенные три кривые, сгенерированные из ядер фильтров с: $M = 20$, *40*, и *200*. Полоса перехода из уравнения (16.3) соответственно: $BW = 0,2$, *0,1*, и *0,02*. На рис. 16.3b показано, что форма частотной характеристики не зависит от выбранной частоты среза.

Поскольку время, необходимое для выполнения свертки пропорционально длине сигнала, уравнение (16.3) выражает компромисс между *временем вычисления* (зависит от значения M) и *крутизной полосы перехода* фильтра (значение BW). Например, *20%* более медленного завала окна Блэкмена (по сравнению с окном Хемминга) можно скомпенсировать, используя на *20%* более длинное ядро фильтра. Другими словами, можно сказать, что окно Блэкмена исполняется на *20%* медленнее, чем окно Хемминга с эквивалентным завалом. Это важно, потому что скорость исполнения у фильтров с ограниченной окном синк функцией - уже ужасно медленная.

Как также показано на рис. 16.3b, частота среза фильтра с ограниченной окном синк функцией определяется в точке с *половинной амплитудой*. Почему вместо стандартных *0,707* (*-3* dB), принятых в аналоговой электронике и других цифровых фильтрах, используется *0,5*? Это связано с тем, что частотная характеристика

ограниченной окном синк функции *симметрична* между полосой пропускания и полосой заграждения. Например, окно Хемминга дает порядка *0,2%* пульсацию в полосе пропускания и *идентичное* порядка *0,2%* ослабление в полосе заграждения (пульсации в полосе заграждения). Другие фильтры не демонстрируют такой симметрии и, следовательно, не имеют никакого преимущества в использовании для определения частоты среза точки половинной амплитуды. Как будет показано позже в этой главе, такая симметрия делает ограниченную окном синк функцию идеальной для *инверсии спектра*.

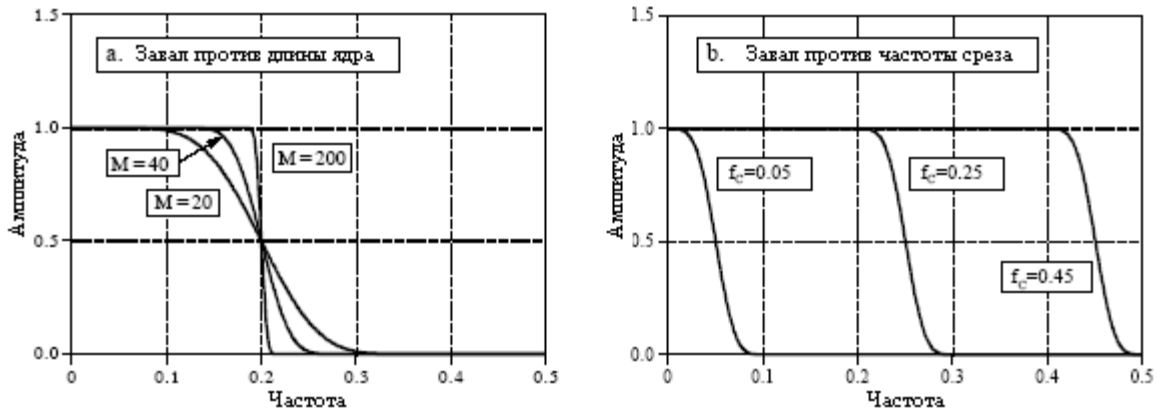


Рис. 16.3 Длина фильтра против завала фильтра с ограниченной окном синк функцией

После того как были выбраны f_C и M, вычисляется ядро фильтра из соотношения:

$$h[i] = K\frac{\sin(2\pi f_c(i - M/2))}{i - M/2}\left[0{,}42 - 0{,}5\cos\left(\frac{2\pi i}{M}\right) + 0{,}08\cos\left(\frac{4\pi i}{M}\right)\right], \quad (16.4)$$

Не пугайтесь этого уравнения! Опираясь на предыдущее обсуждение, Вы должны суметь выделить три компоненты: *синк функцию*, *сдвиг на M/2*, и *окно Блэкмена*. Для того чтобы обеспечить единичный коэффициент усиления на нулевой частоте, постоянная *K* должна быть выбрана таким образом, чтобы сумма всех отсчетов была равна единице. На практике, во время вычисления ядра фильтра *K* игнорируется, а затем, если это необходимо, все отсчеты нормализуются. В программе приведенной в таблице 16.1 показано, как это делается. Обратите внимание на то, как выполняются вычисления в центральной точке синк функции *i=M/2*, включающей в себя деление на ноль (Чтобы избежать деления на ноль для *i=M/2*, используется $h[i]=2\pi f_C K$.).

Может быть, это и длинное уравнение, но его довольно легко использовать, просто введите его в Вашу компьютерную программу и забудьте о нем. Позвольте компьютеру самому обрабатывать вычисления. Если Вы поймаете себя на том, что пытаетесь вручную оценивать это уравнение, Вы делаете что-что очень и очень неправильное.

Давайте определимся относительно того, где в Вашем компьютерном массиве расположено ядро фильтра, описанное уравнением 16.4. В качестве примера выберем *M* равным *100*. Помните *M* должно быть четным числом. Первая точка в ядре фильтра находится в массиве в местоположении с номером *0*, в то время как последняя точка находится в местоположении массива с номером *100*. Это означает, что весь сигнал длиной в *101* точку. Центр симметрии находится в точке *50*, т.е. *M/2*. *50* точек слева от точки *50* симметричны 50 точкам справа. Точка *0* имеет то же значение, что и точка *100*, а точка *49* имеет то же значение, что и точка *51*. Если Вам необходимо иметь в ядре фильтра определенное число отсчетов, такое чтобы использовать БПФ, то просто к одному или другому концу добавьте нули. Например, для *M=100* вы можете сделать нулевыми отсчеты со *101* по *127*, что в результате дает ядро фильтра длиной *128* точек.

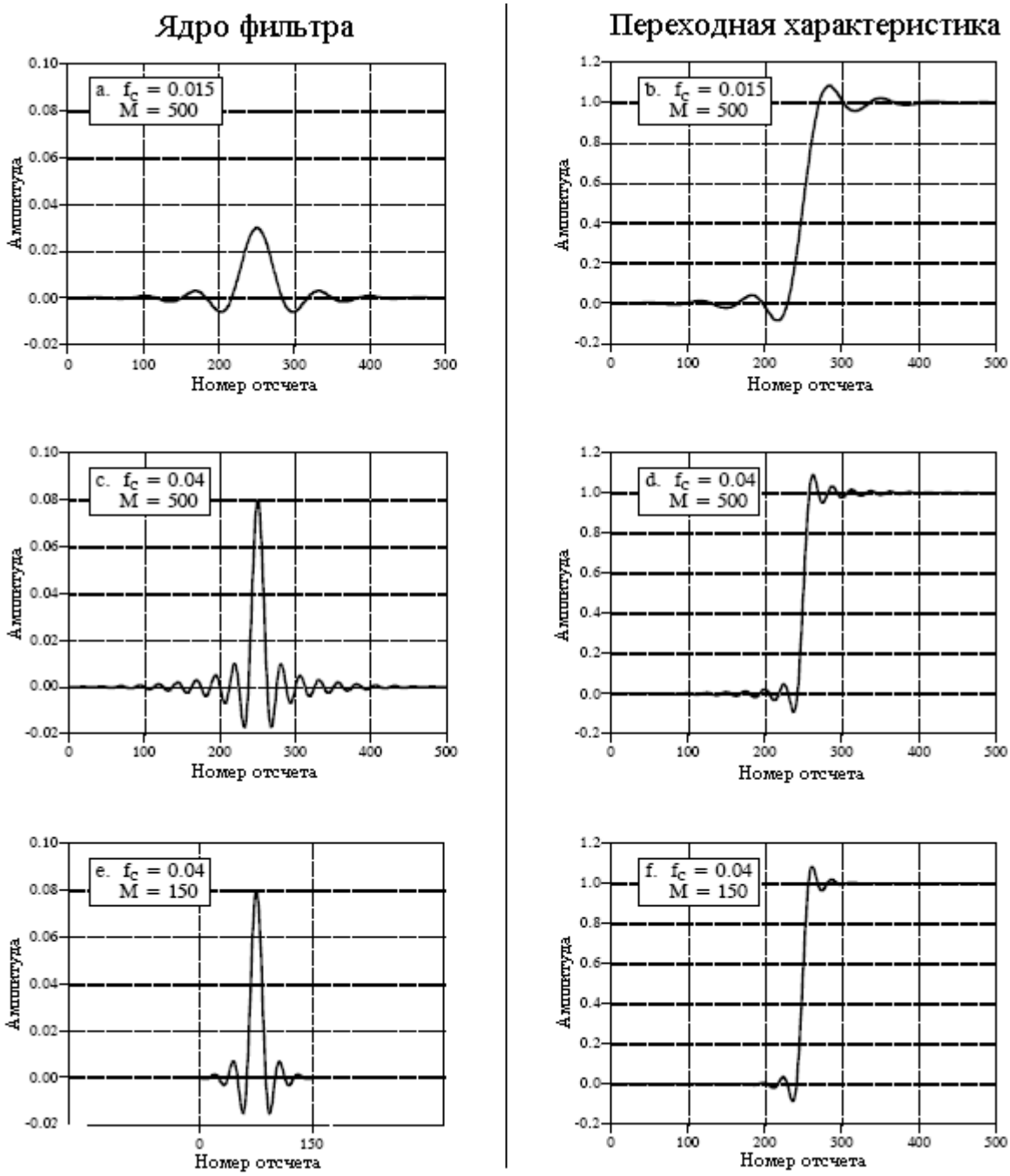


Рис. 16.4 Пример ядер фильтра и соответствующих переходных характеристик

На рис. 16.4 показаны примеры ядер фильтров с ограниченной окном синк функцией и их соответствующих переходных характеристик. Отсчеты в начале и в конце ядер фильтров настолько малы, что их нельзя даже увидеть на графиках. Не допускайте ошибку, думая, что они не важны! Эти отсчеты могут быть маленькими по значению, однако, все вместе они оказывают огромное влияние на работу фильтра. Вот еще почему, для реализации фильтров с ограниченной окном синк функцией обычно используется представление с плавающей запятой. Целые обычно не имеют достаточного динамического диапазона, чтобы охватить большое разнообразие значений содержащихся в ядре фильтра. А как фильтр с ограниченной окном синк функцией работает во временной области? Ужасно! Переходная характеристика имеет перерегулирование и звон; для сигналов с информацией, закодированной во временной области - это *не* фильтр.

Примеры фильтров с ограниченной окном синк функцией

Электроэнцефалограмма или ЭЭГ – это измерение электрической активности мозга. Она может быть обнаружена в виде сигналов уровня милливольт, появляющихся на электродах, приложенных к поверхности головы. Каждая нервная клетка мозга генерирует небольшой электрический импульс. ЭЭГ - совместный результат огромного числа этих электрических импульсов, производимых (надеемся) в скоординированной манере. Хотя взаимосвязь между мыслью и этим электрическим сочетанием очень плохо понятна, различные частоты в ЭЭГ могут быть поставлены в соответствие специфическим умственным состояниям. Если Вы закроете Ваши глаза и расслабитесь, преобладающим образцом ЭЭГ будет медленное колебание между приблизительно *7* и *12* герцами. Эта форма волны называется *альфа ритмом* и ассоциируется с удовлетворенностью и пониженным уровнем внимания. Открывание ваших глаз и осматривание вокруг заставляет ЭЭГ измениться до *бета ритма*, появляющегося между приблизительно 17 и 20 герцами. Иные частоты и формы волн наблюдаются у детей, при снах различной глубины и при различных мозговых расстройствах наподобие эпилепсии.

В этом примере мы примем, что сигнал ЭЭГ был усилен с помощью аналоговой электроники, а затем оцифрован с частотой дискретизации *100* отсчетов в секунду. Сбор данных в течение *50* секунд дает сигнал в *5000* точек. Нашей целью является отделение альфа ритма от бета ритма. Для того чтобы это выполнить, мы спроектируем цифровой фильтр низкой частоты с частотой среза *14* герц или *0,14* от частоты дискретизации. Ширину полосы перехода установим равной *4* герца или *0,04* от частоты дискретизации. Из уравнения (16.3), необходимое ядро фильтра будет около 101 точки в длину, и мы произвольно выберем для использования окно Хемминга. Программа в Таблице 16.1 показывает, как реализован такой фильтр. Частотная характеристика фильтра, полученная при помощи взятия преобразования Фурье от ядра фильтра, показана на рис. 16.5.

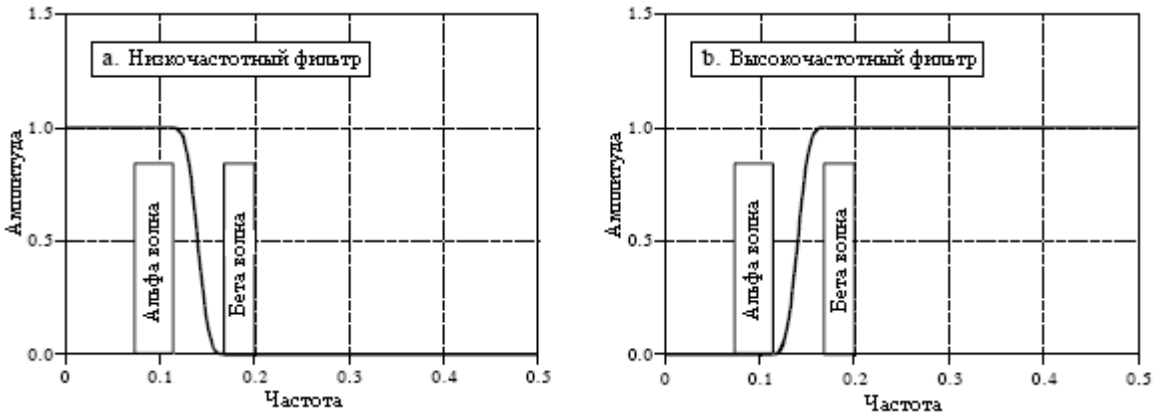


Рис. 16.5 Пример фильтров с ограниченной окном синк функцией

Во втором примере, мы спроектируем *полосно-пропускающий* фильтр для выделения *передаваемого тона* в звуковом сигнале подобного тому, который возникает при нажатии кнопки на телефоне (имеется в виду тональный набор номера – прим. перев.). Предположим, что сигнал был оцифрован на частоте *10* кГц, а целью является выделение полосы частот шириной *80* герц, сосредоточенных вокруг *2* кГц. В терминах частоты дискретизации, мы хотим заблокировать все частоты ниже *0,196* и выше *0,204* (что отвечает *1960* Гц и *2040* Гц, соответственно). Чтобы достичь ширины полосы перехода в *50* Гц (*0,005* от частоты дискретизации) сделаем ядро фильтра *801* точку длиной и воспользуемся окном Блэкмена. Таблица 16.2 содержит программу для вычисления ядра фильтра, в то время как частотная характеристика фильтра показана на рис. 16.6. Проектирование включает несколько шагов. Во-первых, проектируются *два*

низкочастотных фильтра, один с частотой среза *0,196*, а другой с частотой среза *0,204*. Затем осуществляется инверсия спектра этого второго фильтра, превращая его в высокочастотный фильтр (см. Главу 14 рис. 14.6). После чего ядра обоих фильтров складываются, давая в результате полосно-заграждающий фильтр (см. рис. 14.9). Наконец, еще одна *инверсия спектра* делает его желаемым полосно-пропускающим фильтром.

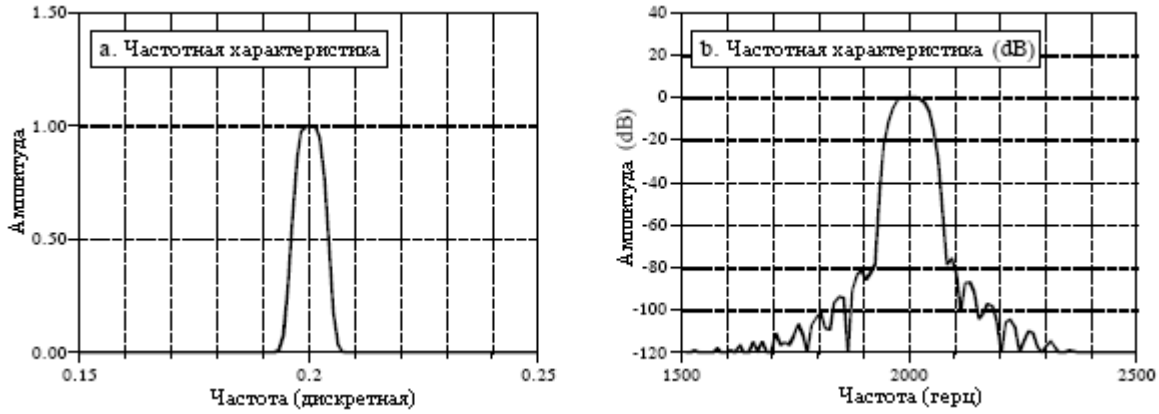


Рис. 16.6 Пример полосно-пропускающего фильтра с ограниченной окном синк функцией

Таблица 16.1

```
100 'LOW-PASS WINDOWED-SINC FILTER
110 'This program filters 5000 samples with a 101 point windowed-sinc filter,
120 'resulting in 4900 samples of filtered data.
130 '
140 DIM X[4999]                    'X[ ] holds the input signal
150 DIM Y[4999]                    'Y[ ] holds the output signal
160 DIM H[100]                     'H[ ] holds the filter kernel
170 '
180 PI = 3.14159265
190 FC = .14                       'Set the cutoff frequency (between 0 and 0.5)
200 M% = 100                       'Set filter length (101 points)
210 '
220 GOSUB XXXX                     'Mythical subroutine to load X[ ]
230 '
240 '                              'Calculate the low-pass filter kernel via Eq. 16.4
250 FOR I% = 0 TO 100
260 IF (I%-M%/2) = 0 THEN H[I%] = 2*PI*FC
270 IF (I%-M%/2) <> 0 THEN H[I%] = SIN(2*PI*FC * (I%-M%/2)) / (I%-M%/2)
280 H[I%] = H[I%] * (0.54 - 0.46*COS(2*PI*I%/M%) )
290 NEXT I%
300 '
310 SUM = 0                        'Normalize the low-pass filter kernel for
320 FOR I% = 0 TO 100              'unity gain at DC
330 SUM = SUM + H[I%]
340 NEXT I%
350 '
360 FOR I% = 0 TO 100
370 H[I%] = H[I%] / SUM
380 NEXT I%
```

```
390 '
400 FOR J% = 100 TO 4999            'Convolve the input signal & filter kernel
410 Y[J%] = 0
420 FOR I% = 0 TO 100
430 Y[J%] = Y[J%] + X[J%-I%] * H[I%]
440 NEXT I%
450 NEXT J%
460 '
470 END
```

Доводя его до предела

Фильтр с ограниченной окном синк функцией может быть доведен до невероятных уровней работы без неприятных сюрпризов. Например, предположим Вам нужно выделить сигнал в *1 милливольт*, считанный с линии электропередачи напряжением *120 вольт*. Понадобится низкочастотный фильтр с ослаблением в полосе подавления, по крайней мере *–120* dB (одна часть на один миллион - для тех, кто отказывается изучать децибелы). Как было показано ранее, окно Блэкмена обеспечивает *–74* dB (одна часть на пять тысяч). К счастью, большое ослабление в полосе заграждения легко получить. Входной сигнал может быть отфильтрован, за счет использования обычного ядра фильтра с ограниченной окном синк функцией, обеспечивающего промежуточный сигнал. Затем промежуточный сигнал может быть пропущен через фильтр второй раз, приводя к дальнейшему увеличению ослабления в полосе заграждения до *–148* dB (*1* часть в *30* миллионах, ничего себе!). Можно также объединить эти два этапа в один фильтр. Ядро объединенного фильтра эквивалентно *свертке* ядер фильтров двух этапов. Это означает также, что свертка любого ядра фильтра *с самим собой* дает ядро фильтра с гораздо улучшенным ослаблением в полосе заграждения. Цена, которую Вы платите - более длинное ядро фильтра и более медленный завал. На рис. 16.7a показана *201* точечная частотная характеристика фильтра низких частот сформированного при помощи свертки с самой собой *101* точечной ограниченной окном Блэкмена синк функции. Удивительная работа! (Если Вам действительно нужно ослабление в полосе заграждения более чем *–100* dB, Вы должны использовать двойную точность. Шум округления одинарной точности на сигнале в полосе пропускания может хаотично появляться в полосе заграждения с амплитудами в диапазоне от *–100* dB до *–120* dB.).

Таблица 16.2

```
100 'BAND-PASS WINDOWED-SINC FILTER
110 'This program calculates an 801 point band-pass filter kernel
120 '
130 DIM A[800]                      'A[ ] workspace for the lower cutoff
140 DIM B[800]                      'B[ ] workspace for the upper cutoff
150 DIM H[800]                      'H[ ] holds the final filter kernel
160 '
170 PI = 3.1415926
180 M% = 800                        'Set filter kernel length (801 points)
190 '
200 '                               'Calculate the first low-pass filter kernel via Eq. 16.4,
210 FC = 0.196                      'with a cutoff frequency of 0.196, store in A[ ]
220 FOR I% = 0 TO 800
230 IF (I%-M%/2) = 0 THEN A[I%] = 2*PI*FC
```

```
240 IF (I%-M%/2) <> 0 THEN A[I%] = SIN(2*PI*FC * (I%-M%/2)) / (I%-M%/2)
250 A[I%] = A[I%] * (0.42 - 0.5*COS(2*PI*I%/M%) + 0.08*COS(4*PI*I%/M%))
260 NEXT I%
270 '
280 SUM = 0                          'Normalize the first low-pass filter kernel for
290 FOR I% = 0 TO 800                'unity gain at DC
300 SUM = SUM + A[I%]
310 NEXT I%
320 '
330 FOR I% = 0 TO 800
340 A[I%] = A[I%] / SUM
350 NEXT I%
360 '                               'Calculate the second low-pass filter kernel via Eq. 16.4,
370 FC = 0.204                      'with a cutoff frequency of 0.204, store in B[ ]
380 FOR I% = 0 TO 800
390 IF (I%-M%/2) = 0 THEN B[I%] = 2*PI*FC
400 IF (I%-M%/2) <> 0 THEN B[I%] = SIN(2*PI*FC * (I%-M%/2)) / (I%-M%/2)
410 B[I%] = B[I%] * (0.42 - 0.5*COS(2*PI*I%/M%) + 0.08*COS(4*PI*I%/M%))
420 NEXT I%
430 '
440 SUM = 0                          'Normalize the second low-pass filter kernel for
450 FOR I% = 0 TO 800                'unity gain at DC
460 SUM = SUM + B[I%]
470 NEXT I%
480 '
490 FOR I% = 0 TO 800
500 B[I%] = B[I%] / SUM
510 NEXT I%
520 '
530 FOR I% = 0 TO 800               'Change the low-pass filter kernel in B[ ] into a high-pass
540 B[I%] = - B[I%]                 'filter kernel using spectral inversion (as in Fig. 14.5)
550 NEXT I%
560 B[400] = B[400] + 1
570 '
580 '
590 FOR I% = 0 TO 800               'Add the low-pass filter kernel in A[ ], to the high-pass
600 H[I%] = A[I%] + B[I%]           'filter kernel in B[ ], to form a band-reject filter kernel
610 NEXT I%                         'stored in H[ ] (as in Fig. 14.8)
620 '
630 FOR I% = 0 TO 800               'Change the band-reject filter kernel into a band-pass
640 H[I%] = -H[I%]                  'filter kernel by using spectral inversion
650 NEXT I%
660 H[400] = H[400] + 1
670 '                               'The band-pass filter kernel now resides in H[ ]
680 END
```

На рис. 16.7b показан еще один пример невероятной работы синк функции ограниченной окном: низкочастотный фильтр с 32001 точками в ядре. Как и ожидалось, появляется частотная характеристика с завалом *0,000125* от частоты дискретизации. Насколько хорош этот фильтр? Попытайтесь построить аналоговый электронный фильтр, пропускающий сигнал от постоянной составляющей до *1000* герц с разбросом менее чем *0,02%* и блокирующим все частоты выше *1001* герца с разбросом менее чем *0,02%*. Сейчас

- это фильтр! Если Вы действительно хотите быть поражены, вспомните, что оба фильтра на рис. 16.7 используют *одинарную точность*. Использование *двойной точности* позволяет расширить уровни этих рабочих характеристик в *миллионы* раз.

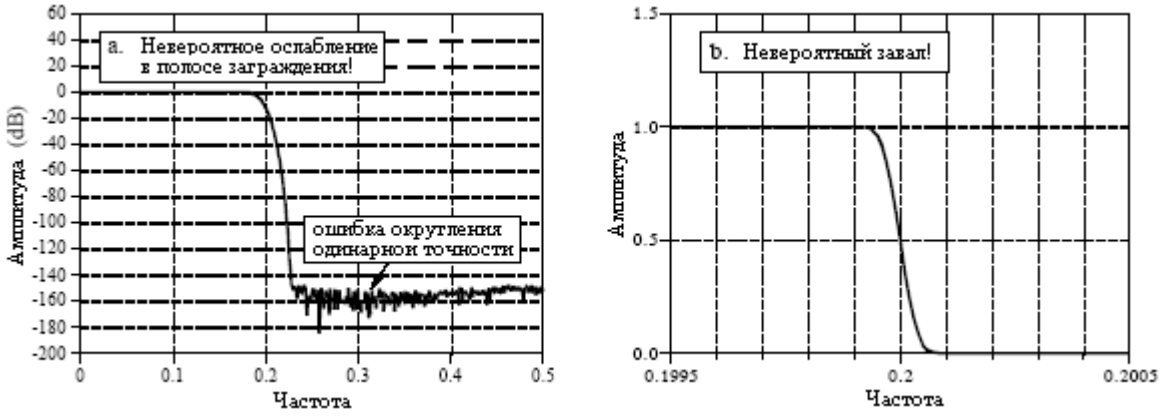


Рис. 16.7 Невероятные характеристики фильтра с ограниченной окном синк функцией

Самым сильным ограничением фильтра с ограниченной окном синк функцией является *время исполнения*; если ядро фильтра содержит большое число точек и используется стандартная свертка, оно может быть неприемлемо большим. Высокоскоростной алгоритм для этого фильтра (БПФ свертка) представлен в Главе 18. Рекурсивные фильтры (Глава 19) также обеспечивают хорошее разделение частот и являются разумной альтернативой фильтрам с ограниченной окном синк функцией.

Является ли ограниченная окном синк функция оптимальным ядром фильтра для разделения частот? Нет, ядра фильтра, вытекающие из более утонченных методов, могут быть лучше. Но будьте осторожны! Прежде, чем Вы окунетесь в эту очень математическую область, Вы должны точно определить то, что Вы надеетесь улучшить. Ограниченная окном синк функция обеспечит *любой* уровень работы, который возможно Вам нужен. Все, что могут дать продвинутые методы проектирования фильтра для заданного уровня работы, так это только слегка более короткое ядро фильтра. Это, в свою очередь, может означать слегка более высокую скорость исполнения. Не забывайте, что Вы можете получить слабую отдачу за потраченные усилия.

ГЛАВА
17

ЗАКАЗНЫЕ ФИЛЬТРЫ

Большинство фильтров имеют одну из четырех стандартных частотных характеристик: низкочастотную, высокочастотную, полосно-пропускающую или полосно-заграждающую. Эта глава представляет основной метод проектирования цифровых фильтров с *произвольной* частотной характеристикой, скроенной в соответствии с потребностями вашего конкретного приложения. В этой области ЦОС превосходит всех, решая проблемы, значительно превосходящие способности аналоговой электроники. В этой главе обсуждаются два важных применения заказных фильтров: *обратная свертка*, способ восстановления сигналов, которые подверглись нежелательной свертке, и *оптимальная фильтрация*, проблема отделения сигналов с перекрывающимися частотными спектрами. Это и есть ЦОС во всей своей красе.

Произвольная частотная характеристика

Подход, использованный в предыдущей главе для получения фильтра с ограниченной окном синк функцией, может быть также использован при проектировании фильтров фактически с *любой* частотной характеристикой. Единственное отличие заключается в том, как желаемая характеристика переносится из частотной области во временную. В фильтре с ограниченной окном синк функцией частотная характеристика и ядро фильтра - оба представлены *уравнениями,* и переход между ними осуществляется при помощи вычисления *математики* преобразования Фурье. В методе, показанном здесь, оба сигнала представлены *массивами чисел*, причем, для нахождения одного из другого используется *компьютерная программа* (БПФ).

На рис. 17.1 приведен пример того, как это все работает. Частотная характеристика, которую мы хотим, чтобы давал фильтр, приведена на рис. 17.1а. Мягко говоря, она очень нерегулярна и фактически с помощью аналоговой электроники ее было бы невозможно получить. Такая идеальная частотная характеристика задается массивом отобранных чисел, а не некоторым математическим уравнением. В этом примере взято *513* отсчетов, расположенных между *0* и *0,5* от частоты дискретизации. Для лучшего представления желаемой частотной характеристики должно быть использовано большее количество точек, в то время как для сокращения времени вычислений при проектировании фильтра может потребоваться меньшее число. Однако, подобные беспокойства обычно незначительны, и 513 - это хорошая длина для большинства приложений.

Кроме желаемого массива *модулей*, показанного на рис. 17.1а, должен быть соответствующий массив *фаз* такой же длины. В этом примере фаза желаемой частотной характеристики везде *нулевая* (этот массив не показан на рис. 17.1). Так же, как и массив модулей, массив фаз может быть загружен любой произвольной кривой, которую Вы желаете, чтобы давал фильтр. Однако помните, что первый и последний отсчеты (т.е. *0* и *512*) массива фаз должны иметь значение равное *нулю* (или, что одно и то же, кратны 2π). Вместо использования модуля и фазы частотная характеристика может быть также задана и в прямоугольной форме посредством задания входного массива для *действительной* и *мнимой частей*.

Следующий шаг – взять обратное ДПФ для перемещения фильтра во временную область. Наиболее быстрый способ это сделать – преобразовать частотную область в прямоугольную форму, а затем использовать обратное БПФ. Это даст *1024* отсчета сигнала, изменяющихся от *0* до *1023*, как показано на рис. 17.1b. Это - импульсная характеристика, соответствующая частотной характеристике, которую мы хотим получить, однако, она не подходит для использования в качестве ядра фильтра (вскоре, чуть подробнее об этом). Так же, как и в предыдущей главе, ее необходимо: *сдвинуть*, *усечь* и ограничить *окном*. В данном примере мы будем проектировать ядро фильтра с *M=40*, т.е. *41* точка, изменяющаяся от отсчета *0* до отсчета *40*. В таблице 17.1 приведена компьютерная программа, преобразующая сигнал на рис. 17.1b в сигнал на рис. 17.1c. Так же, как и в случае с фильтром с ограниченной окном синк функцией точки вблизи концов ядра фильтра настолько малы, что на графике они кажутся равными нулю. Не совершайте ошибку, думая, что их можно отбросить!

Таблица 17.1

```
100 'CUSTOM FILTER DESIGN
110 'This program converts an aliased 1024 point impulse response into an M+1 point
120 'filter kernel (such as Fig. 17.1b being converted into Fig. 17.1c)
130 '
140 DIM REX[1023]                   'REX[ ] holds the signal being converted
150 DIM T[1023]                     'T[ ] is a temporary storage buffer
160 '
170 PI = 3.14159265
180 M% = 40                         'Set filter kernel length (41 total points)
190 '
200 GOSUB XXXX                      'Mythical subroutine to load REX[ ] with impulse response
210 '
220 FOR I% = 0 TO 1023              'Shift (rotate) the signal M/2 points to the right
230 INDEX% = I% + M%/2
240 IF INDEX% > 1023 THEN INDEX% = INDEX%-1024
250 T[INDEX%] = REX[I%]
260 NEXT I%
270 '
280 FOR I% = 0 TO 1023
290 REX[I%] = T[I%]
300 NEXT I%
310 '                               'Truncate and window the signal
320 FOR I% = 0 TO 1023
330 IF I% <= M% THEN REX[I%] = REX[I%] * (0.54 - 0.46 * COS(2*PI*I%/M%))
340 IF I% > M% THEN REX[I%] = 0
350 NEXT I%
360 '                               'The filter kernel now resides in REX[0] to REX[40]
370 END
```

Последним шагом является *проверка* ядра фильтра. Она выполняется при помощи взятия ДПФ (используя БПФ) для нахождения действительной частотной характеристики, как показано на рис. 17.1d. Для получения лучшего разрешения в частотной области перед взятием БПФ дополните ядро фильтра нулями. Например, использование всего *1024* отсчетов (*41* в ядре фильтра плюс *983* нуля) в результате дает *513* отсчетов между *0* и *0,5*.

Как показано на рис. 17.2, длина ядра фильтра определяет, насколько хорошо *фактическая* частотная характеристика соответствует *желаемой* частотной

характеристике. Необыкновенная работа цифровых КИХ-фильтров очевидна; фактически, если используется достаточно длинное ядро фильтра, может быть получена любая частотная характеристика.

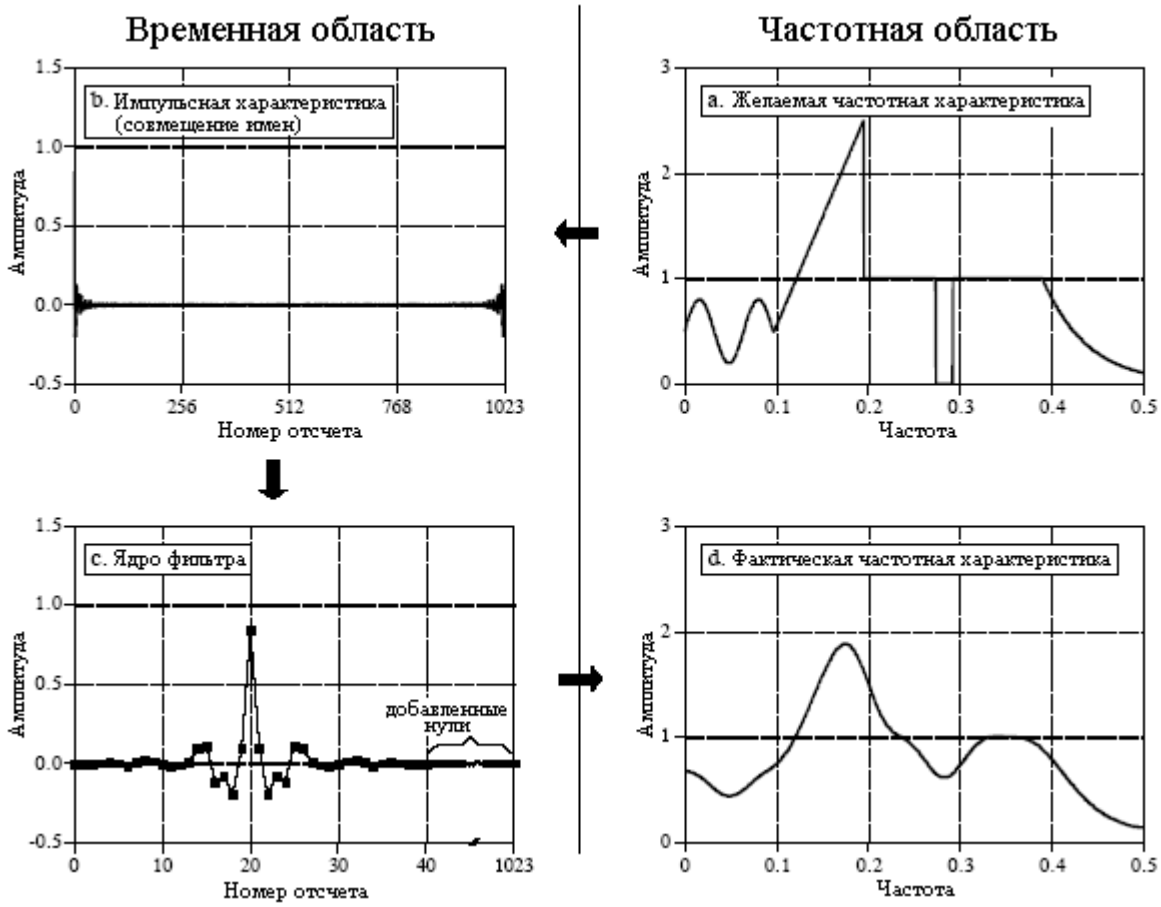


Рис. 17.1 Пример проектирования КИХ-фильтра

Это общий способ проектирования, однако, есть ряд тонких *теоретических* моментов, которые требуют разъяснений. Почему нельзя в качестве ядра фильтра непосредственно использовать импульсную характеристику, показанную на рис. 17.1b? В конце концов, если рис. 17.1a является преобразованием Фурье рис. 17.1b, будет ли свертка входного сигнала с рис. 17.1b давать *точную* частотную характеристику, которую мы хотим? Ответом будет - *нет*, и вот почему.

При проектировании заказного фильтра желаемая частотная характеристика определена значениями ряда. Теперь рассмотрите следующее: что делает частотная характеристика *между* конкретными точками? Для простоты можно представить два случая один “хороший” и один “плохой”. В “хорошем” случае частотная характеристика между заданными отсчетами - плавная кривая. В “плохом” случае меду отсчетами – дикие колебания. К счастью или к несчастью, импульсная характеристика, рис. 17.1b, соответствует “плохой” частотной характеристике. Это может быть показано при помощи дополнения ее большим количеством нулей и затем взятием ДПФ. Частотная характеристика, полученная этим методом, покажет неправильное поведение между первоначально определенными отсчетами и будет выглядеть просто ужасно.

Для того чтобы это понять, представьте, что мы вынуждаем частотную характеристику быть тем, чем мы хотим, определяя ее на бесконечном числе точек между *0* и *0,5*. То есть мы создаем непрерывную кривую. Затем, для нахождения импульсной характеристики, которая будет *бесконечной* в длину, используется обратное ДВПФ. Другими словами, “хорошая” частотная характеристика соответствует чему-то, что не

может быть представлено в компьютере, бесконечно длинной импульсной характеристике. Когда мы представляем частотный спектр с помощью *N/2+1* отсчетов, во временной области обеспечивается всего *N* отсчетов, что делает временную область неспособной правильно содержать сигнал. В результате бесконечно длинная импульсная характеристика накручивается (совмещение имен) на эти *N* точек. Когда происходит такое совмещение имен, частотная характеристика меняется с “хорошей” на “плохую”. К счастью, ограничение окном *N* точечной импульсной характеристики значительно снижает это совмещение имен, давая плавную кривую между отсчетами частотной области.

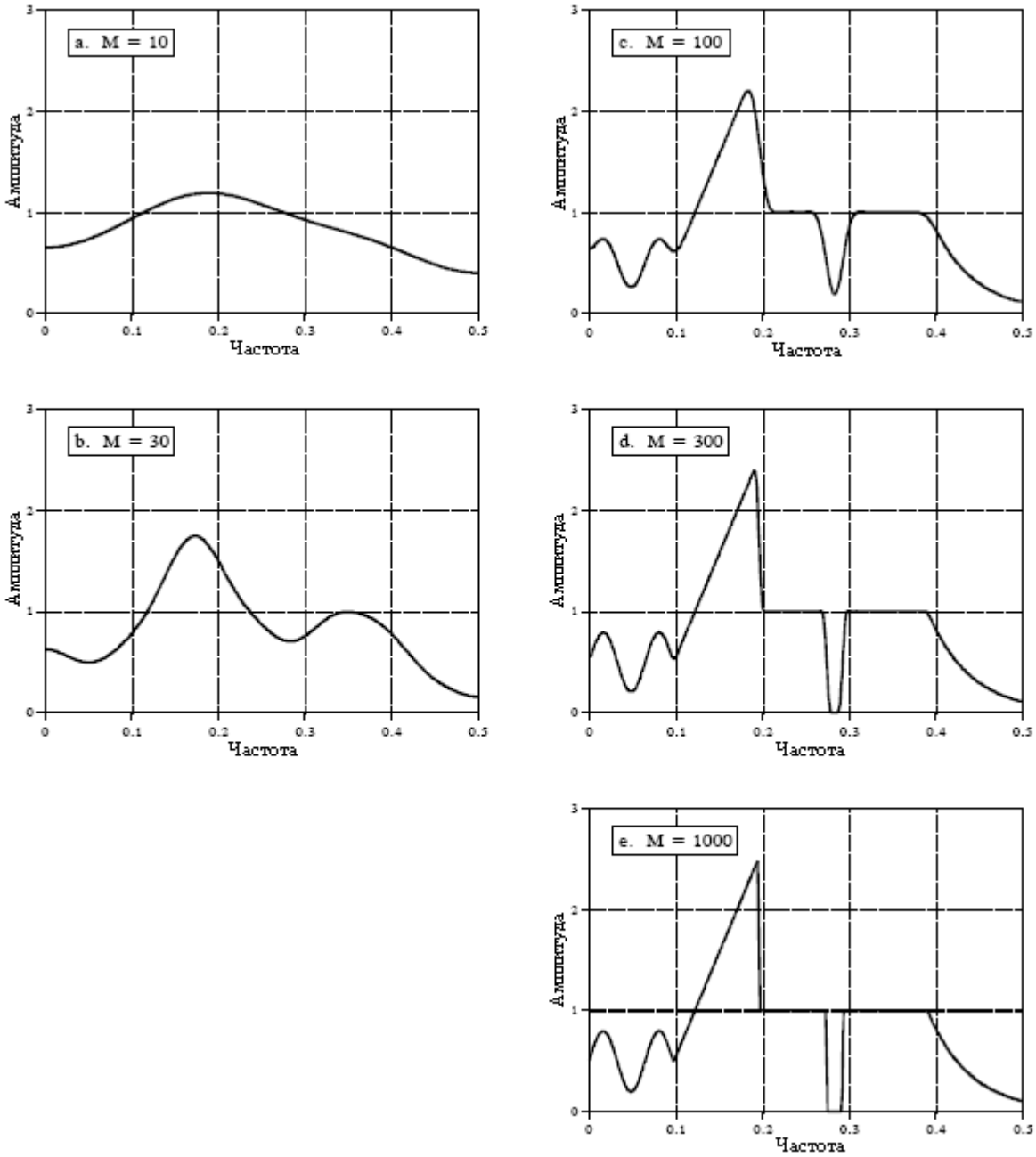


Рис. 17.2 Частотная характеристика против длинны ядра фильтра

Проектирование цифрового фильтра для получения заданной частотной характеристики весьма просто. Трудной частью является поиск того, какую частотную характеристику использовать. Давайте посмотрим на некоторые стратегии, используемые в ЦОС для проектирования заказных фильтров.

Обратная свертка

Нежелательная свертка является проблемой свойственной передаче аналоговой информации. Например, все следующее может быть интерпретировано как свертка: смазанность изображения в трясущемся фотоаппарате, эхо в междугороднем телефонном разговоре, ограниченная полоса пропускания аналоговых датчиков и электроники и т.д. Обратная свертка является процессом фильтрации сигнала компенсирующим нежелательную свертку. Целью обратной свертки является восстановление сигнала в том виде, в котором он существовал *до того* момента, когда произошла свертка. Обычно это требует знания характеристик свертки (т.е. импульсной или частотной характеристик). Такую обратную свертку следует отличать от **обратной свертки вслепую**, где характеристики паразитной свертки являются *неизвестными*. Обратная свертка вслепую представляет собой значительно более сложную проблему, не имеющую общего решения, и здесь подход к ней должен быть построен в соответствии с конкретным приложением.

Понять обратную свертку *во временной области* почти невозможно, но *в частотной области* она вполне доступна пониманию. Каждая синусоида, составляющая исходный сигнал, проходя через нежелательную свертку, может быть изменена по амплитуде и/или фазе. Для выделения исходного сигнала фильтр обратной свертки должен *ликвидировать* эти изменения амплитуды и фазы. Например, если свертка изменяет амплитуду синусоиды до *0,5* с фазовым сдвигом в *30* градусов, фильтр обратной свертки должен усилить синусоиду в *2* раза с изменением фазового сдвига на *–30* градусов.

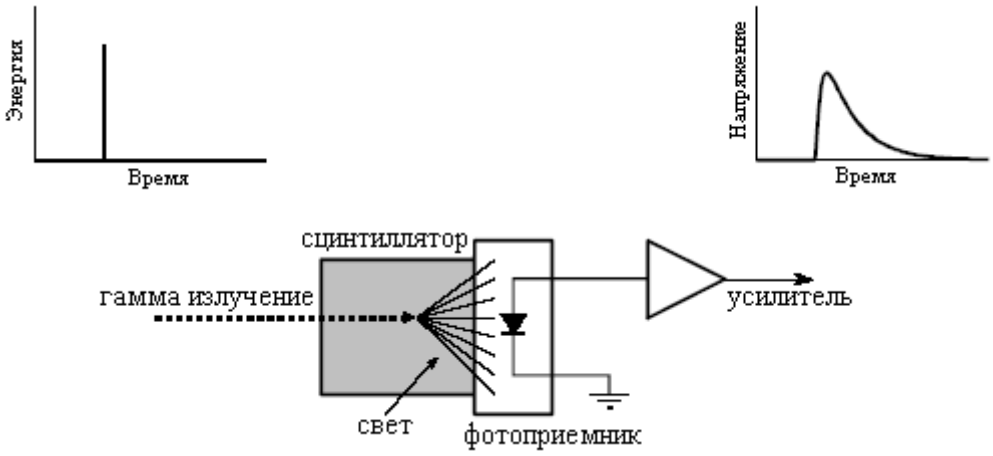


Рис. 17.3 Пример неизбежной свертки

Примером, который мы будем использовать для иллюстрации обратной свертки, является *детектор гамма излучения*. Как показано на рис. 17.3, этот прибор состоит из двух частей *сцинтиллятора* и *фотоприемника*. Сцинтиллятор представляет собой специальный тип прозрачного материала, типа йодида натрия или германита висмута. Эти соединения преобразуют энергию каждого гамма кванта в короткую вспышку видимого света. Затем при помощи светового детектора, такого как фотодиод или фотоэлектронный умножитель, этот свет преобразуется в электронный сигнал. Каждый импульс, производимый детектором, похож на *одностороннюю экспоненту* с несколько скругленными углами. Такая форма обусловливается характеристиками используемого сцинтиллятора. Когда гамма излучение привносит свою энергию в сцинтиллятор, близлежащие атомы возбуждаются до более высокого энергетического уровня. Случайным образом возвращаясь к *невозбужденному* состоянию, эти атомы испускают отдельный фотон видимого света. Общим результатом является импульс света с

затухающей амплитудой в течение нескольких сотен наносекунд (для йодида натрия). Поскольку появление каждого гамма кванта является *дельта импульсом*, выходной импульс от детектора (т.е. односторонняя экспонента) представляет собой *импульсный отклик* системы.

На рис. 17.4 показаны импульсы, генерируемые детектором в ответ на произвольно поступающее гамма излучение. Информацией, которую нам бы хотелось извлечь из этого выходного сигнала, является *амплитуда* каждого импульса, которая является пропорциональной *энергии* генерирующего импульс гамма излучения. Такая информация является полезной, поскольку энергия может рассказать интересные вещи о том, где побывало гамма излучение. Например, это может дать медицинскую информацию о пациенте, сообщить возраст отдаленной галактики, обнаружить бомбу в авиа багаже и т.д.

Если бы только случайное гамма излучение было бы обнаружено, все было бы хорошо, но это обычно не тот случай. Как показано на рис. 17.4a, два или более импульсов могут перекрываться, смещая измеренную амплитуду. Одним из ответов на эту проблему является *устранение свертки* выходного сигнала детектора, что делает импульсы более узкими настолько, что происходит меньше скоплений. В идеале, нам бы хотелось, чтобы каждый импульс походил на исходный дельта импульс. Как Вы, возможно, подозреваете, это невозможно, и мы должны согласиться на импульс конечной длины, но значительно короче, чем обнаруженный импульс. Данная цель иллюстрируется рис. 17.4b.

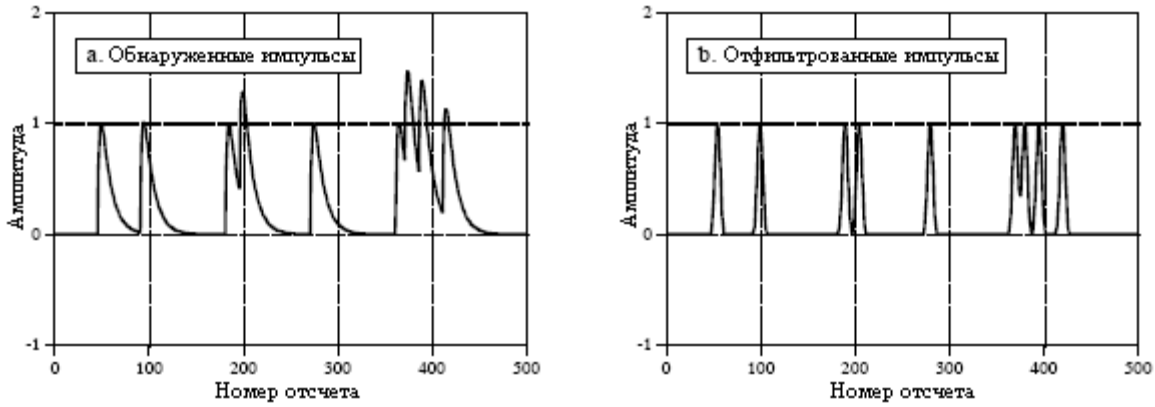


Рис. 17.4 Пример обратной свертки

Даже притом, что сигнал детектора содержит свою информацию закодированной во *временной области*, многое из нашего анализа должно быть проделано в *частотной области* там, где задачу легче понять. На рис. 17.5a показан сигнал, выдаваемый детектором (что-то, что нам известно). На рис. 17.5c показан сигнал, который нам бы хотелось иметь (также что-то, что нам известно). Этот желаемый импульс был произвольно выбран так, чтобы иметь туже самую форму, что и окно Блэкмена с длиной около одной третьей от первоначального импульса. Нашей целью является нахождение ядра фильтра (рис. 17.5e), свертка которого с сигналом на рис. 17.5a даст сигнал на рис. 17.5c. В виде уравнения: если $a * e = c$, а a и c заданы то, найти e.

Если бы вместо свертки эти сигналы объединялись при помощи сложения или умножения, решение было бы легким: чтобы выполнить операцию противоположную сложению используется *вычитание*, а *деление* используется, чтобы выполнить операцию противоположную умножению. Со сверткой другое дело, здесь нет простой противоположной операции, которую можно было бы назвать операцией обратной свертке. Свертка слишком сумбурна, чтобы можно было непосредственным манипулированием сигналами временной области выполнить обратную ей операцию.

К счастью, в частотной области эта проблема более проста. Помните, *свертка* в одной области соответствует *умножению* в другой области. Снова обратимся к сигналам

на рис. 17.5: если $b \times f = d$, а b и d заданы то, найти f. Эту задачу легко решить: частотная характеристика фильтра (рис. 17.5f) является частотным спектром желаемого импульса (рис. 17.5d), *деленным* на частотный спектр обнаруженного импульса (рис. 17.5b). Поскольку обнаруженный импульс является асимметричным, он будет иметь *ненулевую* фазу. Это означает, что должно быть использовано *комплексное* деление (т.е. модуль и фаза, деленные на другой модуль и фазу). В случае если Вы забыли, в Главе 9 дается определение того, как выполняется комплексное деление одного спектра на другой. Затем из частотной характеристики методом заказного фильтра (обратное ДПФ, сдвиг, усечение и умножение на окно) находится требуемое ядро фильтра (рис. 17.5e).

Существует предел улучшения, которое может дать обратная свертка. Другими словами, если Вы начинаете жадничать, все разваливается. В этом примере жадничать означает пытаться сделать желаемый импульс чрезмерно узким. Давайте посмотрим на то, что произойдет в этом случае. Если желаемый импульс сделан более узким, его частотный спектр должен содержать больше высокочастотных составляющих. Поскольку у этих высокочастотных составляющих в обнаруженном импульсе очень низкие амплитуды, на этих частотах фильтр должен обладать очень большим усилением. Например, на рис. 17.5f показано, что для достижения желаемого импульса на рис. 17.5c, некоторые частоты должны быть умножены на коэффициент равный *трем*. Если желаемый импульс сделать уже, усиление фильтра обратной свертки на высоких частотах будет еще больше.

Проблема в том, что в этой ситуации маленькие ошибки весьма непростительны. Например, если некоторая частота усилена в 30 раз, тогда как требуется только в 28, сигнал после устранения свертки, вероятно, будет представлять собой хаос. Когда обратная свертка доводится до высочайших уровней работы, характеристики нежелательной свертки должны быть поняты с большей *достоверностью* и *точностью*. В приложениях реального мира всегда существует неизвестность, вызванная такими злодеями как: шум электроники, температурный дрейф, разброс параметров приборов и т.д. Эти неизвестные величины устанавливают предел того, насколько хорошо будет работать обратная свертка.

Даже если нежелательная свертка совершенно понятна, все еще остается фактор, ограничивающий действие обратной свертки: *шум*. Например, большинство нежелательных сверток имеют вид низкочастотного фильтра, снижающего амплитуду высокочастотных составляющих сигнала. Обратная свертка осуществляет коррекцию этого за счет усиления этих частот. Однако если амплитуда этих составляющих падает ниже собственного шума системы, информация, содержащаяся в этих частотах, теряется. Никакая обработка сигнала не сможет ее восстановить. Она ушла навсегда. Всего хорошего! Счастливо оставаться! Прощайте! Попытка вернуть эту информацию обратно только усилит шум. В экстремальном случае, амплитуда некоторых частот может быть полностью уменьшена до *нуля*. Этот не только уничтожает информацию, это заставляет фильтр обратной свертки иметь *бесконечное* усиление на этих частотах. Решение: проектируйте менее агрессивный фильтр обратной свертки и/или установите ограничение на величину усиления на любой из частот.

Насколько далеко Вы можете пойти? Насколько жадным является слишком жадно? Это полностью зависит от задачи, которую Вы штурмуете. Если сигнал ведет себя хорошо и имеет низкий уровень шума то, вероятно, может быть сделано значительное улучшение (полагайте в 5-10 раз). Если сигнал изменяется во времени, не достаточно хорошо понятен или зашумлен, Вы не очень преуспеете (полагайте в 1-2 раза). Успешная обратная свертка подразумевает огромное количество тестов. Если это работает на одном уровне то, пытайтесь идти дальше; Вы увидите, когда все это развалится. Никакая теоретическая работа не позволит Вам обойти этот итеративный процесс.

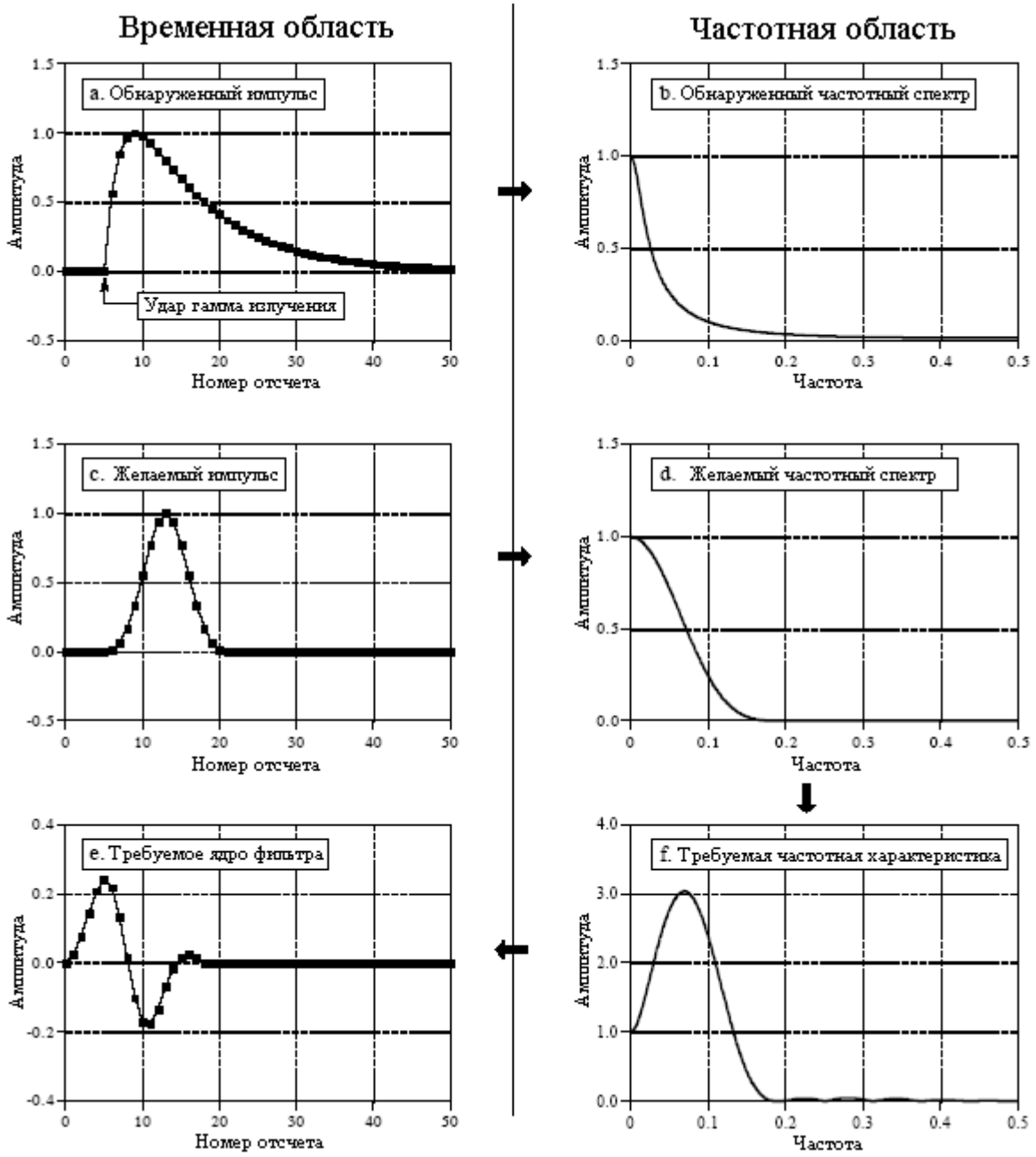


Рис. 17.5 Пример обратной свертки во временной и в частотной областях

Обратная свертка может быть также применена к сигналам с информацией закодированной в *частотной области*. Классическим примером является восстановление старых записей знаменитого оперного певца Энрико Карузо (1873-1921). Эти записи по современным стандартам были сделаны на очень примитивном оборудовании. Наиболее существенную проблему представляют здесь *резонансы* длинного трубчатого записывающего рупора, используемого для собирания звука. Всякий раз, когда случается, что певец берет одну из этих резонансных частот, громкость записи резко увеличивается. Цифровая обратная свертка улучшила субъективное качество этих записей за счет уменьшения громких всплесков в музыке. Мы только опишем общий метод; для детального описания см. оригинальную статью: T. Stockham, T. Cannon, and R. Ingebretsen, "Blind Deconvolution Through Digital Signal Processing", *Proc. IEEE*, vol. 63, Apr. 1975, pp. 678-692.

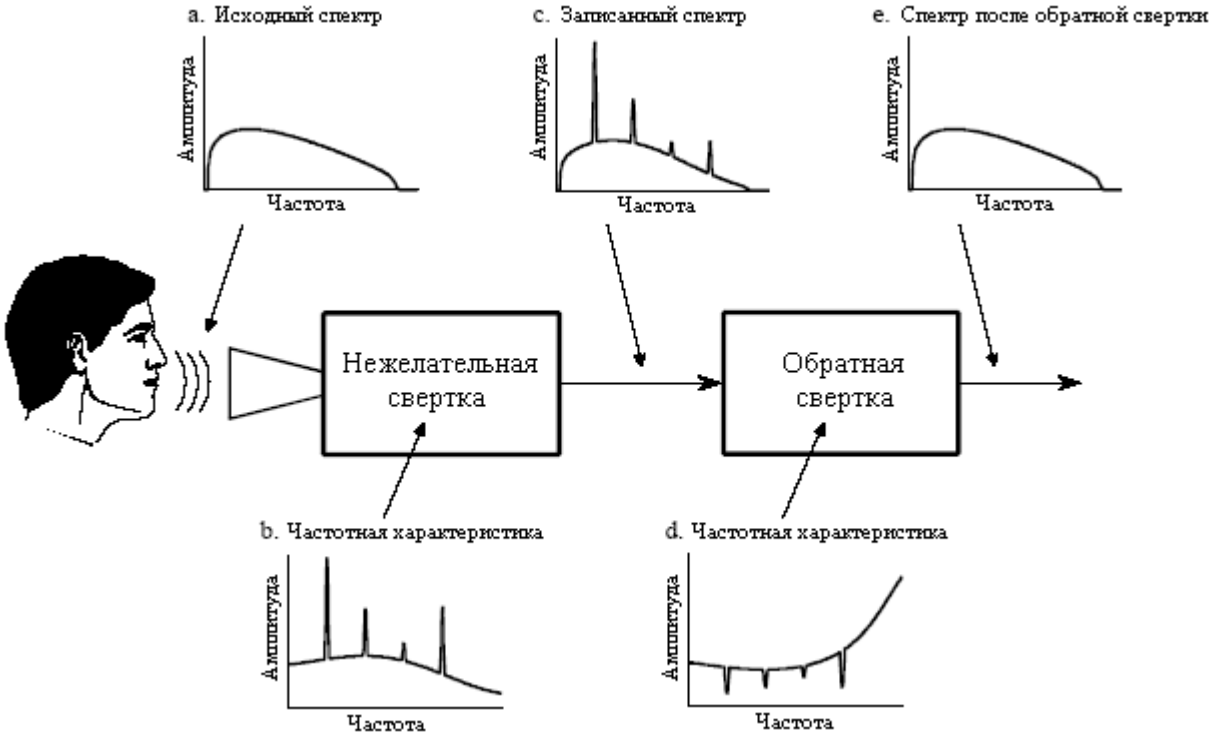


Рис. 17.6 Обратная свертка старых фонограмм

На рис. 17.6 показан общий подход. Частотный спектр исходного звукового сигнала иллюстрируется на рис. 17.6a. На рис. 17.6b показана частотная характеристика звукозаписывающего оборудования, относительно плавная кривая за исключением нескольких острых резонансных пиков. Спектр записанного сигнала показан на рис. 17.6c, он эквивалентен истинному спектру (рис. 17.6a), умноженному на неравномерную частотную характеристику, на рис. 17.6b. Целью обратной свертки является *противодействие* нежелательной свертке. Другими словами, частотная характеристика фильтра обратной свертки (рис.17.6d) должна быть *обратной* частотной характеристике, показанной на рис. 17.6b. То есть, каждый пик на рис. 17.6b поглощается соответствующей впадиной на рис. 17.6d. Если бы этот фильтр был спроектирован идеально, результирующий сигнал имел бы спектр, показанный на рис. 17.6e идентичный исходному. А вот и ловушка: оригинальное записывающее оборудование уже давным-давно выброшено, и его частотная характеристика (рис. 17.6b) является тайной. Другими словами это задача *обратной свертки вслепую*; задан только спектр на рис. 17.6c, каким же образом мы можем определить частотную характеристику, на рис. 17.6d?

Задачи обратной свертки вслепую обычно штурмуются с помощью приблизительных оценок или предположений относительно неизвестных параметров. Что касается данного примера, принимается, что *средний спектр* оригинальной музыки соответствует *среднему спектру* той же самой музыки, исполненной современным певцом, использующим современное оборудование. *Средний спектр* находится при помощи методов описанных в Главе 9: сигнал разбивается на большое число сегментов, от каждого сегмента берется ДПФ, преобразуется к полярному виду, а затем модули вместе усредняются. В простейшем случае, неизвестная частотная характеристика берется как средний спектр современной записи. (Метод, используемый Стокхэмом и другими, основан на более утонченной технике называемой *гомоморфной* обработкой, обеспечивающей лучшую оценку характеристик записывающей системы.)

Оптимальные фильтры

Рис. 17.7a иллюстрирует типичную проблему фильтрации: попытку извлечь форму волны (в данном примере экспоненциальный импульс), скрытую в случайном шуме. Как показано на рис. 17.7b, в частотной области эта проблема выглядит не легче. Сигнал обладает спектром, состоящим в основном из низкочастотных составляющих. Для сравнения спектр шума является *белым* (одинаковая амплитуда на всех частотах). Поскольку спектр сигнала и шума *перекрываются*, неясно, как эти двое могут быть наилучшим образом разделены. Фактически, действительным вопросом является то, как определить что означает "наилучшим". Посмотрим на три фильтра, каждый из которых некоторым образом является "наилучшим". На рис. 17.8 показано ядро фильтра и частотная характеристика для каждого из этих фильтров. На рис. 17.9 показан результат использования этих фильтров на примере формы волны, представленной на рис. 17.7a.

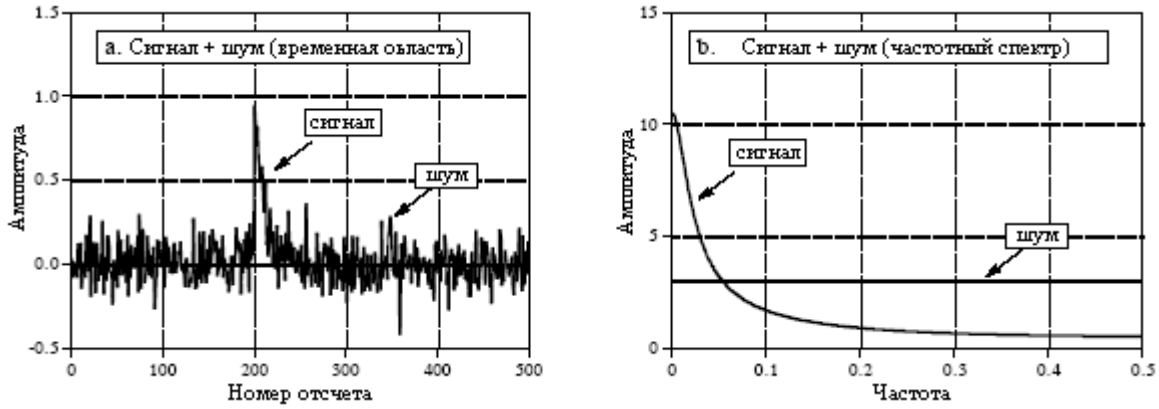


Рис. 17.7 Пример оптимальной фильтрации

Фильтр скользящего среднего является темой Главы 15. Как Вы помните, каждая выходная точка, вырабатываемая фильтром скользящего среднего, является средним значением некоторого числа точек входного сигнала. Последнее делает ядро фильтра, прямоугольным импульсом с амплитудой равной обратной величине числа точек в среднем. Фильтр скользящего среднего оптимален в том смысле, что он обеспечивает самую быструю переходную характеристику для заданного снижения шума.

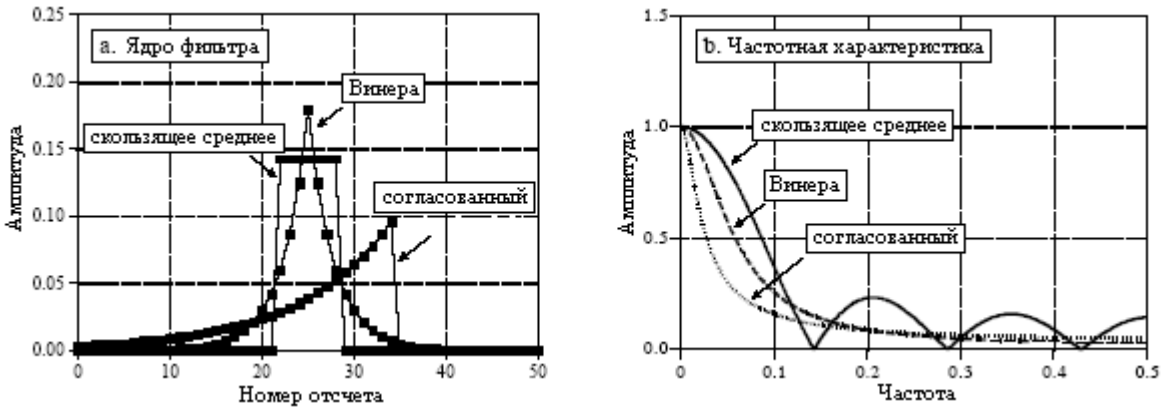


Рис. 17.8 Примеры оптимальных фильтров

Согласованный фильтр был обсужден ранее в Главе 7. Как показано на рис. 17.8a, ядро фильтра для согласованного фильтра такое же, как и обнаруживаемый сигнал

цели, за исключением того, что оно перевернуто слева направо. Идеей лежащей в основе согласованного фильтра является *корреляция*, и этот переворот необходим для того, чтобы выполнить *корреляцию*, используя *свертку*. Амплитуда каждой точки в выходном сигнале представляет собой меру того, насколько хорошо ядро фильтра *согласуется* с соответствующим участком входного сигнала. Вспомните, что выходной сигнал согласованного фильтра не обязательно выглядит как обнаруживаемый сигнал. Она действительно не имеет значения; ведь если используется согласованный фильтр, то форма сигнала цели должна уже быть известна. Согласованный фильтр является оптимальным в том смысле, что вершина пика гораздо выше шума, чем это может быть достигнуто любым другим линейным фильтром (см. рис. 17.9b).

Фильтр Винера (назван в честь Норберта Винера, после его теории оптимальной оценки) разделяет сигналы, опираясь на их частотные спектры. Как показано на рис. 17.7b, на некоторых частотах главным образом имеется сигнал, в то время как на других главным образом имеется шум. Кажется логичным, что частоты, содержащие "главным образом сигнал" нужно пропустить через фильтр, в то время как частоты, содержащие "главным образом шум" должны быть блокированы. Фильтр Винера продвигает эту идею на шаг вперед; усиление фильтра *на каждой из частот* определяется относительным количеством сигнала и шума *на данной частоте*:

$$H[f] = \frac{S^2[f]}{S^2[f] + N^2[f]}. \qquad (17.1)$$

Это соотношение используется для преобразования спектра на рис. 17.7b в частотную характеристику фильтра Винера, показанную на рис. 17.8b. Фильтр Винера оптимален в том смысле, что он максимизирует отношение мощности сигнала к мощности шума (по длине всего сигнала, а не в каждой отдельной точке). Соответствующее ядро фильтра находится из частотной характеристики фильтра Винера с использованием метода заказного фильтра.

Хотя идеи стоящие за этими оптимальными фильтрами математически изящны, на практике они часто терпят неудачу. Это не означает, что они никогда не должны использоваться. Суть состоит в том, чтобы не слышать слова "оптимальный" и не переставать думать. Давайте посмотрим на несколько причин, по которым Вы можете *не* захотеть их использовать.

Во-первых, не очень впечатляет различие между сигналами на рис. 17.9. Действительно, если бы Вам не сообщили заранее, то, глядя на сигналы, Вы, вероятно, не смогли бы сказать, какие параметры здесь оптимизированы. Это обычный случай для задач, содержащих перекрытие частотных спектров. Небольшая величина дополнительных преимуществ в работе, полученных от оптимального фильтра, возможно, не стоит увеличенной сложности программы, лишних усилий по проектированию и более длительного времени ее исполнения.

Во-вторых, фильтры Винера и согласованные фильтры полностью обусловливаются характеристиками задачи. Другие же фильтры, такие как фильтры с ограниченной окном синк функцией и скользящего среднего, могут быть скроены по вашему вкусу. Сторонники оптимального фильтра будут заявлять, что этот обман может только снизить эффективность фильтра. Это очень спорно. Запомните, каждый из этих фильтров является оптимальным лишь некоторым определенным образом (т.е. "в некотором смысле"). Достаточно редко требуется, чтобы задача была полностью оптимизирована, особенно, если результирующие сигналы интерпретируются человеком-наблюдателем. Например, инженер биомедицины может использовать фильтр Винера для максимизации отношения сигнал/шум в электрокардиограмме. Однако, совсем не

очевидно, что это также оптимизирует способность врача, глядя на сигнал, обнаружить неритмичную работу сердца.

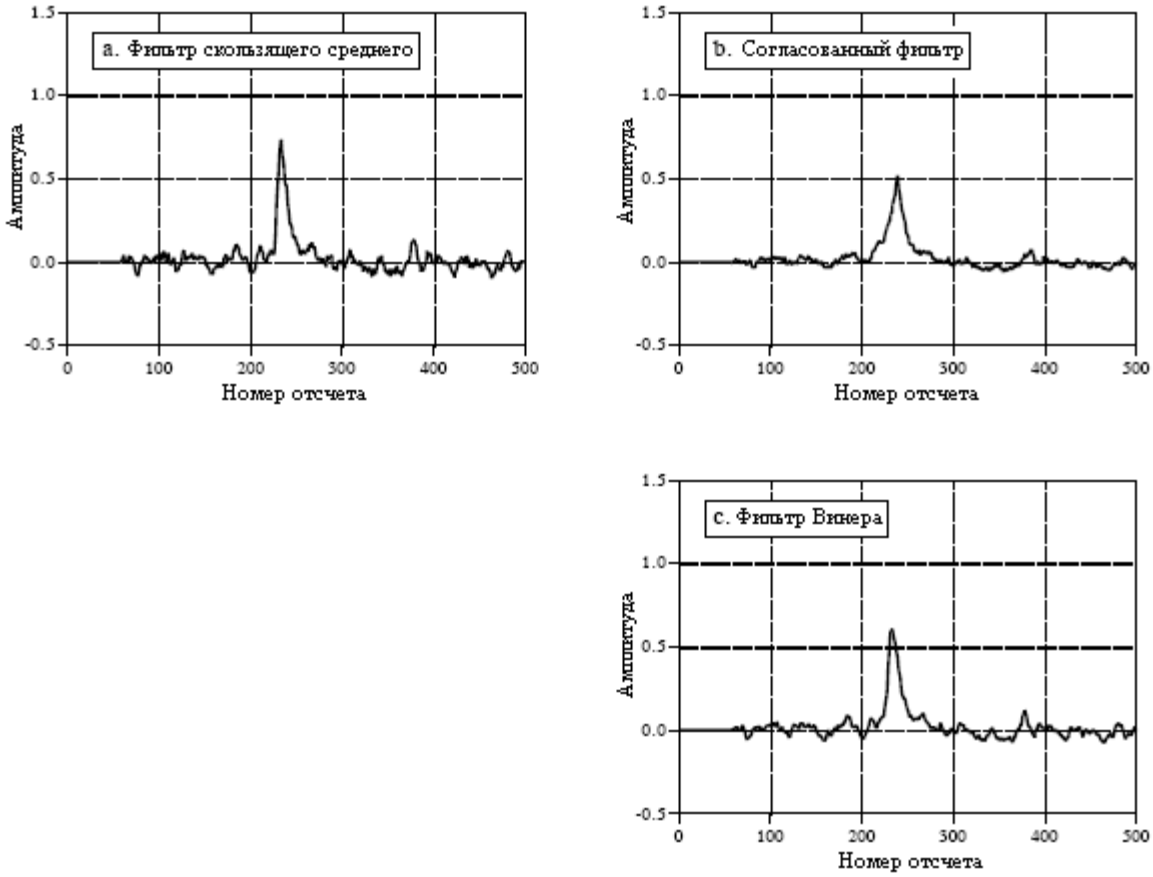


Рис. 17.9 Пример использования трех оптимальных фильтров

В-третьих, Фильтр Винера и согласованный фильтр должны быть реализованы с помощью *свертки*, что делает их чрезвычайно медленными при исполнении. Даже при улучшениях скорости, обсуждаемых в следующей главе (БПФ свертка), время вычисления может быть чрезмерно длительным. Для сравнения, *рекурсивные* фильтры (такие как скользящего среднего и другие, представленные в Главе 19) гораздо быстрее и могут обеспечить приемлемый уровень работы.

ГЛАВА
18

БПФ СВЕРТКА

Эта глава представляет два важных метода ЦОС: *метод суммирования перекрытий* и *БПФ свертку*. Метод суммирования перекрытий используется для разбиения длинного сигнала на более мелкие сегменты с целью облегчения его обработки. БПФ свертка использует метод суммирования перекрытий совместно с быстрым преобразованием Фурье, что позволяет осуществить свертку сигналов посредством перемножения их частотных спектров. Для ядер фильтра длиннее, чем приблизительно *64* точки, БПФ свертка быстрее, чем стандартная свертка, хотя дает точно такой же результат.

Метод суммирования перекрытий

Существует большое число приложений ЦОС, в которых длинный сигнал должен быть подвергнут фильтрации по *частям*. Например, высококачественное цифровое *воспроизведение звука* требует скорости передачи данных порядка *5* Мбайт/мин, а цифровое *воспроизведение изображения* требует около *500* Мбайт/мин. При таких высоких скоростях передачи данных компьютерам обычно не хватает памяти для одновременного хранения всего обрабатываемого сигнала. Существуют также системы, ведущие обработку данных "сегмент за сегментом", поскольку они работают в режиме *реального времени*. Например, телефонный сигнал не может быть задержан больше чем на несколько сотен миллисекунд, что ограничивает количество данных доступное для обработки в конкретный момент времени. *Обработка* может потребовать, сегментирования сигнала и во многих других приложениях. Примером такой обработки является БПФ свертка, основная тема этой главы.

Метод суммирования перекрытий основан на фундаментальной технике в ЦОС: (1) разложить сигнал на простые компоненты, (2) обработать каждую из компонент некоторым полезным образом и (3) объединить обработанные компоненты в окончательный сигнал. Пример того, как это осуществляется для метода суммирования перекрытий, показан на рис. 18.1. На рис. 18.1a представлен сигнал, который следует подвергнуть фильтрации, в то время как на рис. 18.1b показано ядро фильтра, которое предстоит использовать, ядро низкочастотного фильтра с ограниченной окном синк функцией. Переходя к нижней части рисунка, на рис. 18.1i показан отфильтрованный сигнал, сглаженная версия рис. 18.1a. Ключом к этому методу является то, как свертка воздействует на *длину* этих сигналов. Когда осуществляется свертка сигнала, состоящего из *N* отсчетов, с ядром фильтра, состоящим из *M* отсчетов, получается выходной сигнал длиной в *N+M-1* отсчетов. Например, входной сигнал (рис. 1.18a) содержит *300* отсчетов (изменяется от *0* до *299*), ядро фильтра (рис. 1.18b) содержит *101* отсчет (изменяется от *0* до *100*) и выходной сигнал (рис. 1.18i) содержит 400 отсчетов (изменяется от *0* до *399*).

Другими словами, когда осуществляется фильтрация сигнала, содержащего *N* отсчетов, он будет *расширен* на *M-1* точку *вправо*. (Это предполагает то, что ядро фильтра изменяется от индекса с номером *0* до индекса с номером *M*. Если в ядре фильтра используются отрицательные индексы, расширение будет происходить *влево*.) Для того

чтобы проиллюстрировать, где произойдет это расширение, на рис. 18.1a к сигналу между отсчетами *300* и *399* были добавлены нули. Пусть Вас не смущают маленькие значения по концам выходного сигнала (рис. 18.1i). Это просто результат того, что ядро фильтра с ограниченной окном синк функцией, имеет по своим концам маленькие значения. Все *400* отсчетов на рис. 18.1i отличны от нуля, даже притом, что некоторые из них слишком малы, чтобы это было видно на графике.

Разложение, используемое в методе суммирования перекрытий, показано на рис. 18.1c, рис. 18.1d и рис. 18.1e. Сигнал разбивается на сегменты, каждый из которых содержит *100* отсчетов исходного сигнала. Кроме того, справа от каждого сегмента добавлено *100* нулей. На следующем шаге при помощи свертки с ядром фильтра осуществляется индивидуальная фильтрация каждого сегмента. Это дает выходные сегменты, показанные на рис. 18.1f, рис. 18.1g и рис. 18.1h. Так как каждый входной сегмент *100* отсчетов в длину, а ядро фильтра *101* отсчет в длину, каждый выходной сегмент будет *200* отсчетов в длину. Важным моментом для понимания является то, что для того, чтобы допустить расширение во время свертки, к каждому входному сегменту было добавлено *100* нулей.

Обратите внимание на то, что расширение приводит к частичному *перекрытию* выходных сегментов друг с другом. Эти перекрывающиеся выходные сегменты складываются, давая выходной сигнал на рис. 18.1i. Например, отсчеты с *200* по *299* на рис. 18.1i найдены при помощи суммирования соответствующих отсчетов на рис. 18.1g и рис. 18.1h. Метод суммирования перекрытий дает точно такой же выходной сигнал, как и прямая свертка. Неудобством является намного большая сложность программы, обусловленная отслеживанием участков перекрывающихся отсчетов.

БПФ свертка

БПФ свертка использует принцип того, что *умножение* в частотной области соответствует *свертке* во временной области. При помощи ДПФ входной сигнал преобразуется в частотную область, перемножается с частотной характеристикой фильтра, а затем, с помощью обратного ДПФ преобразуется назад во временную область. Этот основной метод был известен со времен Фурье, однако по сути никто не обращал на него внимания. Это объясняется тем, что время необходимое для вычисления ДПФ было *больше*, чем время на непосредственное вычисление свертки. Все изменилось в 1965 году с появлением быстрого преобразования Фурье (БПФ). При использовании алгоритма БПФ для вычисления ДПФ, свертка через частотную область может быть *быстрее*, чем непосредственная свертка сигналов временной области. Конечный результат остается одним и тем же, изменяется лишь число вычислений за счет более эффективного алгоритма. По этой причине БПФ свертку также называют **высокоскоростной сверткой**.

БПФ свертка использует метод суммирования перекрытий, показанный на рис. 18.1; изменен только способ, которым входные сегменты преобразуются в выходные сегменты. На рис. 18.2 показан пример того, как при помощи БПФ свертки входной сегмент преобразуется в выходной сегмент. Для начала, используя БПФ, при помощи взятия ДПФ от ядра фильтра, находится частотная характеристика фильтра. Например, на рис. 18.2a показан пример ядра фильтра для полосно-пропускающего фильтра с ограниченной окном синк функцией. БПФ преобразует его в действительную и мнимую части частотной характеристики, показанные на рис. 18.2b и рис. 18.2c. Эти сигналы частотной области могут быть *не похожи* на полосно-пропускающий фильтр, поскольку они представлены в прямоугольной системе. Запомните, для понимания человеком частотной области полярная система обычно является самой лучшей, в то время как прямоугольная система обычно является самой лучшей для математических вычислений. Эти действительная и мнимая части сохраняются в компьютере для использования их при вычислении каждого из сегментов.

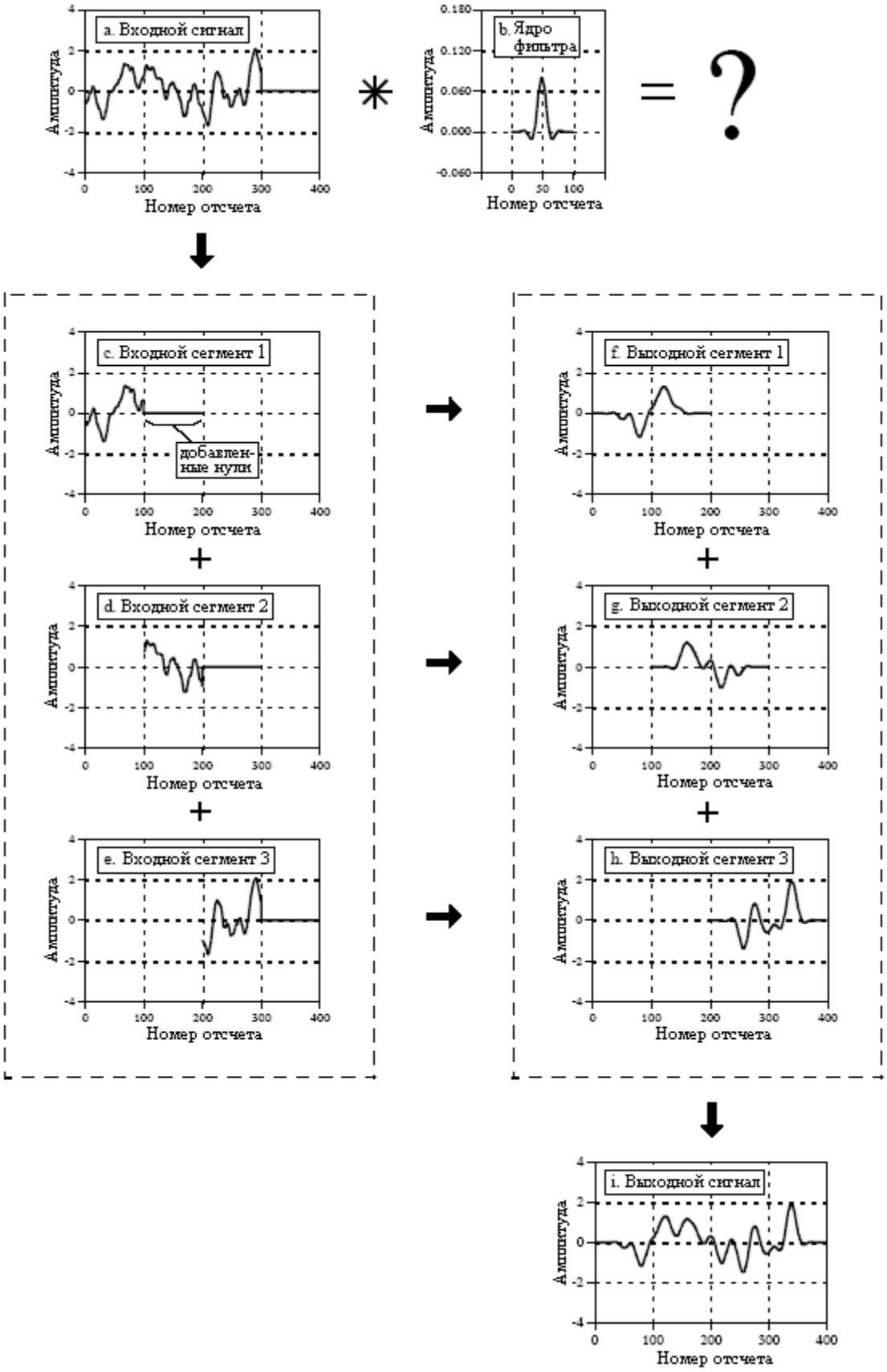


Рис. 18.1 Метод суммирования перекрытий

На рис. 18.2d показан обрабатываемый входной сегмент. Для нахождения его частотного спектра, показанного на рис. 18.2e и рис. 18.2f, используется БПФ. Затем, при

помощи перемножения частотной характеристики фильтра, представленной на рис. 18.2b и рис. 18.2c, со спектром входного сегмента, изображенным рис. 18.2e и рис. 18.2f, находится спектр выходного сегмента, показанный на рис. 18.2h и рис. 18.2i. Поскольку эти спектры состоят из действительной и мнимой частей, их перемножение осуществляется в соответствии с уравнением (9.1) Главы 9. Далее для нахождения выходного сегмента, показанного на рис. 18.2g, из его частотного спектра, приведенного на рис. 18.2h и рис. 18.2i, используется обратное БПФ. Важно понимать, что этот выходной сегмент точно такой же, как если бы он был получен путем непосредственной свертки входного сегмента на рис. 18.2d и ядра фильтра на рис. 18.2a.

Для того чтобы не возникало *круговой свертки* (также описывается в Главе 9), БПФ должно быть достаточно длинным. Это означает, что БПФ должно иметь такую же длину, как и выходной сегмент на рис. 18.2g. В частности, в примере на рис. 18.2 ядро фильтра содержит *129* точек, а каждый сегмент содержит *128* точек, что делает выходной сегмент *256* точек в длину. Это требует использования *256* точечного БПФ. Данное означает, что ядро фильтра на рис. 18.2a, для того чтобы довести его общую длину до *256* точек, должно быть дополнено *127* нулями. Аналогично, каждый из входных сегментов на рис. 18.2d должен быть дополнен *128* нулями. В качестве другого примера представьте, что Вам нужно осуществить свертку очень длинного сигнала с ядром фильтра содержащем *600* отсчетов. Одной из альтернатив была бы необходимость использовать сегменты из *425* точек и *1024* точечное БПФ. Другой альтернативой была бы необходимость использовать сегменты из *1449* точек и *2048* точечное БПФ.

В таблице 18.1 приведен пример программы, выполняющей БПФ свертку. Эта программа осуществляет фильтрацию сигнала, содержащего *10* миллионов точек посредством его свертки с *400* точечным ядром фильтра. Это осуществляется за счет разбиения входного сигнала на *16000* сегментов, каждый из которых содержит *625* точек. Когда производится свертка каждого из этих сегментов с ядром фильтра, получается выходной сегмент с *625+400-1=1024* точками. Таким образом, используется *1024* точечное БПФ. После определения и инициализации всех массивов (строки от *130* до *230*), первым шагом является вычисление и сохранение частотной характеристики фильтра (строки от *250* до *310*). Строка *260* вызывает мифическую подпрограмму, загружающую ядро фильтра с XX[0] по XX [399] и устанавливающую нулевое значение с XX[400] по XX[1023]. Подпрограмма в строке *270* представляет собой БПФ, преобразующее *1024* отсчета, содержащиеся в XX[ ], в *513* отсчетов, содержащихся в REX[ ] и IMX[ ], соответственно действительной и мнимой частях частотной характеристики. Эти значения передаются в массивы REFR[ ] и IMFR[ ] (для, соответственно действительной и мнимой частотной характеристики) для дальнейшего использования в программе.

Цикл FOR-NEXT в строках с *340* по *580* управляет обработкой *16000* сегментов. В строке 360 подпрограмма загружает с XX[0] по XX[624] следующий обрабатываемый сегмент и устанавливает нулевое значение с XX[625] по XX[1023]. В строке 370 для нахождения частотного спектра этих сегментов используется подпрограмма БПФ, действительная часть спектра помещается в *513* точек REX[ ], а мнимая помещается в *513* точек IMX[ ]. Строки с *390* по *430* показывают перемножение частотного спектра сегментов содержащегося в REX[ ] и IMX[ ] с частотной характеристикой фильтра, содержащейся в REFR[ ] и IMFR[ ]. Результат перемножения сохраняется в REX[ ] и IMX[ ], записываясь вместо данных хранимых там ранее. Поскольку теперь это является частотным спектром выходного сегмента, то для нахождения самого выходного сегмента может быть использовано обратное БПФ. Это выполняется в строке *450* мифической подпрограммой обратного БПФ, преобразующей *513* точек, содержащихся в REX[ ] и IMX[ ], в *1024* точки выходного сегмента, содержащиеся в XX[ ].

Строки от *470* до *550* обрабатывают перекрытия сегментов. Каждый выходной сегмент делится на две секции. Первые *625* точек (с *0* по *624*) должны быть объединены с перекрывающимися точками *предыдущего* выходного сегмента, а затем дописаны к

выходному сигналу. Последние 399 точек (с *625* по *1023*) должны быть сохранены таким образом, чтобы они могли быть наложены на *следующий* выходной сегмент.

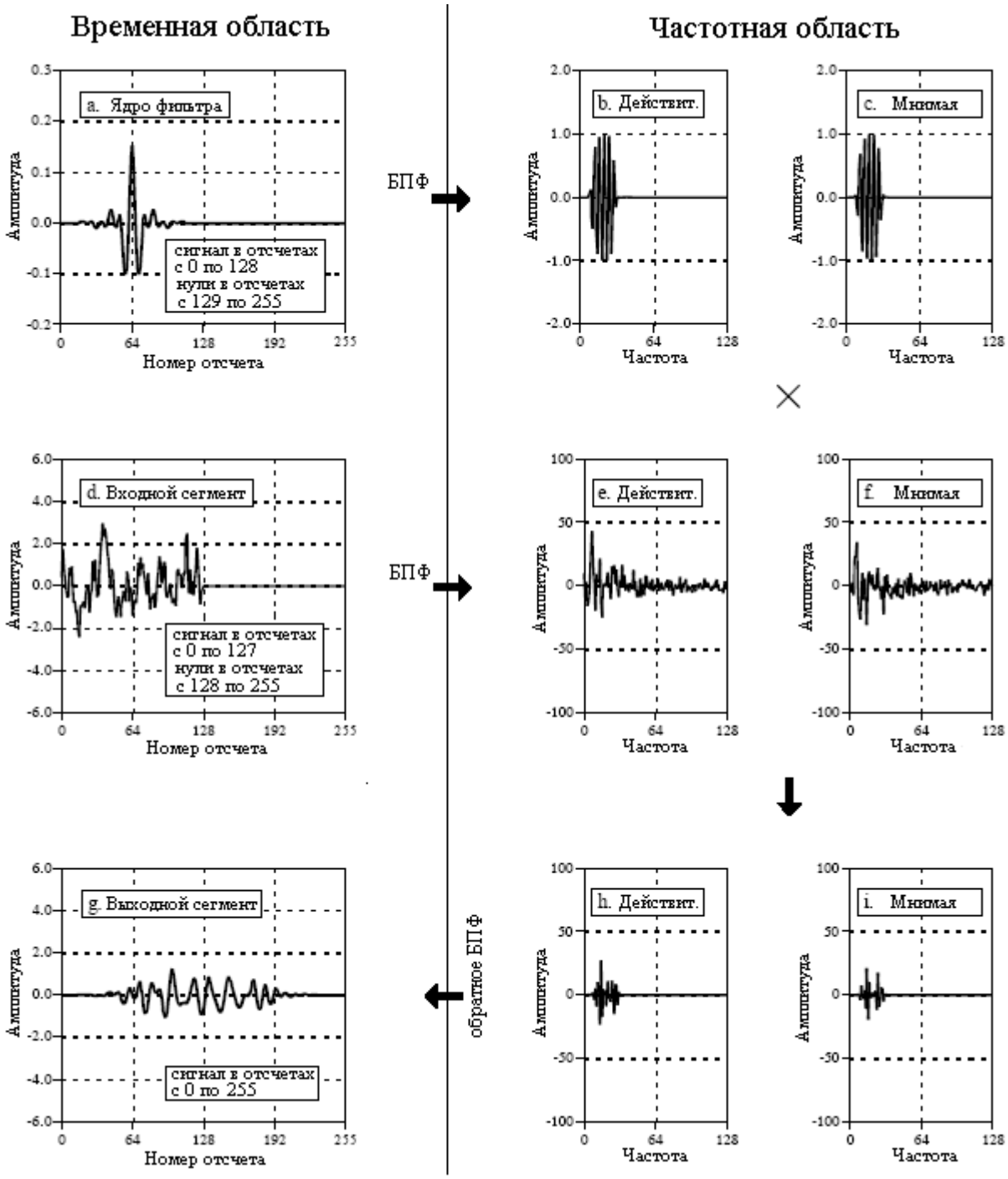


Рис. 18.2 БПФ свертка

Чтобы понять это, вернемся назад к рис. 18.1. Отсчеты с *100* по *199* на рис. 18.1g должны быть объединены с перекрывающимися точками *предыдущего* выходного сегмента на рис. 18.1f и могут быть тогда перемещены в выходной сигнал на рис. 18.1i. Для сравнения, отсчеты с *200* по *299* на рис. 18.1g должны быть сохранены таким образом, чтобы они могли быть объединены со следующим выходным сегментом на рис. 18.1h.

А теперь вернемся к программе. Массив OLAP[ ] используется для хранения *399* отсчетов одного сегмента, перекрывающихся со следующим сегментом. В строках с *470* по *490* к выходному, обрабатываемому в данный момент сегменту, добавляется *399*

значений из этого массива (из предыдущего выходного сегмента), результат сохраняется в XX[ ]. Затем, мифическая подпрограмма в строке *550* выводит *625* отсчетов с XX[0] по XX[624] в содержащий выходной сигнал файл. Далее, в строках с *510* по *530* в OLAP[ ] сохраняются *399* отсчетов текущего выходного сегмента, которые должны быть сохранены для следующего выходного сегмента.

Таблица 18.1.

```
100 'FFT CONVOLUTION
110 'This program convolves a 10 million point signal with a 400 point filter kernel. The input
120 'signal is broken into 16000 segments, each with 625 points. 1024 point FFTs are used.
130 '
130 '                                   'INITIALIZE THE ARRAYS
140 DIM XX[1023]                        'the time domain signal (for the FFT)
150 DIM REX[512]                        'real part of the frequency domain (for the FFT)
160 DIM IMX[512]                        'imaginary part of the frequency domain (for the FFT)
170 DIM REFR[512]                       'real part of the filter's frequency response
180 DIM IMFR[512]                       'imaginary part of the filter's frequency response
190 DIM OLAP[398]                       'holds the overlapping samples from segment to segment
200 '
210 FOR I% = 0 TO 398                   'zero the array holding the overlapping samples
220 OLAP[I%] = 0
230 NEXT I%
240 '
250 '                                 'FIND & STORE THE FILTER'S FREQUENCY RESPONSE
260 GOSUB XXXX                        'Mythical subroutine to load the filter kernel into XX[ ]
270 GOSUB XXXX                        'Mythical FFT subroutine: XX[ ] --> REX[ ] & IMX[ ]
280 FOR F% = 0 TO 512                 'Save the frequency response in REFR[ ] & IMFR[ ]
290 REFR[F%] = REX[F%]
300 IMFR[F%] = IMX[F%]
310 NEXT F%
320 '
330 '                                  'PROCESS EACH OF THE 16000 SEGMENTS
340 FOR SEGMENT% = 0 TO 15999
350 '
360 GOSUB XXXX                        'Mythical subroutine to load the next input segment into XX[ ]
370 GOSUB XXXX                        'Mythical FFT subroutine: XX[ ] --> REX[ ] & IMX[ ]
380 '
390 FOR F% = 0 TO 512                 'Multiply the frequency spectrum by the frequency response
400 TEMP = REX[F%]*REFR[F%] - IMX[F%]*IMFR[F%]
410 IMX[F%] = REX[F%]*IMFR[F%] + IMX[F%]*REFR[F%]
420 REX[F%] = TEMP
430 NEXT F%
440 '
450 GOSUB XXXX                         'Mythical IFFT subroutine: REX[ ] & IMX[ ] --> XX[ ]
460 '
470 FOR I% = 0 TO 398                 'Add the last segment's overlap to this segment
480 XX[I%] = XX[I%] + OLAP[I%]
490 NEXT I%
500 '
510 FOR I% = 625 TO 1023               'Save the samples that will overlap the next segment
520 OLAP[I%-625] = XX[I%]
```

```
530 NEXT I%
540 '
550 GOSUB XXXX                  'Mythical subroutine to output the 625 samples stored
560 '                           'in XX[0] to XX[624]
570 '
580 NEXT SEGMENT%
590 '
600 GOSUB XXXX                  'Mythical subroutine to output all 399 samples in OLAP[ ]
610 END
```

После того как будут обработаны все сегменты с *0* до *15999*, массив OLAP[ ] будет содержать *399* отсчетов сегмента *15999*, перекрывающихся с сегментом *16000*. Поскольку сегмента *16000* не существует (или его можно рассматривать как сегмент, содержащий все нули), в строке *600* эти 399 отсчетов записываются в выходной сигнал. Это делает выходной сигнал *16000×625+399=10000399* отсчетов в длину. Данное соответствует длине входного сигнала, плюс длина ядра фильтра, минус *1*.

Увеличение скорости

В каких случаях БПФ свертка быстрее, чем стандартная свертка? Как показано на рис. 18.3 (время исполнения приведено для процессора Pentium 100 Мгц, использующего плавающую запятую одинарной точности), ответ зависит от длины ядра фильтра. Время для стандартной свертки прямо пропорционально числу точек в ядре фильтра. Для сравнения, время, требуемое для БПФ свертки, растет очень медленно, пропорционально логарифму числа точек в ядре фильтра. Пересечение происходит тогда, когда ядро фильтра содержит приблизительно от *40* до *80* отсчетов (в зависимости от конкретных используемых аппаратных средств ЭВМ).

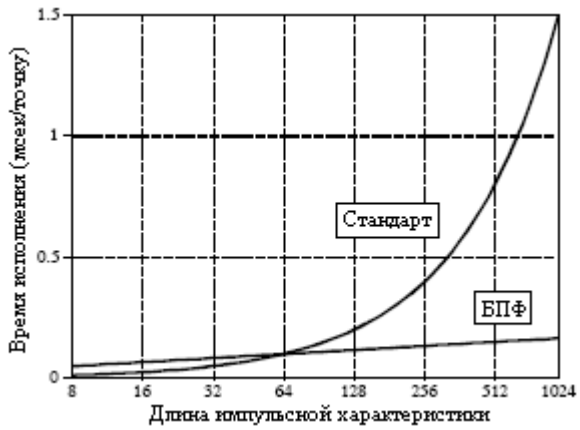


Рис. 18.3 Время исполнения для БПФ свертки

Главное, что следует запомнить: ядра фильтра короче, чем приблизительно 60 точек при помощи стандартной свертки могут быть реализованы быстрее, и время исполнения пропорционально длине ядра. Более длинные ядра фильтра с помощью БПФ свертки могут быть реализованы быстрее. С помощью БПФ свертки ядро фильтра может быть сделано настолько большим, насколько Вам нужно, при очень небольшой потере времени исполнения. Например, *16000* точечное ядро фильтра требует всего приблизительно *вдвое* большего времени исполнения, чем ядро всего с *64* точками.

Скорость свертки диктует также *точность* вычисления (как это описано для БПФ в Главе 12). Это связано с тем, что ошибка округления в выходном сигнале зависит от

общего числа вычислений, которое прямо пропорционально времени вычисления. Если выходной сигнал вычисляется *быстрее*, то он будет также вычисляться более *точно*. Например, представьте свертку сигнала с *1000* точечным ядром фильтра, с плавающей запятой одинарной точности. При использовании стандартной свертки, типичный шум округления, который может наблюдаться, будет приблизительно составлять *1* часть на *20000* (из справочного материала Главы 4). Для сравнения, БПФ свертка, как можно ожидать, будет на порядок по величине *быстрее* и на порядок по величине более *точной* (т.е. *1* часть на *200000*).

Держите припрятанной БПФ свертку для тех случаев, когда у Вас будет большое число данных для обработки и требуется чрезвычайно длинное ядро фильтра. Оперируйте терминами сигнала в *миллион* отсчетов и ядра фильтра в *тысячу* точек. Что-нибудь меньшее не будет оправдывать дополнительных усилий по программированию. Не хотите ли написать свою собственную программу БПФ свертки? Тогда загляните в библиотеку пакетов программ за уже написанными программами. Начните с вебсайта этой книги (см. страницу авторского права).

ГЛАВА
19

РЕКУРСИВНЫЕ ФИЛЬТРЫ

Рекурсивные фильтры – это эффективный способ получения длинной импульсной характеристики без необходимости выполнять длинную свертку. Они работают очень быстро, но обладают меньшим набором качеств и меньшей гибкостью, чем другие цифровые фильтры. Рекурсивные фильтры называют также фильтрами с *бесконечной импульсной характеристикой* (БИХ), поскольку их импульсные характеристики состоят из затухающих экспонент. Это отличает их от цифровых фильтров, реализованных посредством свертки и называемых фильтрами с *конечной импульсной характеристикой* (КИХ). Эта глава представляет собой введение в то, как работают рекурсивные фильтры и каким образом могут быть спроектированы простейшие члены их семейства. Главы 20, 26 и 33 представляют более утонченные методы проектирования.

Рекурсивный метод

Чтобы начать обсуждение рекурсивных фильтров, представьте, что Вам нужно извлечь информацию из некоторого сигнала *x*[ ]. Вам это настолько сильно необходимо, что Вы нанимаете старого профессора-математика для того, чтобы обработать для Вас данные. Задача профессора – отфильтровать *x*[ ] с целью получения *y*[ ], которое, как можно надеяться, содержит интересующую Вас информацию. Профессор начинает свою работу с вычисления каждой точки в *y*[ ] в соответствии с некоторым алгоритмом, который глубоко сидит в его сверх развитом мозге. На полпути к решению задачи происходит наиболее неприятная вещь. Профессор начинает бормотать об аналитических сингулярностях, дробных трансформантах и других демонах из кошмаров математика. Становится ясным, что профессор сошел с ума. И Вы с беспокойством наблюдаете, как профессор и Ваш алгоритм уводятся несколькими людьми в белых халатах.

В отчаянии Вы просматриваете записи профессора, чтобы найти алгоритм, которым он пользовался. Вы обнаруживаете, что он уже закончил вычисление точек с *y*[0] по *y*[27] и собирался начать с точки *y*[28]. Мы позволим переменной *n*, как показано на рис. 19.1, представлять вычисляемую в данный момент точку. Это означает, что *y*[*n*] является отсчетом *28* выходного сигнала, *y*[*n-1*] является отсчетом *27*, *y*[*n-2*] является отсчетом *26* и т.д. Подобным образом *x*[*n*] является точкой *28* входного сигнала, *x*[*n-1*] является точкой *27* и т.д. Для того чтобы понять используемый алгоритм, мы задаем себе вопрос: "Какая информация была доступна профессору для вычисления обрабатываемого в данный момент отсчета *y*[*n*]?"

Наиболее очевидным источником информации является *входной сигнал*, то есть значения: *x*[*n*], *x*[*n-1*], *x*[*n-2*], …. Профессор мог умножать каждую точку во входном сигнале на некоторый коэффициент и складывать произведения вместе:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \ldots.$$

Вы должны увидеть, что это ни что иное, как обыкновенная свертка с коэффициентами:

a_0, a_1, a_2, …, образующими ядро свертки. Если бы это было все, что делал профессор, то не было бы необходимости говорить об этой истории или об этой главе. Однако, есть еще один источник информации, к которому профессор имел доступ: ранее вычисленные значения выходного сигнала, содержащиеся в: $y[n-1]$, $y[n-2]$, $y[n-3]$, …. При использовании этой дополнительной информации, алгоритм был бы в виде:

$$\begin{aligned} y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \ldots \\ + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \ldots \quad . \end{aligned} \tag{19.1}$$

Говоря словами, каждая точка в выходном сигнале находится умножением значения входного сигнала на коэффициенты "*a*", умножением ранее рассчитанных значений выходного сигнала на коэффициенты "*b*", и суммированием результатов произведений вместе. Обратите внимание, что здесь отсутствует значение для b_0, поскольку оно соответствует отсчету, вычисляемому в данный момент. Уравнение (19.1) называется **рекурсивным уравнением**, а использующие его фильтры называются **рекурсивными фильтрами**. Определяющие фильтр значения "*a*" и "*b*" называются **рекурсивными коэффициентами**. В обычной практике может быть использована не более чем дюжина рекурсивных коэффициентов, в противном случае фильтр становится неустойчивым (т.е. выходной сигнал непрерывно увеличивается или в нем возникают непрерывные колебания). В таблице 19.1 показан пример программы рекурсивного фильтра.

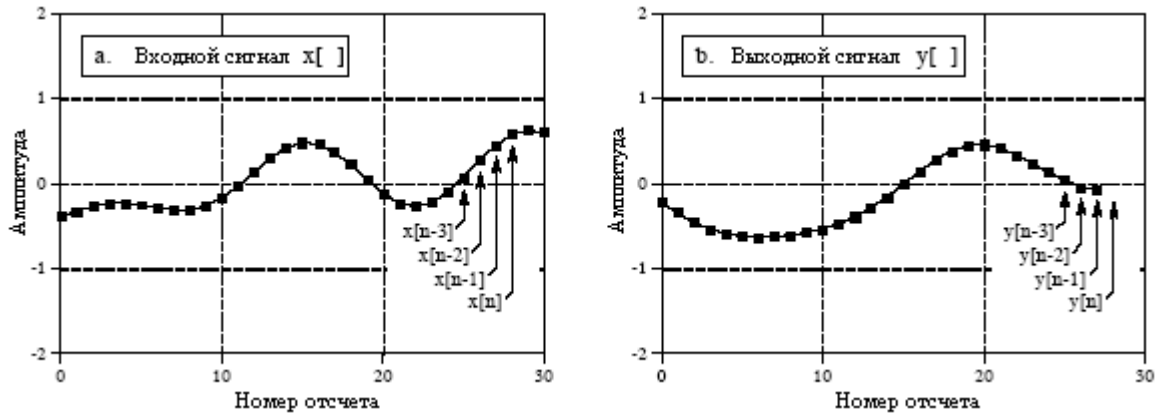


Рис. 19.1 Система обозначений рекурсивного фильтра

Рекурсивные фильтры полезны, потому что они позволяют *обойти* длинную свертку. Например, рассмотрим, что произойдет, когда через рекурсивный фильтр пройдет дельта функция. Выходной сигнал является импульсной характеристикой фильтра и, обычно, будет представлять собой затухающее по экспоненте синусоидальное колебание. Так как эта импульсная характеристика бесконечно длинна, рекурсивные фильтры часто называют фильтрами с *бесконечной импульсной характеристикой* (БИХ). Действительно, рекурсивные фильтры осуществляют свертку входного сигнала с очень длинным ядром фильтра, хотя используется всего несколько коэффициентов.

Соотношения между рекурсивными коэффициентами и откликом фильтра задаются математической техникой, называемой **z-преобразованием**, тема Главы 33. Например, z-преобразование может использоваться для таких задач, как взаимное преобразование между рекурсивными коэффициентами и частотной характеристикой, последовательное и параллельное соединение каскадов в единый фильтр, проектирование рекурсивных систем имитирующих аналоговые фильтры и т.д. К сожалению, z-

преобразование слишком математизировано и более сложно чем то, с чем желает иметь дело большинство *пользователей* ЦОС. Это - царство тех, кто специализируется на ЦОС.

Таблица 19.1.

```
100 'RECURSIVE FILTER
110 '
120 DIM X[499]                    'holds the input signal
130 DIM Y[499]                    'holds the filtered output signal
140 '
150 GOSUB XXXX                    'mythical subroutine to calculate the recursion
160 '                             'coefficients: A0, A1, A2, B1, B2
170 '
180 GOSUB XXXX                    'mythical subroutine to load X[ ] with the input data
190 '
200 FOR I% = 2 TO 499
210 Y[I%] = A0*X[I%] + A1*X[I%-1] + A2*X[I%-2] + B1*Y[I%-1] + B2*Y[I%-2]
220 NEXT I%
230 '
240 END
```

Существуют три способа нахождения рекурсивных коэффициентов, не требующих понимания z-преобразования. Во-первых, эта глава дает уравнения для проектирования нескольких простых типов рекурсивных фильтров. Во-вторых, Глава 20 дает "поваренную книгу" - компьютерную программу для проектирования более сложных низкочастотных и высокочастотных фильтров *Чебышева*. В-третьих, Глава 26 описывает итеративный метод для проектирования рекурсивных фильтров с *произвольной* частотной характеристикой.

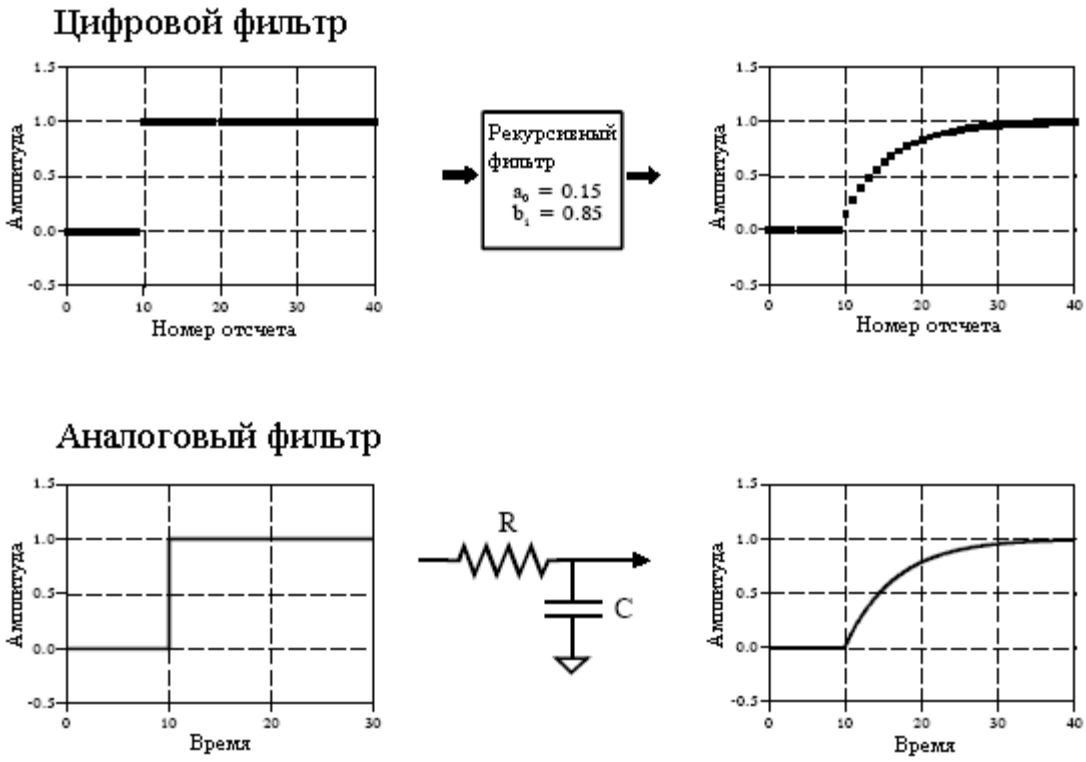


Рис. 19.2 Однополюсный низкочастотный фильтр

Однополюсный рекурсивный фильтр

На рис. 19.2 показан пример того, что называют **однополюсным** низкочастотным фильтром. Этот рекурсивный фильтр использует всего только два коэффициента $a_0=0,15$ и $b_1=0,85$. Для этого примера входным сигналом является ступенчатая функция. Как Вы и должны ожидать для фильтра низких частот, выходной сигнал плавно нарастает до установившегося уровня. На этом рисунке показано также что-то, что связано с Вашими познаниями в электронике. Этот низкочастотный рекурсивный фильтр полностью аналогичен электронному фильтру низких частот, состоящему из одного резистора и одного конденсатора.

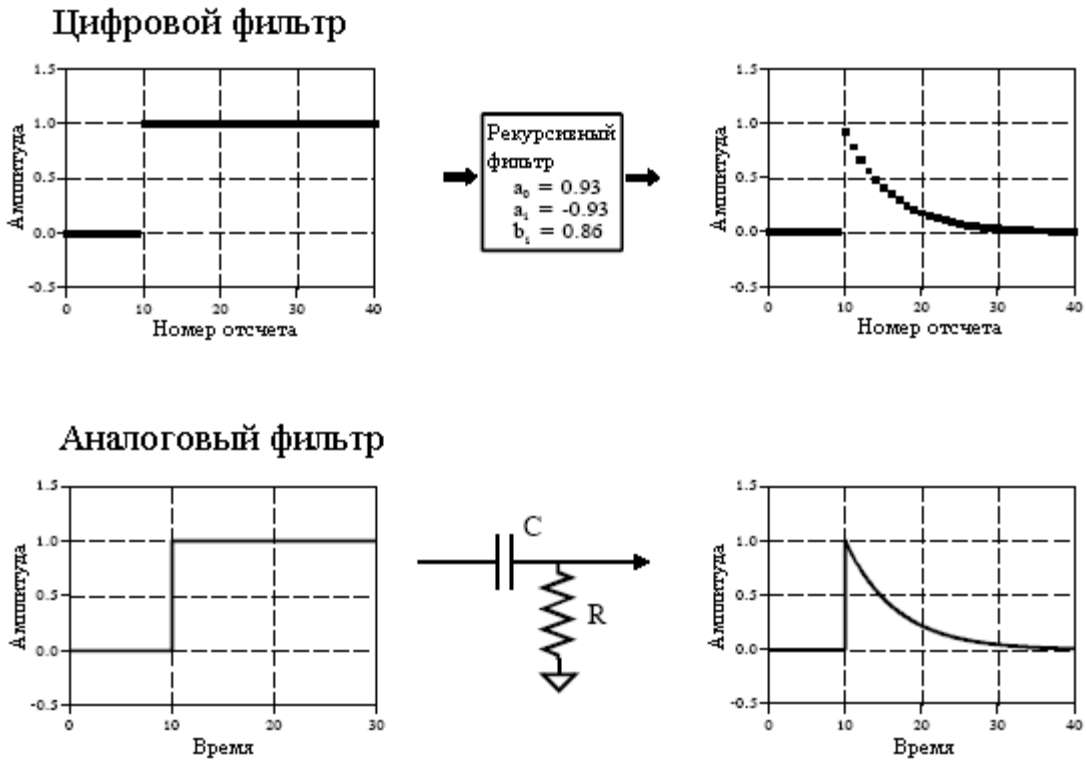


Рис. 19.3 Однополюсный высокочастотный фильтр

Красота рекурсивного метода заключается в его способности создавать широкое разнообразие откликов при изменении всего нескольких параметров. Например, на рис. 19.3 показан фильтр с тремя коэффициентами: $a_0=0,93$, $a_1=-0,93$ и $b_1=0,86$. Как видно из аналогичных переходных характеристик, этот цифровой фильтр имитирует электронный *R-C* фильтр верхних частот.

Такие однополюсные рекурсивные фильтры определенно являются чем-то, что Вам бы хотелось держать в своем инструментарии ЦОС. Вы можете использовать их для обработки цифровых сигналов точно так же, как Вы бы использовали *RC* цепочки для обработки сигналов аналоговой электроники. Сюда входит все, что Вы могли бы предположить: удаление постоянной составляющей, подавление высокочастотных шумов, формирование формы волны, сглаживание и т.д. Их легко программировать, они быстро исполняются и преподносят мало сюрпризов. Коэффициенты находятся из следующих простых уравнений:

для ФНЧ

$$\begin{aligned} a_0 &= 1 - x \\ b_1 &= x \end{aligned}, \qquad (19.2)$$

для ФВЧ

$$\begin{aligned} a_0 &= \frac{1+x}{2} \\ a_1 &= -\frac{1+x}{2}. \\ b_1 &= x \end{aligned} \qquad (19.3)$$

Характеристики этих фильтров регулируются параметром *x*, значение которого находится между нулем и единицей. Физически *x* представляет собой величину *затухания* между смежными отсчетами. Например, на рис. 19.3 *x* равен *0,86*, что означает, что величина каждого отсчета в выходном сигнале составляет *0,86* от величины предыдущего отсчета. Чем больше значение *x*, тем слабее затухание. Заметьте, что если *x* становится больше чем единица, фильтр становится *неустойчивым*. То есть любое ненулевое значение на входе заставит выходной сигнал увеличиваться до тех пор, пока не наступит переполнение.

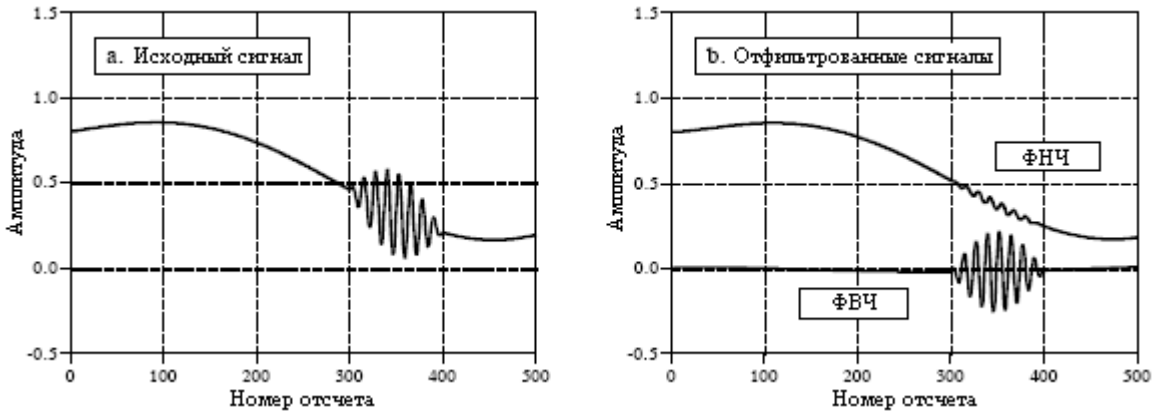


Рис. 19.4 Пример однополюсного рекурсивного фильтра

Значение для *x* может быть непосредственно задано, или найдено из желаемой *постоянной времени* фильтра. Точно так же, как *R*×*C* представляет собой количество секунд, требуемых *R*-*C* цепочке для того, чтобы достигнуть величины затухания *36,8%* от своего конечного значения, *d* представляет собой число отсчетов, необходимых рекурсивному фильтру для того, чтобы достигнуть той же самой величины затухания:

$$x = e^{-1/d}. \qquad (19.4)$$

Например, затухание от отсчета к отсчету порядка *x=0,86* соответствует постоянной времени *d=6,63* отсчета (как показано на рис. 19.3). Существует также фиксированное соотношение между *x* и *частотой среза* f_C цифрового фильтра на уровне *-3dB*:

$$x = e^{-2\pi f_c}. \qquad (19.5)$$

Это дает три способа для нахождения коэффициентов "*a*" и "*b*" - начинать от постоянной времени, частоты среза или просто от непосредственного задания *x*.

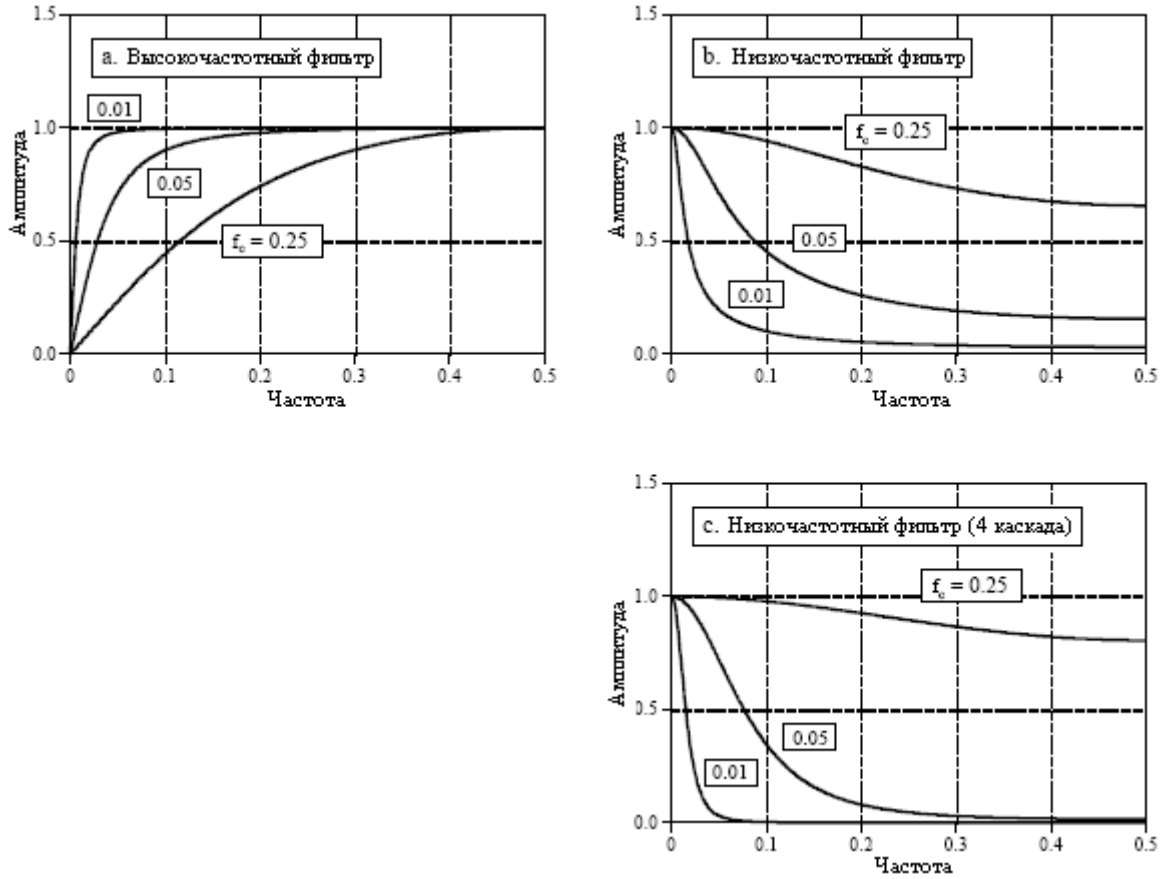


Рис. 19.5 Однополюсные частотные характеристики

На рис. 19.4 показан пример использования однополюсного рекурсивного фильтра. На рис. 19.4a исходный сигнал представляет собой плавную кривую за исключением вспышки высокочастотной синусоидальной волны. На рис. 19.4b показан сигнал после прохождения его через низкочастотный и высокочастотный фильтры. Сигналы были отделены довольно хорошо, но не идеально, так же, как если бы для аналогового сигнала использовались простые *R-C* цепи.

На рис. 19.5 показаны частотные характеристики различных однополюсных рекурсивных фильтров. Эти кривые получены при помощи пропускания через фильтр дельта функции с целью нахождения импульсной характеристики фильтра. После чего, для преобразования импульсной характеристики в частотную характеристику, использовалось БПФ. В принципе, импульсная характеристика бесконечно длинна, однако, через приблизительно от *15* до *20* постоянных времени она затухает ниже шума округления одинарной точности. Например, когда постоянная времени фильтра составляет *d=6,63* отсчетов, импульсная характеристика может содержать около *128* отсчетов.

Ключевой характеристикой на рис. 19.5 является то, что однополюсные рекурсивные фильтры обладают незначительный способностью отделять одну полосу частот от другой. Другими словами они хорошо работают во временной области и плохо в частотной области. Частотная характеристика может быть слегка улучшена с помощью последовательного соединения каскадов. Это может быть сделано двумя способами. Во-первых, сигнал можно пропустить через фильтр несколько раз. Во-вторых, можно

использовать z-преобразование для нахождения рекурсивных коэффициентов, объединяющих последовательно соединенные каскады в одну ступень. Оба способа работоспособны и часто используются. На рис. 19.5c показана частотная характеристика четырех соединенных последовательно низкочастотных фильтров. Хотя подавление в полосе заграждения значительно улучшается, завал все еще остается ужасным. Если Вам необходимы более лучшие характеристики в частотной области, взгляните на фильтры Чебышева в следующей главе.

Четырех каскадный фильтр низкой частоты сопоставим с фильтром Блэкмена и Гауссовым фильтром (родственниками скользящего среднего, Глава 15), но с намного более высокой скоростью исполнения. Уравнения проектирования для четырех каскадного фильтра нижних частот следующие (соотношение между x и частотой среза для этого фильтра задается уравнением (19.5), где 2π заменяется на $14{,}445$):

$$\begin{aligned} a_0 &= (1-x)^4 \\ b_1 &= 4x \\ b_2 &= -6x^2 \\ b_3 &= 4x^3 \\ b_4 &= -x^4 \end{aligned} \quad . \qquad (19.6)$$

Узкополосные фильтры

Типичной потребностью в электронике и ЦОС является изолирование узкой полосы частот от более широкого полосового сигнала. Например, Вы можете пожелать изолировать тональные сигналы в телефонной сети или устранить в измерительной системе помеху 60 герц. Доступными здесь являются два типа частотных характеристик: *полосно-пропускающая* и *полосно-заграждающая* (также называемая **фильтром пробкой**). На рис. 19.6 показана частотная характеристика этих фильтров, рекурсивные коэффициенты которых задаются соответственно следующими уравнениями:

$$\begin{aligned} a_0 &= 1-K \\ a_1 &= 2(K-R)\cos(2\pi f) \\ a_2 &= R^2-K \\ b_1 &= 2R\cos(2\pi f) \\ b_2 &= -R^2 \end{aligned} \quad . \qquad (19.7)$$

$$
\begin{aligned}
a_0 &= K \\
a_1 &= -2K\cos(2\pi f) \\
a_2 &= K \\
b_1 &= 2R\cos(2\pi f) \\
b_2 &= -R^2
\end{aligned}
\qquad (19.8)
$$

где

$$K = \frac{1 - 2R\cos(2\pi f) + R^2}{2 - 2\cos(2\pi f)};$$

$$R = 1 - 3BW\,.$$

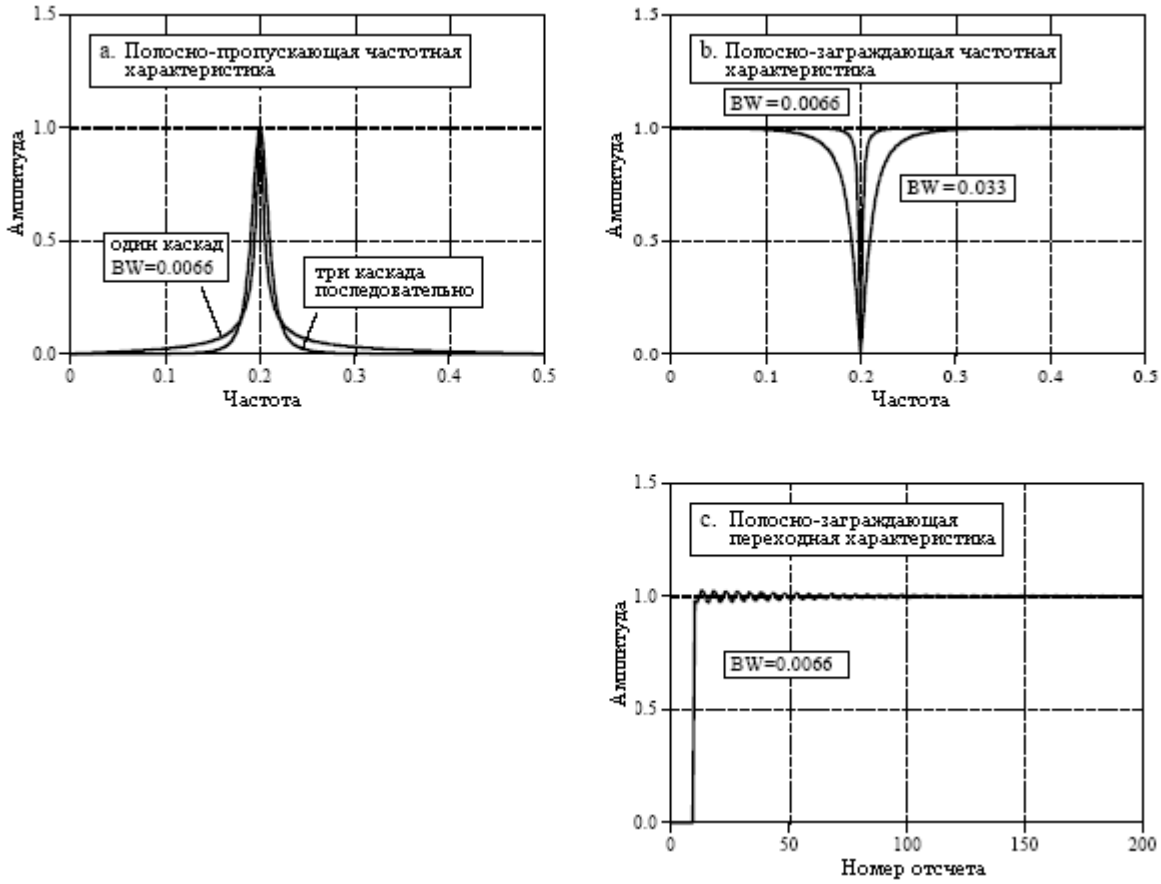


Рис. 19.6 Характеристики узкополосных фильтров

До того как использовать эти уравнения должны быть выбраны два параметра: центральная частота *f* и ширина полосы *BW* (измеренная на уровне амплитуды *0,707*). Оба они выражаются в долях от частоты дискретизации и, следовательно, должны находиться между *0* и *0,5*. Из этих двух указанных величин вычисляют промежуточные переменные: *R* и *K*, а затем рекурсивные коэффициенты.

Как показано на рис. 19.6a полосно-пропускающий фильтр имеет относительно большие, тянущиеся от основного пика, *хвосты*. Эту ситуацию можно улучшить последовательным соединением нескольких каскадов. Поскольку уравнения проектирования становятся тогда весьма длинными то, вместо того чтобы пытаться найти коэффициенты необходимые для отдельного фильтра, проще реализовать такое последовательное соединение, выполнив фильтрацию сигнала несколько раз.

На рис. 19.6b показан пример полосно-заграждающего фильтра. Самая узкая ширина полосы, которую можно получить при использовании одинарной точности, составляет около *0,0003* от частоты дискретизации. При выходе за это ограничение ослабление пробки будет уменьшаться. На рис. 19.6c показана переходная характеристика полосно-заграждающего фильтра. Здесь заметно перерегулирование и звон, однако его амплитуда очень мала. Это дает возможность фильтру устранять узкополосные помехи (*60* Гц и т.п.) лишь с незначительным искажением формы волны временной области.

Фазовая характеристика

Существует три типа *фазовых характеристик*, которые может иметь фильтр: **нулевая фаза**, **линейная фаза** и **нелинейная фаза**. Пример каждой из них показан на рис. 19.7. Как показано на рис. 19.7a, фильтр *нулевой фазы* характеризуется симметричной относительно нулевого отсчета импульсной характеристикой. Фактическая форма кривой значения не имеет, однако отсчеты, имеющие отрицательную нумерацию являются зеркальным изображением отсчетов имеющих положительную нумерацию. Когда от такой симметричной формы волны берется преобразование Фурье, как показано на рис. 19.7b, фаза везде будет нулевая.

Недостатком фильтра нулевой фазы является то, что он требует использования отрицательных индексов, которые могут быть неудобными в работе. Способом обойти это является фильтр линейной фазы. Импульсная характеристика на рис. 19.7d идентична той, которая показана на рис. 19.7a за исключением того, что она сдвинута таким образом, чтобы использовались только положительные номера отсчетов. Импульсная характеристика сохраняет симметрию слева направо, однако положение оси симметрии относительно нуля смещено. В результате это смещение приводит к фазе на рис. 19.7e в виде *прямой линии*, отсюда и название: *линейная фаза*. Наклон этой прямой линии прямо пропорционален величине сдвига. Поскольку сдвиг импульсной характеристики не вносит ничего, кроме идентичного сдвига выходного сигнала, для большинства приложений фильтр линейной фазы эквивалентен фильтру нулевой фазы.

На рис. 19.7g показана импульсная характеристика, *не* являющаяся симметричной слева направо. Соответственно фаза на рис. 19.7h *не* является прямой линией. Другими словами, здесь - *нелинейная фаза*. Не путайте термины: *нелинейная* и *линейная фаза* с понятием *линейная система*, описанным в Главе 5. Хотя и там, и там используется слово *линейная*, они не связаны друг с другом.

Почему следует обращать внимание на то, является ли фаза линейной или нет? Ответ показан на рис. 19.7c, рис. 19.7f и рис. 19.7i. Они показывают **отклик на прямоугольный импульс** каждого из этих трех фильтров. Отклик на прямоугольный импульс представляет собой ничто иное, как переходную характеристику в положительном направлении, сопровождаемую переходной характеристикой в отрицательном направлении. Отклик на прямоугольный импульс используется здесь для того, чтобы показать, что происходит в обоих случаях, в случае наличия нарастающего и падающего фронтов в сигнале. А вот и важная часть: у фильтров нулевой и линейной фазы левый и правый фронты выглядят *одинаково*, в то время, как у фильтров нелинейной фазы левый и правый фронты выглядят *по-разному*. В большом числе приложений нельзя допустить того, чтобы левые и правые фронты выглядели по-разному. Одним из таких примеров является экран осциллографа, где такое различие может быть интерпретировано

неправильно, как характерная особенность измеряемого сигнала. Другим примером является обработка изображения. Можете ли Вы себе представить, что, включая свой телевизор, Вы обнаружите, что левое ухо вашего любимого актера, выглядит не так, как его правое ухо?

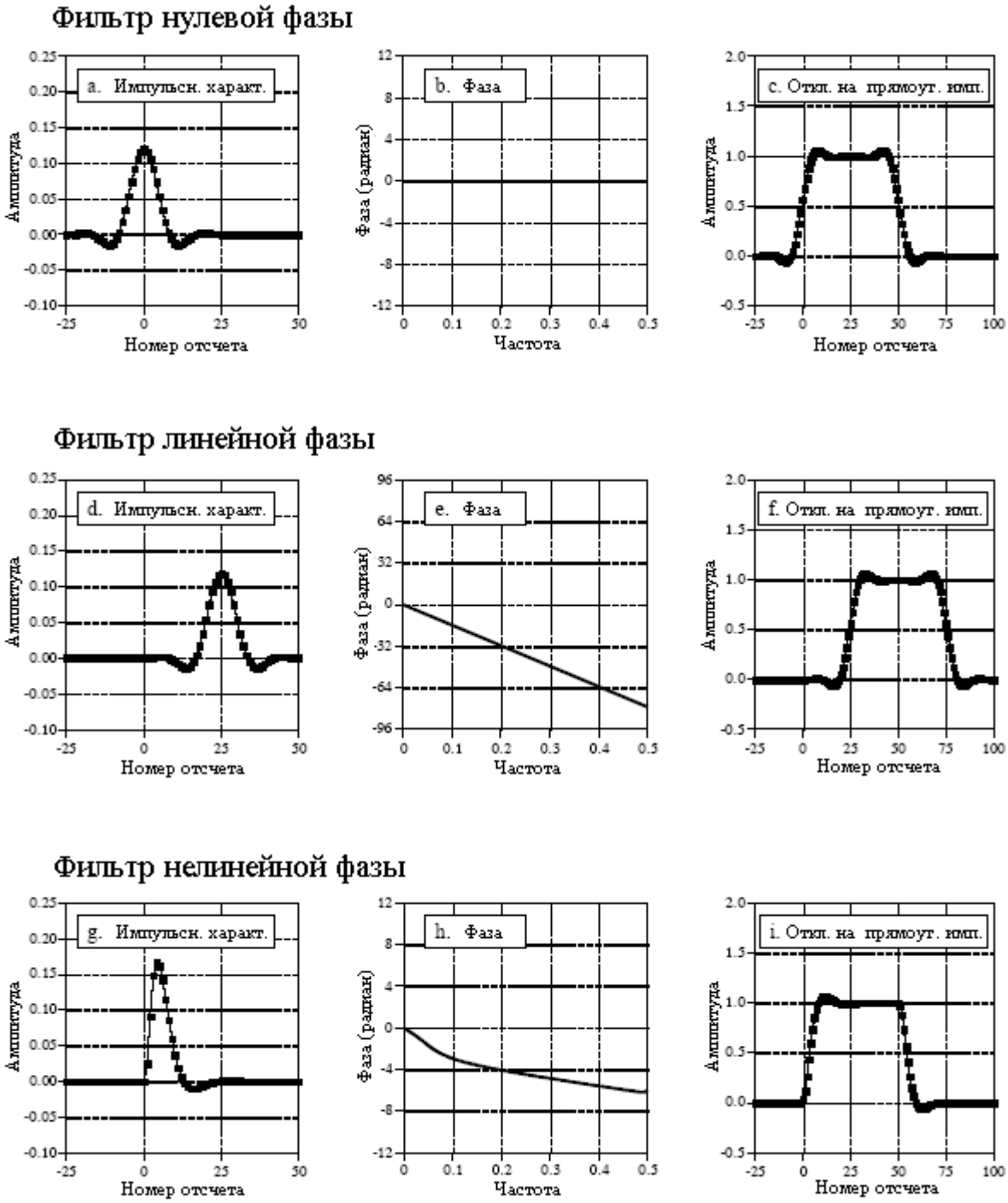


Рис. 19.7 Фильтры нулевой, линейной и нелинейной фазы

Сделать КИХ (конечная импульсная характеристика) фильтр, обладающий линейной фазой - легко. Это в силу того что, что импульсная характеристика (ядро фильтра) непосредственно *задается* в процессе проектирования. Все, что требуется, так это сделать ядро фильтра симметричным слева направо. В случае с БИХ (рекурсивными) фильтрами дело обстоит не так, поскольку то, что задано, это рекурсивные коэффициенты, а не импульсная характеристика. Импульсная характеристика

рекурсивного фильтра *не* обладает симметрией слева направо, а, следовательно, имеет *нелинейную* фазу.

Аналоговые электронные схемы с фазовой характеристикой имеют те же самые проблемы. Представьте стоящую на вашем столе схему, составленную из резисторов и конденсаторов. Если на вход всегда подан ноль, на выходе также всегда будет ноль. Когда на вход подается дельта импульс, конденсаторы быстро заряжаются до некоторой величины, а затем по экспоненте разряжаются через резисторы. Импульсная характеристика (т.е. выходной сигнал) представляет собой комбинацию этих по-разному затухающих экспонент. Импульсная характеристика *не может* быть симметричной, поскольку до прихода дельта импульса на выходе был ноль, а экспоненциальное затухание никогда снова не достигнет нулевого уровня. Разработчики аналоговых фильтров штурмуют эту проблему при помощи **фильтров Бесселя**, представленных в Главе 3. Фильтры Бесселя спроектированы таким образом, чтобы иметь, насколько это возможно, линейную фазу, однако, в работе они намного отстают от цифровых фильтров. Способность обеспечивать *точную* линейную фазу является явным преимуществом цифровых фильтров.

К счастью существует простой способ модификации рекурсивных фильтров для получения *нулевой фазы*. Пример того, как все это работает, показан на рис. 19.8. Входной сигнал, фильтрацию которого следует осуществить, показан на рис. 19.8a. На рис. 19.8b показан сигнал после того, как он был отфильтрован с помощью однополюсного низкочастотного фильтра. Поскольку это фильтр нелинейной фазы, левый и правый фронты выглядят неодинаково, они являются перевернутыми версиями друг друга. Как описывалось ранее, этот рекурсивный фильтр реализуется посредством вычисления каждого отсчета, начиная с отсчета *0* и продолжая обработку по отсчет *150*.

Теперь, предположим, что вместо того, чтобы двигаться от отсчета *0* к отсчету *150*, мы начнем с отсчета *150* и будем двигаться к отсчету *0*, другими словами, каждый отсчет в выходном сигнале вычисляется из входных и выходных отсчетов находящихся *справа* от вычисляемого в данный момент отсчета. Это означает, что рекурсивное уравнение (19.1) изменяется до:

$$\begin{aligned} y[n] = a_0 x[n] + a_1 x[n+1] + a_2 x[n+2] + a_3 x[n+3] + \ldots \\ + b_1 y[n+1] + b_2 y[n+2] + b_3 y[n+3] + \ldots \quad . \end{aligned} \tag{19.9}$$

На рис. 19.8c показан результат такой **обратной фильтрации**. Это аналогично пропусканию аналогового сигнала через электронную *R-C* цепь при условии, что время течет *вспять*. !йоннелесв юиневонзечси и юитажс к итсевирп тежом ьтяпсв инемерв еинечет – еинежеретсодерП

Фильтрация в обратном направлении сама по себе не дает никакой выгоды; отфильтрованный сигнал все еще обладает неодинаковыми левым и правым фронтами. Чудо происходит тогда, когда прямая и обратная фильтрации *объединяются*. Рис. 19.8d является результатом фильтрации сигнала в прямом направлении, а затем повторной фильтрации в обратном направлении. Вуаля! Это дает рекурсивный фильтр *нулевой фазы*. Действительно, любой рекурсивный фильтр может быть преобразован к нулевой фазе при помощи такой техники **двунаправленной фильтрации**. Единственным пенальти за такое улучшение характеристик является коэффициент два во времени исполнения и сложности программы.

А как Вам найти импульсную и частотную характеристики всего фильтра? Модули частотной характеристики для каждого направления одинаковы, тогда как фазы противоположны по знаку. Когда эти два направления объединяются, модуль становится возведенным *в квадрат*, в то время как фаза гасится до *нуля*. Во временной области это соответствует свертке исходной импульсной характеристики с ее собственной версией

перевернутой слева направо. Например, импульсная характеристика однополюсного низкочастотного фильтра представляет собой одностороннюю экспоненту. Импульсная характеристика соответствующего двунаправленного фильтра представляет собой свертку односторонней экспоненты, затухающей вправо с односторонней экспонентой, затухающей влево. После математических преобразований это оказывается двусторонней экспонентой, затухающей в обе стороны влево и вправо с тем же самым коэффициентом затухания, что и у исходного фильтра.

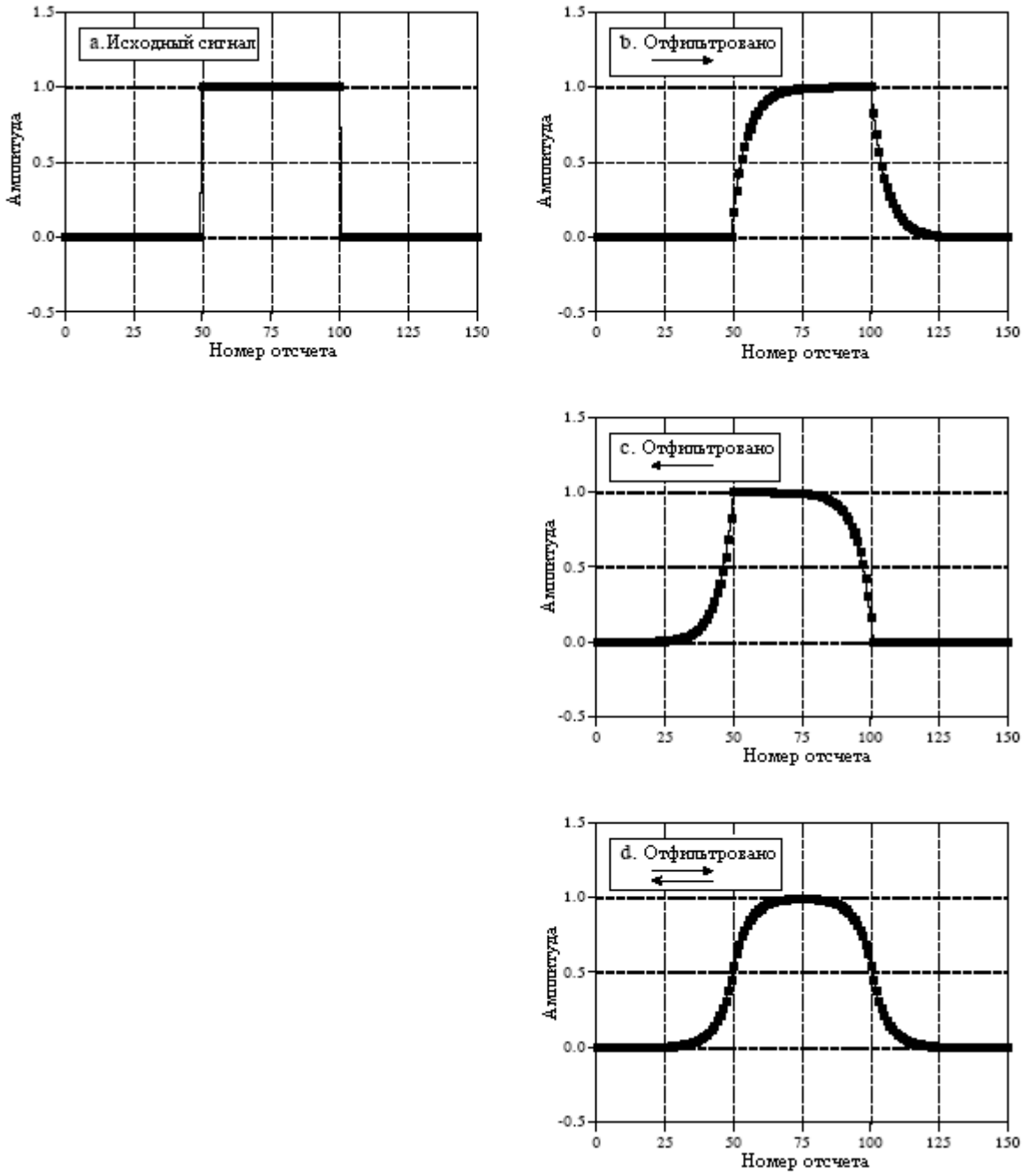


Рис. 19.8 Двунаправленный рекурсивный фильтр

В некоторых приложениях, подобных системам, осуществляющим на непрерывной основе попеременный ввод и вывод данных, в конкретный момент времени в компьютере хранится только часть сигнала. В этих случаях двунаправленная фильтрация может быть использована при объединении ее с методом суммирования перекрытий, описанным в предыдущей главе. Когда Вы подходите к вопросу о том, насколько длинна импульсная

характеристика, не говорите "бесконечна". Если Вы делаете так, то Вам понадобится дополнить каждый сегмент сигнала *бесконечным* количеством нулей. Запомните, импульсная характеристика может быть усечена в тот момент, когда ее затухание становится ниже шума округления, т.е. приблизительно в районе от 15 до 20 постоянных времени. Для обеспечения расширения во время двунаправленной фильтрации, каждый сегмент должен быть дополнен нулями как слева, так и справа.

Использование целых чисел

Плавающая запятая одинарной точности является идеальной для применения в таких простых рекурсивных фильтрах. Возможно и использование целых, но это гораздо труднее. Здесь две основные проблемы. Во-первых, ошибка округления от ограниченного числа бит может ухудшить характеристику фильтра или даже сделать его неустойчивым. Во-вторых, дробные значения рекурсивных коэффициентов должны обрабатываться математикой целых чисел. Один из способов разрешения этой проблемы состоит в том, чтобы представить каждый коэффициент в виде дроби. Например, *0,15* становится *19/128*. Вместо умножения на *0,15*, Вы сначала умножаете на *19*, а затем делите на *128*. Другим способом является замена умножений на справочные таблицы. Например, *12* битовый АЦП выдает отсчеты, значения которых лежат в пределах от *0* до *4095*. Вместо умножения каждого отсчета на *0,15*, Вы пропускаете отсчеты через справочную таблицу длиной в *4096* значений. Значение, полученное из справочной таблицы, равно значению входной величины умноженной на *0,15*. Этот метод является очень быстрым, но он требует дополнительной памяти; для каждого коэффициента нужна отдельная справочная таблица. Прежде, чем Вы попробуете какой-нибудь из этих методов целых чисел, убедитесь, что рекурсивный алгоритм для фильтра скользящего среднего не будет удовлетворять вашим потребностям. Ведь он *любит* целые числа.

ГЛАВА
20

ФИЛЬТРЫ ЧЕБЫШЕВА

Фильтры Чебышева используются для отделения одной полосы частот от другой. Хотя они и не могут соответствовать характеристикам фильтров с ограниченной окном синк функцией, для многих приложений они более чем адекватны. Первое неотъемлемое свойство фильтров Чебышева – это их скорость, обычно по величине более чем на порядок выше, чем у ограниченной окном синк функции. Это связано с тем, что чаще всего они реализуются с помощью *рекурсии*, а не с помощью *свертки*. Проектирование этих фильтров основано на математической технике называемой *z-преобразованием*, обсуждаемой в Главе 33. Эта глава представляет информацию, необходимую для *использования* фильтров Чебышева, без необходимости пробираться через трясину современной математики.

Характеристики фильтров Чебышева и Баттерворта

Характеристика Чебышева является математической стратегией для достижения более быстрого *завала* при допущении в частотной характеристике *пульсаций*. Аналоговые и цифровые фильтры, использующие этот подход, называются *фильтрами Чебышева*. Например, в Главе 3 аналоговые фильтры Чебышева использовались для аналого-цифрового и цифро-аналогового преобразования. Эти фильтры называются так потому, что они используют *полиномы Чебышева*, развитые великим русским математиком Пафнутием Львовичем Чебышевым (1821-1894). Эта фамилия была переведена с русского языка и появилась в литературе с различным написанием, таким как: Chebychev, Tschebyscheff, Tchebysheff и Tchebichef (в некоторых старых отечественных изданиях встречается написание Чебышов – прим. перев.).

На рис. 20.1 показана частотная характеристика низкочастотного фильтра Чебышева с пульсацией в полосе пропускания равной: *0%*, *0,5%* и *20%*. По мере увеличения пульсации (плохо), завал становится круче (хорошо). Характеристика Чебышева представляет собой оптимальный компромисс между этими двумя параметрами. Когда пульсация установлена на *0%*, фильтр называется **максимально пологим** или **фильтром Баттерворта** (по имени британского инженера С. Баттерворта, который описал эту характеристику в 1930). Для цифровых фильтров пульсации в *0,5%* часто являются хорошим выбором. Это соответствует типичной точности и достоверности аналоговой электроники, через которую прошел сигнал.

Фильтры Чебышева, обсуждаемые в этой главе, называются фильтрами **типа 1**, что означает, что пульсации допускаются только в *полосе пропускания*. Для сравнения, фильтры Чебышева **типа 2** имеют пульсации только в *полосе заграждения*. Фильтры типа 2 используются редко, и мы не будем их обсуждать. Существует, однако, важная разработка называемая **эллиптическим фильтром**, у которого пульсации есть в *обеих* и полосе пропускания, и полосе заграждения. Для заданного числа полюсов эллиптические фильтры обеспечивают наиболее быстрый завал, но их проектирование гораздо труднее. Здесь мы не будем обсуждать эллиптический фильтр, но Вы должны знать, что часто он является первым выбором профессиональных проектировщиков фильтров, как в аналоговой электронике, так и в ЦОС. Если Вы хотите достичь такого уровня характерис-

тик, купите пакет программ по проектированию цифровых фильтров.

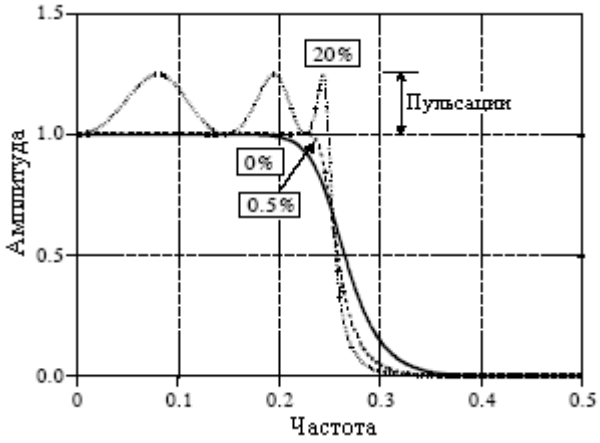


Рис. 20.1 Частотная характеристика фильтра Чебышева

Проектирование фильтра

Для проектирования фильтра Чебышева Вы должны выбрать четыре параметра: (1) низкочастотную или высокочастотную характеристику, (2) частоту среза, (3) процент пульсаций в полосе пропускания и (4) число полюсов. Только вот, что такое *полюс*? Вот два ответа. Если Вам не понравится один, может быть будет полезным другой:

Ответ 1 – преобразование Лапласа и z-преобразование являются математическими способами разложения импульсной характеристики на синусоиды и затухающие экспоненты. Это делается путем выражения характеристик системы в виде одного комплексного полинома, деленного на другой комплексный полином. Корни числителя называются *нулями*, тогда как корни знаменателя называются *полюсами*. Поскольку полюса и нули могут быть комплексными числами, часто говорят, что они имеют свое "местоположение" на комплексной плоскости. Сложные системы имеют большее количество полюсов и нулей, чем простые. Проектирование рекурсивных фильтров начинается с выбора местоположения полюсов и нулей, а затем поиска соответствующих рекурсивных коэффициентов (или аналоговых компонент). Например, фильтры Баттерворта обладают полюсами, лежащими на комплексной плоскости, на *окружности*, тогда как у фильтров Чебышева они лежат на *эллипсе*. Это является темой Глав 32 и 33.

Ответ 2 – полюса это сундуки, заполненные волшебным порошком. Чем больше полюсов в фильтре, тем лучше работает фильтр. Теперь шутки в сторону, дело в том, что Вы можете очень эффективно использовать эти фильтры, вообще не зная стоящей за ними утомительной математики. Проектирование же фильтров - это специальность. В реальной практике, большинство инженеров, ученых и программистов думают категориями ответа 2, нежели чем ответа 1.

На рис. 20.2 показаны частотные характеристики нескольких фильтров Чебышева с *0,5%* пульсацией. Для используемого здесь метода число полюсов должно быть *четным*. Частота среза каждого фильтра измеряется в месте, где амплитуда пересекает уровень 0,707 (-3dB). Фильтры с частотой среза вблизи *0* или *0,5* обладают более крутым завалом, чем фильтры с частотой среза в центральной части частотного диапазона. Например, двухполюсный фильтр с f_C*=0,05* имеет точно такой же завал, что и четырех полюсный фильтр с f_C*=0,25*. Это – удача; по причине шума округления, вблизи *0* и *0,5* может использоваться меньшее количество полюсов. Более подробно об этом позже.

Существует два способа нахождения рекурсивных коэффициентов без использования z-преобразования. Первый способ – путь трусливых: использовать

таблицу. В таблицах 20.1 и 20.2 даны рекурсивные коэффициенты для соответственно низкочастотных и высокочастотных фильтров с *0,5%* пульсацией в полосе пропускания. Если Вам всего лишь нужен быстрый и черновой проект, скопируйте соответствующие коэффициенты в свою программу, и у Вас все готово.

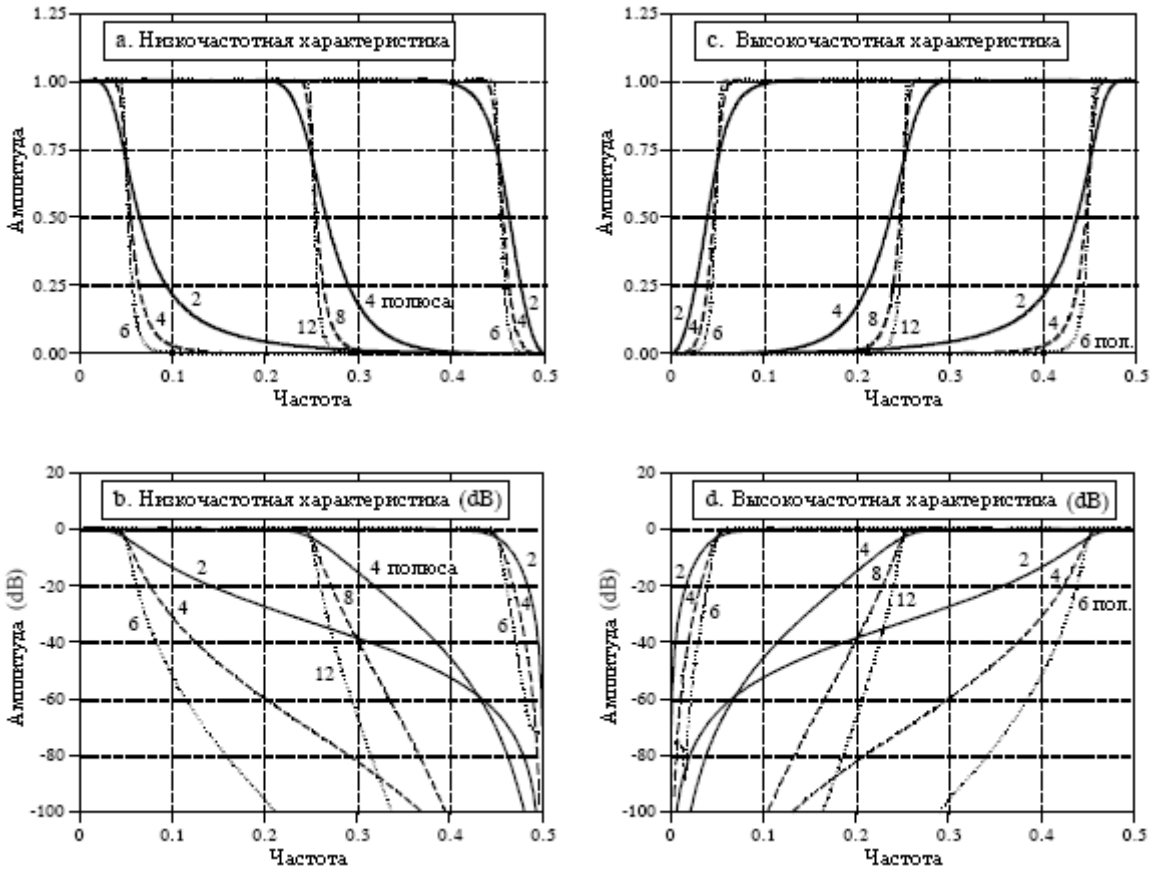


Рис. 20.2 Частотные характеристики фильтров Чебышева

Существуют две проблемы использования таблиц для проектирования цифровых фильтров. Во-первых, таблицы дают ограниченный выбор параметров. Например, таблица 20.1 дает всего *12* различных частот среза, максимум *6* полюсов на фильтр и *никакого* выбора величины пульсаций в полосе пропускания. В отсутствии возможности осуществлять выбор параметров из непрерывной области значений, проектирование фильтра не может быть *оптимизировано*. Во-вторых, коэффициенты должны быть перенесены из таблицы в программу вручную. Это – процесс, отнимающий очень много времени, который отобьет у Вас охоту пытаться пробовать альтернативные значения.

Вместо использования табулированных величин, рассмотрим включение в Вашу программу подпрограммы, которая сама *вычисляет* коэффициенты. Такая программа показана в таблице 20.4. Хорошим известием является то, что по структуре программа относительно проста. После того, как введены четыре параметра фильтра, программа выплевывает коэффициенты "a" и "b" в массивы A[ ] и B[ ]. Плохим известием является то, что программа вызывает подпрограмму приведенную в таблице 20.5. На первый взгляд эта подпрограмма выглядит действительно безобразно. Но не отчаивайтесь; она не так уж плоха, как это кажется! В строке 1120 находится одно простое ветвление. Все остальное в подпрограмме является прямым перемалыванием чисел. В программу поступают шесть переменных, покидают программу пять переменных, а в пределах программы используются пятнадцать временных переменных (плюс индексы). В таблице 20.6 приведены два набора тестовых данных для отладки этой подпрограммы. Детальная работа этой программы обсуждается в Главе 31.

Таблица 20.1

f_C	2 полюса	4 полюса	6 полюсов
0,01	a_0=8,663387E-04 a_1=1,732678E-03 b_1=1,919129E+00 a_2=8,663387E-04 b_2=-9,225943E-01	a_0=4,149425E-07 (!! Не устойчив !!) a_1=1,659770E-06 b_1=3,893453E+00 a_2=2,489655E-06 b_2=-5,688233E+00 a_3=1,659770E-06 b_3=3,695783E+00 a_4=4,149425E-07 b_4=-9,010106E-01	a_0=1,391351E-10 (!! Не устойчив !!) a_1=8,348109E-10 b_1=5,883343E+00 a_2=2,087027E-09 b_2=-1,442798E+01 a_3=2,782703E-09 b_3=1,887786E+01 a_4=2,087027E-09 b_4=-1,389914E+01 a_5=8,348109E-10 b_5=5,459909E+00 a_6=1,391351E-10 b_6=-8,939932E-01
0,025	a_0=5,112374E-03 a_1=1,022475E-02 b_1=1,797154E+00 a_2=5,112374E-03 b_2=-8,176033E-01	a_0=1,504626E-05 a_1=6,018503E-05 b_1=3,725385E+00 a_2=9,027754E-05 b_2=-5,226004E+00 a_3=6,018503E-05 b_3=3,270902E+00 a_4=1,504626E-05 b_4=-7,705239E-01	a_0=3,136210E-08 (!! Не устойчив !!) a_1=1,881726E-07 b_1=5,330512E+00 a_2=2,656634E-05 b_2=-1,196611E+01 a_3=3,542179E-05 b_3=1,447067E+01 a_4=2,656634E-05 b_4=-9,937710E+00 a_5=1,062654E-05 b_5=3,673283E+00 a_6=1,771089E-06 b_6=-5,707561E-01
0,05	a_0=1,868823E-02 a_1=3,737647E-02 b_1=1,593937E+00 a_2=1,868823E-02 b_2=-6,686903E-01	a_0=2,141509E-04 a_1=8,566037E-04 b_1=3,425455E+00 a_2=1,284906E-03 b_2=-4,479272E+00 a_3=8,566037E-04 b_3=2,643718E+00 a_4=2,141509E-04 b_4=-5,933269E-01	a_0=1,771089E-06 a_1=1,062654E-05 b_1=5,330512E+00 a_2=2,656634E-05 b_2=-1,196611E+01 a_3=3,542179E-05 b_3=1,447067E+01 a_4=2,656634E-05 b_4=-9,937710E+00 a_5=1,062654E-05 b_5=3,673283E+00 a_6=1,771089E-06 b_6=-5,707561E-01
0,075	a_0=3,869430E-02 a_1=7,738860E-02 b_1=1,392667E+00 a_2=3,869430E-02 b_2=-5,474446E-01	a_0=9,726342E-04 a_1=3,890537E-03 b_1=3,103944E+00 a_2=5,835806E-03 b_2=-3,774453E+00 a_3=3,890537E-03 b_3=2,111238E+00 a_4=9,726342E-04 b_4=-4,562908E-01	a_0=1,797538E-05 a_1=1,078523E-04 b_1=4,921746E+00 a_2=2,696307E-04 b_2=-1,035734E+01 a_3=3,595076E-04 b_3=1,189764E+01 a_4=2,696307E-04 b_4=-7,854533E+00 a_5=1,078523E-04 b_5=2.822109E+00 a_6=1,797538E-05 b_6=-4,307710E-01
0,1	a_0=6,372802E-02 a_1=1,274560E-01 b_1=1,194365E+00 a_2=6,372802E-02 b_2=-4,492774E-01	a_0=2,780755E-03 a_1=1,112302E-02 b_1=2,764031E+00 a_2=1,668453E-02 b_2=-3,122854E+00 a_3=1,112302E-02 b_3=1,664554E+00 a_4=2,780755E-03 b_4=-3,502232E-01	a_0=9,086148E-05 a_1=5,451688E-04 b_1=4,470118E+00 a_2=1,362922E-03 b_2=-8,755594E+00 a_3=1,817229E-03 b_3=9,543712E+00 a_4=1,362922E-03 b_4=-6,079376E+00 a_5=5,451688E-04 b_5=2,140062E+00 a_6=9,086148E-05 b_6=-3,247363E-01
0,15	a_0=1,254285E-01 a_1=2,508570E-01 b_1=8,070778E-01 a_2=1,254285E-01 b_2=-3,087918E-01	a_0=1,180009E-02 a_1=4,720034E-02 b_1=2,039039E+00 a_2=7,080051E-02 b_2=-2,012961E+00 a_3=4,720034E-02 b_3=9,897915E-01 a_4=1,180009E-02 b_4=-2,046700E-01	a_0=8,618665E-04 a_1=5,171199E-03 b_1=3,455239E+00 a_2=1,292800E-02 b_2=-5,754735E+00 a_3=1,723733E-02 b_3=5,645387E+00 a_4=1,292800E-02 b_4=-3,394902E+00 a_5=5,171199E-03 b_5=1,177469E+00 a_6=8,618665E-04 b_6=-1,836195E-01
0,2	a_0=1,997396E-01 a_1=3,994792E-01 b_1=4,291048E-01 a_2=1,997396E-01 b_2=-2,280633E-01	a_0=3,224554E-02 a_1=1,289821E-01 b_1=1,265912E+00 a_2=1,934732E-01 b_2=-1,203878E+00 a_3=1,289821E-01 b_3=5,405908E-01 a_4=3,224554E-02 b_4=-1,185538E-01	a_0=4,187408E-03 a_1=2,512445E-02 b_1=2,315806E+00 a_2=6,281112E-02 b_2=-3,293726E+00 a_3=8,374816E-02 b_3=2,904826E+00 a_4=6,281112E-02 b_4=-1,694128E+00 a_5=2,512445E-02 b_5=6,021426E-01 a_6=4,187408E-03 b_6=-1,029147E-01
0,25	a_0=2,858110E-01 a_1=5,716221E-01 b_1=5,423258E-02 a_2=2,858110E-01 b_2=-1,974768E-01	a_0=7,015301E-02 a_1=2,806120E-0 b_1=4,541481E-01 a_2=4,209180E-01 b_2=-7,417536E-01 a_3=2,806120E-01 b_3=2,361222E-01 a_4=7,015301E-02 b_4=-7,096476E-02	a_0=1,434449E-02 a_1=8,606697E-02 b_1=1,076052E+00 a_2=2,151674E-01 b_2=-1,662847E+00 a_3=2,868899E-01 b_3=1,191063E+00 a_4=2,151674E-01 b_4=-7,403087E-01 a_5=8,606697E-02 b_5=2,752158E-01 a_6=1,434449E-02 b_6=-5,722251E-02
0,3	a_0=3,849163E-01 a_1=7,698326E-01 b_1=-3,249116E-01 a_2=3,849163E-01 b_2=-2,147536E-01	a_0=1,335566E-01 a_1=5,342263E-01 b_1=-3,904486E-01 a_2=8,013394E-01 b_2=-6,784138E-01 a_3=5,342263E-01 b_3=-1,412021E-02 a_4=1,335566E-01 b_4=-5,392238E-02	a_0=3,997487E-02 a_1=2,398492E-01 b_1=-2,441152E-01 a_2=5,996231E-01 b_2=-1,130306E+00 a_3=7,994975E-01 b_3=1,063167E-01 a_4=5,996231E-01 b_4=-3,463299E-01 a_5=2,398492E-01 b_5=8,882992E-02 a_6=3,997487E-02 b_6=-3,278741E-02
0,35	a_0=5,001024E-01 a_1=1,000205E+00 b_1=-7,158993E-01 a_2=5,001024E-01 b_2=-2,845103E-01	a_0=2,340973E-01 a_1=9,363892E-01 b_1=-1,263672E+00 a_2=1,404584E+00 b_2=-1,080487E+00 a_3=9,363892E-01 b_3=-3,276296E-01 a_4=2,340973E-01 b_4=-7,376791E-02	a_0=9,792321E-02 a_1=5,875393E-01 b_1=-1,627573E+00 a_2=1,468848E+00 b_2=-1,955020E+00 a_3=1,958464E+00 b_3=-1,075051E+00 a_4=1,468848E+00 b_4=-5,106501E-01 a_5=5,875393E-01 b_5=-7,239843E-02 a_6=9,792321E-02 b_6=-2,639193E-02
0,40	a_0=6,362308E-01 a_1=1,272462E+00 b_1=-1,125379E+00	a_0=3,896966E-01 a_1=1,558787E+00 b_1=-2,161179E+00	a_0=2,211834E-01 a_1=1,327100E+00 b_1=-3,058672E+00

	a_2=6,362308E-01 b_2=-4,195441E-01	a_2=2,338180E+00 b_2=-2,033992E+00 a_3=1,558787E+00 b_3=-8,789098E-01 a_4=3,896966E-01 b_4=-1,610655E-01	a_2=3,317751E+00 b_2=-4,390465E+00 a_3=4,423668E+00 b_3=-3,523254E+00 a_4=3,317751E+00 b_4=-1,684185E+00 a_5=1,327100E+00 b_5=-4,414881E-01 a_6=2,211834E-01 b_6=-5,767513E-02
0,45	a_0=8,001101E-01 a_1=1,600220E+00 b_1=-1,556269E+00 a_2=8,001101E-01 b_2=-6,441713E-01	a_0=6,291693E-01 a_1=2,516677E+00 b_1=-3,077062E+00 a_2=3,775016E+00 b_2=-3,641323E+00 a_3=2,516677E+00 b_3=-1,949229E+00 a_4=6,291693E-01 b_4=-3,990945E-01	a_0=4,760635E-01 a_1=2,856381E+00 b_1=-4,522403E+00 a_2=7,140952E+00 b_2=-8,676844E+00 a_3=9,521270E+00 b_3=-9,007512E+00 a_4=7,140952E+00 b_4=-5,328429E+00 a_5=2,856381E+00 b_5=-1,702543E+00 a_6=4,760635E-01 b_6=-2,303303E-01

Таблица 20.2

f_C	2 полюса	4 полюса	6 полюсов
0,01	a_0=9,567529E-01 a_1=-1,913506E+00 b_1=1,911437E+00 a_2=9,567529E-01 b_2=-9,155749E-01	a_0=9,121579E-01 (!! Не устойчив !!) a_1=-3,648632E+00 b_1=3,815952E+00 a_2=5,472947E+00 b_2=-5,465026E+00 a_3=-3,648632E+00 b_3=3,481295E+00 a_4=9,121579E-01 b_4=-8,322529E-01	a_0=8,630195E-01 (!! Не устойчив !!) a_1=-5,178118E+00 b_1=5,705102E+00 a_2=1,294529E+01 b_2=-1,356935E+01 a_3=-1,726039E+01 b_3=1,722231E+01 a_4=1,294529E+01 b_4=-1,230214E+01 a_5=-5,178118E+00 b_5=4,689218E+00 a_6=8,630195E-01 b_6=-7,451429E-01
0,025	a_0=8,950355E-01 a_1=-1,790071E+00 b_1=1,777932E+00 a_2=8,950355E-01 b_2=-8,022106E-01	a_0=7,941874E-01 a_1=-3,176750E+00 b_1=3,538919E+00 a_2=4,765125E+00 b_2=-4,722213E+00 a_3=-3,176750E+00 b_3=2,814036E+00 a_4=7,941874E-01 b_4=-6,318300E-01	a_0=6,912863E-01 (!! Не устойчив !!) a_1=-4,147718E+00 b_1=5,261399E+00 a_2=1,036929E+01 b_2=-1,157800E+01 a_3=-1,382573E+01 b_3=1,363599E+01 a_4=1,036929E+01 b_4=-9,063840E+00 a_5=-4,147718E+00 b_5=3,223738E+00 a_6=6,912863E-01 b_6=-4,793541E-01
0,05	a_0=8,001102E-01 a_1=-1,600220E+00 b_1=1,556269E+00 a_2=8,001102E-01 b_2=-6,441715E-01	a_0=6,291694E-01 a_1=-2,516678E+00 b_1=3,077062E+00 a_2=3,775016E+00 b_2=-3,641324E+00 a_3=-2,516678E+00 b_3=1,949230E+00 a_4=6,291694E-0 b_4=-3,990947E-01	a_0=4,760636E-01 a_1=-2,856382E+00 b_1=4,522403E+00 a_2=7,140954E+00 b_2=-8,676846E+00 a_3=-9,521272E+00 b_3=9,007515E+00 a_4=7,140954E+00 b_4=-5,328431E+00 a_5=-2,856382E+00 b_5=1,702544E+00 a_6=4,760636E-01 b_6=-2,303304E-01
0,075	a_0=7,142028E-01 a_1=-1,428406E+00 b_1=1,338264E+00 a_2=7,142028E-01 b_2=-5,185469E-01	a_0=4,965350E-01 a_1=-1,986140E+00 b_1=2,617304E+00 a_2=2,979210E+00 b_2=-2,749252E+00 a_3=-1,986140E+00 b_3=1,325548E+00 a_4=4,965350E-01 b_4=-2,524546E-01	a_0=3,259100E-01 a_1=-1,955460E+00 b_1=3,787397E+00 a_2=4,888651E+00 b_2=-6,288362E+00 a_3=-6,518201E+00 b_3=5,747801E+00 a_4=4,888651E+00 b_4=-3,041570E+00 a_5=-1,955460E+00 b_5=8,808669E-01 a_6=3,259100E-01 b_6=-1,122464E-01
0,1	a_0=6,362307E-01 a_1=-1,272461E+00 b_1=1,125379E+00 a_2=6,362307E-01 b_2=-4,195440E-01	a_0=3,896966E-01 a_1=-1,558786E+00 b_1=2,161179E+00 a_2=2,338179E+00 b_2=-2,033991E+00 a_3=-1,558786E+00 b_3=8,789094E-01 a_4=3,896966E-01 b_4=-1,610655E-01	a_0=2,211833E-01 a_1=-1,327100E+00 b_1=3,058671E+00 a_2=3,317750E+00 b_2=-4,390464E+00 a_3=-4,423667E+00 b_3=3,523252E+00 a_4=3,317750E+00 b_4=-1,684184E+00 a_5=-1,327100E+00 b_5=4,414878E-01 a_6=2,211833E-01 b_6=-5,767508E-02
0,15	a_0=5,001024E-01 a_1=-1,000205E+00 b_1=7,158993E-01 a_2=5,001024E-01 b_2=-2,845103E-01	a_0=2,340973E-01 a_1=-9,363892E-01 b_1=1,263672E+00 a_2=1,404584E+00 b_2=-1,080487E+00 a_3=-9,363892E-01 b_3=3,276296E-01 a_4=2,340973E-01 b_4=-7,376791E-02	a_0=9,792321E-02 a_1=-5,875393E-01 b_1=1,627573E+00 a_2=1,468848E+00 b_2=-1,955020E+00 a_3=-1,958464E+00 b_3=1,075051E+00 a_4=1,468848E+00 b_4=-5,106501E-01 a_5=-5,875393E-01 b_5=7,239843E-02 a_6=9,792321E-02 b_6=-2,639193E-02
0,2	a_0=3,849163E-01 a_1=-7,698326E-01 b_1=3,249116E-01 a_2=3,849163E-01 b_2=-2,147536E-01	a_0=1,335566E-01 a_1=-5,342262E-01 b_1=3,904484E-01 a_2=8,013393E-01 b_2=-6,784138E-01 a_3=-5,342262E-01 b_3=1,412016E-02 a_4=1,335566E-01 b_4=-5,392238E-02	a_0=3,997486E-02 a_1=-2,398492E-01 b_1=2,441149E-01 a_2=5,996230E-01 b_2=-1,130306E+00 a_3=-7,994973E-01 b_3=-1,063169E-01 a_4=5,996230E-01 b_4=-3,463299E-01 a_5=-2,398492E-01 b_5=-8,882996E-02 a_6=3,997486E-02 b_6=-3,278741E-02
0,25	a_0=2,858111E-01 a_1=-5,716222E-01 b_1=-5,423243E-02 a_2=2,858111E-01 b_2=-1,974768E-01	a_0=7,015302E-02 a_1=-2,806121E-01 b_1=-4,541478E-01 a_2=4,209182E-01 b_2=-7,417535E-01 a_3=-2,806121E-01 b_3=-2,361221E-01 a_4=7,015302E-02 b_4=-7,096475E-02	a_0=1,434450E-02 a_1=-8,606701E-02 b_1=-1,076051E+00 a_2=2,151675E-01 b_2=-1,662847E+00 a_3=-2,868900E-01 b_3=-1,191062E+00 a_4=2,151675E-01 b_4=-7,403085E-01 a_5=-8,606701E-02 b_5=-2,752156E-01 a_6=1,434450E-02 b_6=-5,722250E-02
0,3	a_0=1,997396E-01 a_1=-3,994792E-01 b_1=-4,291049E-01 a_2=1,997396E-01 b_2=-2,280633E-01	a_0=3,224553E-02 a_1=-1,289821E-01 b_1=-1,265912E+00 a_2=1,934732E-01 b_2=-1,203878E+00 a_3=-1,289821E-01 b_3=-5,405908E-01	a_0=4,187407E-03 a_1=-2,512444E-02 b_1=-2,315806E+00 a_2=6,281111E-02 b_2=-3,293726E+00 a_3=-8,374815E-02 b_3=-2,904827E+00

		a_4=3,224553E-02 b_4=-1,185538E-01	a_4=6,281111E-02 b_4=-1,694129E+00 a_5=-2,512444E-02 b_5=-6,021426E-01 a_6=4,187407E-03 b_6=-1,029147E-01
0,35	a_0=1,254285E-01 a_1=-2,508570E-01 b_1=-8,070777E-01 a_2=1,254285E-01 b_2=-3,087918E-01	a_0=1,180009E-02 a_1=-4,720035E-02 b_1=-2,039039E+00 a_2=7,080051E-02 b_2=-2,012961E+00 a_3=-4,720035E-02 b_3=-9,897915E-01 a_4=1,180009E-02 b_4=-2,046700E-01	a_0=8,618665E-04 a_1=-5,171200E-03 b_1=-3,455239E+00 a_2=1,292800E-02 b_2=-5,754734E+00 a_3=-1,723733E-02 b_3=-5,645387E+00 a_4=1,292800E-02 b_4=-3,394902E+00 a_5=-5,171200E-03 b_5=-1,177469E+00 a_6=8,618665E-04 b_6=-1,836195E-01
0,40	a_0=6,372801E-02 a_1=-1,274560E-01 b_1=-1,194365E+00 a_2=6,372801E-02 b_2=-4,492774E-01	a_0=2,780754E-03 a_1=-1,112302E-02 b_1=-2,764031E+00 a_2=1,668453E-02 b_2=-3,122854E+00 a_3=-1,112302E-02 b_3=-1,664554E+00 a_4=2,780754E-03 b_4=-3,502233E-01	a_0=9,086141E-05 a_1=-5,451685E-04 b_1=-4,470118E+00 a_2=1,362921E-03 b_2=-8,755595E+00 a_3=-1,817228E-03 b_3=-9,543712E+00 a_4=1,362921E-03 b_4=-6,079377E+00 a_5=-5,451685E-04 b_5=-2,140062E+00 a_6=9,086141E-05 b_6=-3,247363E-01
0,45	a_0=1,868823E-02 a_1=-3,737647E-02 b_1=-1,593937E+00 a_2=1,868823E-02 b_2=-6,686903E-01	a_0=2,141509E-04 a_1=-8,566037E-04 b_1=-3,425455E+00 a_2=1,284906E-03 b_2=-4,479272E+00 a_3=-8,566037E-04 b_3=-2,643718E+00 a_4=2,141509E-04 b_4=-5,933269E-01	a_0=1,771089E-06 a_1=-1.062654E-05 b_1=-5,330512E+00 a_2=2,656634E-05 b_2=-1,196611E+01 a_3=-3,542179E-05 b_3=-1,447067E+01 a_4=2,656634E-05 b_4=-9,937710E+00 a_5=-1,062654E-05 b_5=-3,673283E+00 a_6=1,771089E-06 b_6=-5,707561E-01

Перерегулирование переходной характеристики

Фильтры Баттерворта и Чебышева имеют увеличивающееся с увеличением числа полюсов от *5%* до *30%* перерегулирование переходной характеристики. На рис. 20.3a показана переходная характеристика для двух образцов фильтра Чебышева с частотой среза *0,05*. На рис. 20.3b показано нечто, что является уникальным для цифровых фильтров и не имеет повторения в аналоговой электронике: величина перерегулирования переходной характеристики в незначительной степени зависит от частоты среза фильтра. Чрезмерное перерегулирование и звон переходной характеристики является результатом того, что фильтр Чебышева оптимизирован для *частотной области* за счет *временной области*.

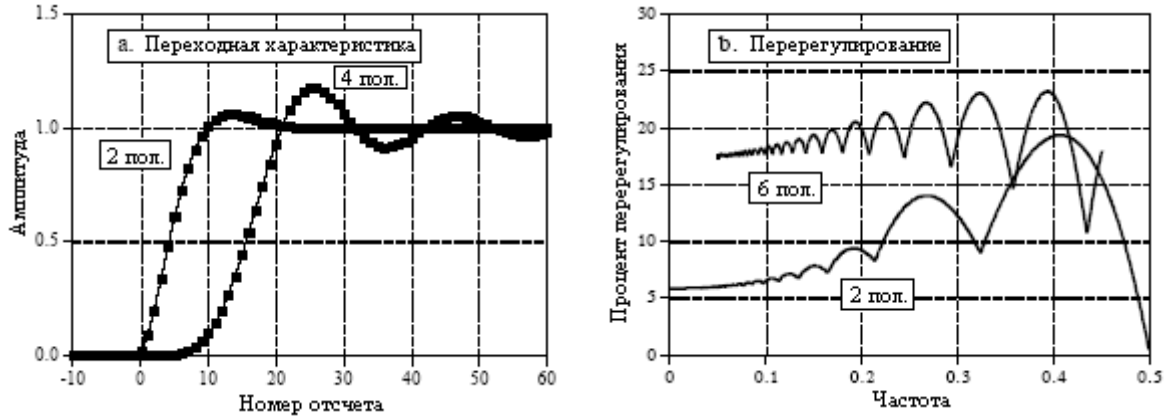


Рис. 20.3 Переходные характеристики фильтров Чебышева

Устойчивость

Главное ограничение цифровых фильтров, реализованных при помощи свертки - *время исполнения*. Если Вы намерены дождаться результата, то здесь можно получить почти любую характеристику фильтра. Рекурсивные фильтры являются прямой противоположностью этому. Они молниеносно исполняются, однако их рабочие

характеристики ограничены. Например, рассмотрим *6* полюсный низкочастотный фильтр с *0,5%* пульсацией и частотой среза равной *0,01*. Рекурсивные коэффициенты для этого фильтра могут быть получены из таблицы 20.1:

a_0=1,391351E-10
a_1=8,348109E-10 b_1=5,883343E+00
a_2=2,087027E-09 b_2=-1,442798E+01
a_3=2,782703E-09 b_3=1,887786E+01
a_4=2,087027E-09 b_4=-1,389914E+01
a_5=8,348109E-10 b_5=5,459909E+00
a_6=1,391351E-10 b_6=-8,939932E-01.

Посмотрите внимательно на эти коэффициенты. Абсолютное значение коэффициентов "b" - порядка *десяти*. При использовании одинарной точности, шум округления для каждого из этих чисел приблизительно составляет одну десятимиллионную значения, т.е. 10^{-6}. Теперь взгляните на коэффициенты "a", со значением порядка 10^{-9}. Здесь определенно что-то не так. Вклад входного сигнала (через коэффициенты "a") будет *1000* раз меньшее, чем *шум* от предварительно вычисленного выходного сигнала (через коэффициенты "b"). Этот фильтр не будет работать! Короче говоря, число полюсов, которое может использоваться в фильтре, ограничивается шумом округления. Фактическое число полюсов будет слегка зависеть от пульсации и от того, является ли этот фильтр высокочастотным или низкочастотным. Приблизительное число полюсов для одинарной точности приведено в таблице 20.3.

Таблица 20.3

Частота среза	*0,02*	*0,05*	*0,10*	*0,25*	*0,40*	*0,45*	*0,48*
Максимальное число полюсов	*4*	*6*	*10*	*20*	*10*	*6*	*4*

По мере приближения к этому пределу характеристики фильтра начнут ухудшаться: увеличится перерегулирование переходной характеристики, ослабление в полосе заграждения станет хилым, а на частотной характеристике появится чрезмерная пульсация. Если пойти с фильтром значительно дальше, или же в коэффициентах окажется ошибка, на выходе по всей вероятности возникнут колебания до тех пор, пока не произойдет переполнения.

Существует два способа расширения максимального числа полюсов, которое может быть использовано. Во-первых, использовать двойную точность. Это требует также использования двойной точности при вычислении коэффициентов (включая значение π).

Второй способ состоит в том, чтобы реализовать фильтр *покаскадно*. Например, шестиполюсный фильтр рассматривается как последовательное соединение трех двухполюсных каскадов. Для облегчения программирования, программа в таблице 20.4 объединяет эти три каскада в один набор рекурсивных коэффициентов. Тем не менее, фильтр является более устойчивым, если реализуется в виде исходных трех отдельных каскадов. Это требует знания для каждого из каскадов коэффициентов "a" и "b". Последние могут быть получены из программы в таблице 20.4. Для каждого каскада в последовательном соединении один раз вызывается подпрограмма, приведенная в таблице 20.5. Например, для шестиполюсного фильтра она вызывается три раза. По завершении подпрограммы, в основную программу возвращаются пять переменных: *A0*, *A1*, *A2*, *B1*, и *B2*. Они являются рекурсивными коэффициентами для обрабатываемых в данный момент двухполюсных каскадов и могут быть использованы для покаскадной реализации фильтра.

Таблица 20.4

```
100 'CHEBYSHEV FILTER- RECURSION COEFFICIENT CALCULATION
110 '
120                                         'INITIALIZE VARIABLES
130 DIM A[22]                               'holds the "a" coefficients upon program completion
140 DIM B[22]                               'holds the "b" coefficients upon program completion
150 DIM TA[22]                              'internal use for combining stages
160 DIM TB[22]                              'internal use for combining stages
170 '
180 FOR I% = 0 TO 22
190 A[I%] = 0
200 B[I%] = 0
210 NEXT I%
220 '
230 A[2] = 1
240 B[2] = 1
250 PI = 3.14159265
260                                         'ENTER THE FOUR FILTER PARAMETERS
270 INPUT "Enter cutoff frequency (0 to .5): ", FC
280 INPUT "Enter 0 for LP, 1 for HP filter: ", LH
290 INPUT "Enter percent ripple (0 to 29): ", PR
300 INPUT "Enter number of poles (2,4,...20): ", NP
310 '
320 FOR P% = 1 TO NP/2                      'LOOP FOR EACH POLE-PAIR
330 '
340 GOSUB 1000                              'The subroutine in TABLE 20.5
350 '
360 FOR I% = 0 TO 22                        'Add coefficients to the cascade
370 TA[I%] = A[I%]
380 TB[I%] = B[I%]
390 NEXT I%
400 '
410 FOR I% = 2 TO 22
420 A[I%] = A0*TA[I%] + A1*TA[I%-1] + A2*TA[I%-2]
430 B[I%] = TB[I%] - B1*TB[I%-1] - B2*TB[I%-2]
440 NEXT I%
450 '
460 NEXT P%
470 '
480 B[2] = 0                                'Finish combining coefficients
490 FOR I% = 0 TO 20
500 A[I%] = A[I%+2]
510 B[I%] = -B[I%+2]
520 NEXT I%
530 '
540 SA = 0                                  'NORMALIZE THE GAIN
550 SB = 0
560 FOR I% = 0 TO 20
570 IF LH = 0 THEN SA = SA + A[I%]
580 IF LH = 0 THEN SB = SB + B[I%]
590 IF LH = 1 THEN SA = SA + A[I%] * (-1)^I%
```

```
600 IF LH = 1 THEN SB = SB + B[I%] * (-1)^I%
610 NEXT I%
620 '
630 GAIN = SA / (1 - SB)
640 '
650 FOR I% = 0 TO 20
660 A[I%] = A[I%] / GAIN
670 NEXT I%
680 '                                   'The final recursion coefficients are in A[ ] and B[ ]
690 END
```

Таблица 20.5

```
1000 'THIS SUBROUTINE IS CALLED FROM TABLE 20.4, LINE 340
1010 '
1020 ' Variables entering subroutine:     PI, FC, LH, PR, HP, P%
1030 ' Variables exiting subroutine:      A0, A1, A2, B1, B2
1040 ' Variables used internally:         RP, IP, ES, VX, KX, T, W, M, D, K,
1050 '                                    X0, X1, X2, Y1, Y2
1060 '
1070 '                                    'Calculate the pole location on the unit circle
1080 RP = -COS(PI/(NP*2) + (P%-1) * PI/NP)
1090 IP = SIN(PI/(NP*2) + (P%-1) * PI/NP)
1100 '
1110 '                                    'Warp from a circle to an ellipse
1120 IF PR = 0 THEN GOTO 1210
1130 ES = SQR( (100 / (100-PR))^2 -1 )
1140 VX = (1/NP) * LOG( (1/ES) + SQR( (1/ES^2) + 1) )
1150 KX = (1/NP) * LOG( (1/ES) + SQR( (1/ES^2) - 1) )
1160 KX = (EXP(KX) + EXP(-KX))/2
1170 RP = RP * ( (EXP(VX) - EXP(-VX) ) /2 ) / KX
1180 IP = IP * ( (EXP(VX) + EXP(-VX) ) /2 ) / KX
1190 '
1200 '                                    's-domain to z-domain conversion
1210 T = 2 * TAN(1/2)
1220 W = 2*PI*FC
1230 M = RP^2 + IP^2
1240 D = 4 - 4*RP*T + M*T^2
1250 X0 = T^2/D
1260 X1 = 2*T^2/D
1270 X2 = T^2/D
1280 Y1 = (8 - 2*M*T^2)/D
1290 Y2 = (-4 - 4*RP*T - M*T^2)/D
1300 '
1310 '                                    'LP TO LP, or LP TO HP transform
1320 IF LH = 1 THEN K = -COS(W/2 + 1/2) / COS(W/2 - 1/2)
1330 IF LH = 0 THEN K = SIN(1/2 - W/2) / SIN(1/2 + W/2)
1340 D = 1 + Y1*K - Y2*K^2
1350 A0 = (X0 - X1*K + X2*K^2)/D
1360 A1 = (-2*X0*K + X1 + X1*K^2 - 2*X2*K)/D
1370 A2 = (X0*K^2 - X1*K + X2)/D
```

```
1380 B1 = (2*K + Y1 + Y1*K^2 - 2*Y2*K)/D
1390 B2 = (-(K^2) - Y1*K + Y2)/D
1400 IF LH = 1 THEN A1 = -A1
1410 IF LH = 1 THEN B1 = -B1
1420 '
1430 RETURN
```

Таблица 20.6

Набор данных 1	**Набор данных 2**
Введите эти значения в подпрограмму:	
FC = 0.1 LH = 0 PR = 0 NP = 4 P% = 1 PI = 3.141592	FC = 0.1 LH = 1 PR = 10 NP = 4 P% = 2 PI = 3.141592
Эти значения должны быть вычислены к строке 1200:	
RP = -0.923879 IP = 0.382683 ES = не используется VX = не используется KX = не используется	RP = -0.136178 IP = 0.933223 ES = 0.484322 VX = 0.368054 KX = 1.057802
Эти значения должны быть вычислены к строке 1310:	
T = 1.092605 W = 0.628318 M = 1.000000 D = 9.231528 X0 = 0.129316 X1 = 0.258632 X2 = 0.129316 Y1 = 0.607963 Y2 = -0.125227	T = 1.092605 W = 0.628318 M = 0.889450 D = 5.656972 X0 = 0.211029 X1 = 0.422058 X2 = 0.211029 Y1 = 1.038784 Y2 = -0.789584
Эти значения должны быть возвращены в основную программу:	
A0 = 0.061885 A1 = 0.123770 A2 = 0.061885 B1 = 1.048600 B2 = -0.296140	A0 = 0.922919 A1 = -1.845840 A2 = 0.922919 B1 = 1.446913 B2 = -0.836653

ГЛАВА

21

СРАВНЕНИЕ ФИЛЬТРОВ

Решения, решения, решения! Как же определить, при таком многообразии выбора фильтров, какой же из них использовать? В этой главе показан жесткий турнир между фильтрами; выберем чемпионов от каждой из сторон, и пусть они сражаются и побеждают. В первом матче, для того чтобы понять, какая технология является лучше, *цифровые* фильтры сражаются с *аналоговыми*. Во втором туре, для того чтобы определить короля фильтров *частотной области*, фильтры с ограниченной окном синк функцией состязаются с фильтрами Чебышева. В последнем поединке, за чемпионский титул во *временной области*, фильтры скользящего среднего борются с однополюсными фильтрами. Довольно разговоров, пусть начнется соревнование!

Состязание № 1: Аналоговые фильтры против цифровых

Большинство цифровых сигналов берут свое начало из аналоговой электроники. Если сигнал должен быть отфильтрован то, что лучше - использовать аналоговый фильтр до оцифровки, или цифровой фильтр после оцифровки сигнала? Мы ответим, на этот вопрос, позволив нанести свои удары двум лучшим претендентам.

Целью будет - создать фильтр низких частот с частотой среза *1* кГц. За аналоговую сторону бой ведет шестиполюсный фильтр Чебышева с пульсацией *0,5*dB (*6%*). Как описывалось в Главе3, он может быть построен на *3* операционных усилителях, *12* резисторах и *6* конденсаторах. В цифровом углу размялся и уже готов драться фильтр с ограниченной окном синк функцией. Аналоговый сигнал оцифрован на частоте дискретизации *10* кГц, что делает частоту среза в масштабе цифровых частот равной *0,1*. Чтобы от 90% уровня до 10% уровня обеспечить такой же завал, как и у аналогового фильтра, длина ограниченной окном синк функции будет выбрана в *129* точек. По честному, так по честному. На рис. 21.1 показаны частотные и переходные характеристики этих двух фильтров.

Давайте сравнивать два фильтра по пунктам. Как показано на рис. 21.1a и рис. 21.1b, у аналогового фильтра в полосе пропускания наблюдается *6%* пульсация, тогда как у цифрового фильтра идеальная пологость (в пределах *0,02%*). Проектировщик аналоговых схем мог бы возразить, что величина пульсации может быть *выбрана* при проектировании; однако это промах по цели. Пологость, достижимая аналоговыми фильтрами ограничена точностью их резисторов и конденсаторов. Даже если проектируются характеристики Баттерворта (т.е. пульсация *0%*), фильтры такой сложности будут иметь остаточную пульсацию, возможно около *1%*. С другой стороны пологость цифровых фильтров, прежде всего, ограничена ошибкой округления, что делает их в *сотни* раз более пологими, чем их аналоговых соперников. Запишите одно очко в пользу цифровых фильтров.

Далее посмотрим на частотные характеристики в логарифмическом масштабе, показанные на рис. 21.1c и рис. 21.1d. И снова, цифровой фильтр становится явным победителем и по *завалу* и по *ослаблению в полосе заграждения*. Даже если улучшить качества аналогового фильтра добавлением дополнительных каскадов, он все же не может

сравниться с цифровым фильтром. Например, представьте, что Вам необходимо в 100 раз улучшить два этих параметра. Это может быть выполнено простой модификацией ограниченной окном синк функции, и фактически невозможно для аналоговой схемы. Еще два очка в пользу цифрового фильтра.

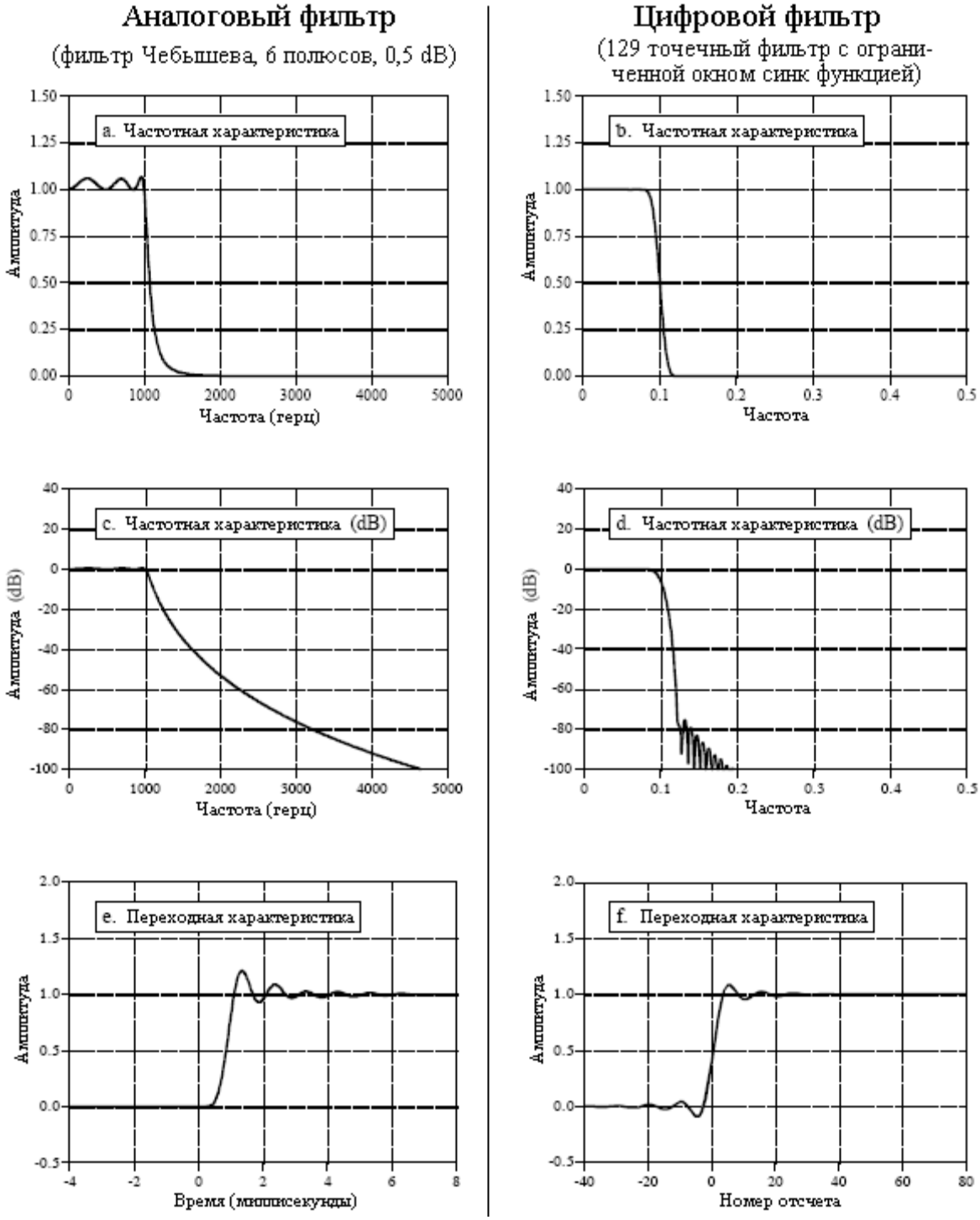


Рис. 21.1 Сравнение аналоговых и цифровых фильтров

Переходные характеристики двух фильтров показаны на рис. 21.1e и рис. 21.1f. Переходная характеристика цифрового фильтра симметрична относительно верхней и нижней частей ступени, т.е. обладает линейной фазой. Переходная характеристика аналогового фильтра *не* является симметричной, т.е. имеет нелинейную фазу. Еще одно очко в пользу цифрового фильтра. Наконец, аналоговый фильтр с одной стороны ступени

имеет перерегулирование порядка *20%*. Цифровой фильтр имеет перерегулирование порядка *10%*, но с двух сторон ступени. Поскольку и то и другое плохо, никому никаких очков не присуждается.

Несмотря на такой разгром, существует все же большое число приложений, где аналоговые фильтры нужно и должно использовать. Это не связано с фактической работой фильтра (т.е. с тем, что в него входит и, что из него выходит), но относится к общим преимуществам, которые аналоговые схемы имеют над цифровыми методами. Первое преимущество - это *скорость*: цифровое медленно; аналоговое быстро. Например, персональный компьютер при использовании БПФ свертки может осуществлять фильтрацию данных только со скоростью около *10000* отсчетов в секунду. Даже простые операционные усилители могут работать на частотах свыше *100* кГц вплоть до *1* МГц, от *10* до *100* раз быстрее, чем цифровые системы!

Второе присущее преимущество аналогового над цифровым - это *динамический диапазон*. Он укладывается в два вида. **Динамический диапазон по амплитуде -** является отношением между самым большим сигналом, который только можно пропустить через систему и свойственным системе шумом. Например, уровень насыщения *12* битового АЦП равен *4095*, а среднеквадратический шум квантования при равномерном распределении цифровых чисел, порядка *0,29*, динамический диапазон приблизительно *14000*. Для сравнения, стандартный операционный усилитель имеет напряжение насыщения порядка *20* вольт и внутренний шум порядка *2* микровольт, динамический диапазон около *десяти миллионов*. Так же, как и прежде, простой операционный усилитель оставляет цифровую систему в одной рубашке.

Другой вид - это **динамический диапазон по частоте**. Например, несложно спроектировать схему на операционном усилителе, одновременно обрабатывающую частоты лежащие между *0,01* Гц и *100* кГц (семь декад). Когда же это пытаются повторить при помощи цифровой системы, компьютер становится, просто завален данными. Например, при частоте дискретизации *200* кГц, для охвата одного целого периода сигнала с частотой *0,01* Гц требуется *20* миллионов точек. Вы, может быть, отметили, что частотные характеристики цифровых фильтров почти всегда вычерчиваются по оси частот в *линейном* масштабе, в тогда как аналоговые фильтры обычно показываются с *логарифмическим* масштабом по оси частот. Это связано с тем, что для того, чтобы показать исключительную работу цифровых фильтров, нужен линейный масштаб, в то время как для того чтобы показать огромный динамический диапазон аналоговых фильтров, необходим логарифмический масштаб.

Состязание № 2: Фильтры с ограниченной окном синк функцией против фильтров Чебышева

Как фильтры с ограниченной окном синк функцией, так и фильтры Чебышева спроектированы для отделения одной полосы частот от другой. Фильтры с ограниченной окном синк функцией являются КИХ-фильтрами, реализуемыми при помощи *свертки*, тогда как фильтры Чебышева являются БИХ-фильтрами, реализуемыми при помощи *рекурсии*. Какой же цифровой фильтр, из них, является лучшим в частотной области? Пусть покажут это в бою.

Соперником рекурсивного фильтра будет *6* полюсный низкочастотный фильтр Чебышева с *0,5%* пульсацией. Честное сравнение усложняется тем фактом, что с изменением частоты среза, частотная характеристика фильтра Чебышева изменяется. Мы будем использовать частоту среза равную *0,2* и выберем ядро фильтра с ограниченной окном синк функцией длиной в *51* точку. Как показано на рис. 21.2а, это заставит оба фильтра от 90% уровня до 10% уровня иметь один и тот же завал.

Ну а теперь начинается расталкивание и распихивание. У рекурсивного фильтра в полосе пропускания имеется *0,5%* пульсация, тогда как у фильтра с ограниченной окном

синк функцией наблюдается пологость. Однако, при необходимости, мы могли бы легко установить *0%* пульсацию и для рекурсивного фильтра. Никому никаких очков. На рис. 21.2b показано, что фильтр с ограниченной окном синк функцией в полосе заграждения имеет гораздо лучшее ослабление, чем фильтр Чебышева. Одно очко в пользу фильтра с ограниченной окном синк функцией.

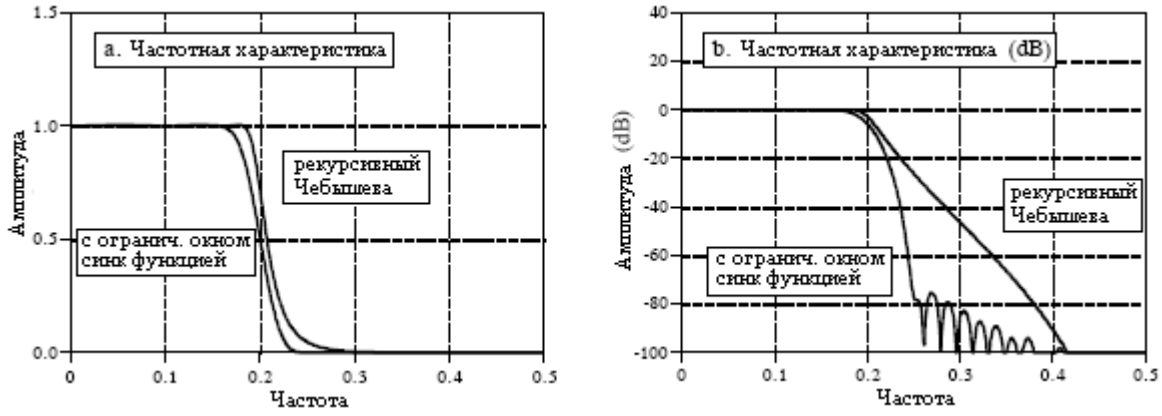


Рис. 21.2 Частотные характеристики фильтра с ограниченной окном синк функцией и фильтра Чебышева

На рис. 21.3 показаны переходные характеристики этих двух фильтров. Как Вы и должны были ожидать, для фильтров частотной области обе плохи. Рекурсивный фильтр имеет нелинейную фазу, правда она может быть скорректирована двунаправленной фильтрацией. Поскольку по этому параметру оба фильтра так безобразны, мы назовем это ничьей.

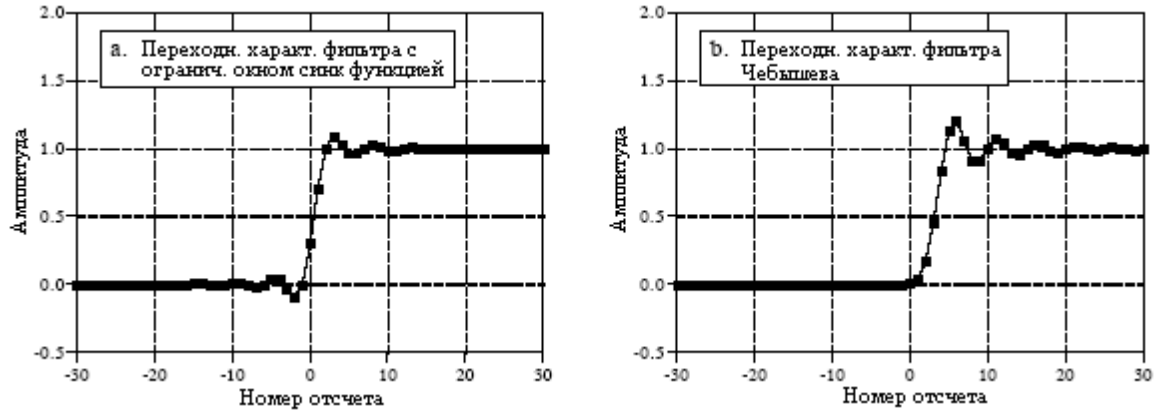


Рис. 21.3 Переходные характеристики фильтра с ограниченной окном синк функцией и фильтра Чебышева

До сих пор не было большой разницы между этими двумя фильтрами; если нужны средние характеристики, то любой из них сможет работать. Тяжелые удары наносятся лишь двумя критическими проблемами: *максимальные качественные характеристики* и *скорость*. Фильтр с ограниченной окном синк функцией – это силовая подстанция, в то время как фильтр Чебышева быстр и проворен. Предположим, что у Вас действительно жесткая задача по разделению частот, скажем, нужно изолировать сигнал в *100* милливольт с частотой *61* герц, который считывается с силовой линии *120* вольт с частотой *60* герц. На рис. 21.4 показывается, как сопоставляются два этих фильтра тогда, когда Вам нужны максимальные качественные характеристики. Здесь рекурсивным

фильтром является *6* полюсный фильтр Чебышева с *0,5%* пульсацией. Это максимальное число полюсов, которое может быть использовано на частоте среза равной *0,05*, при использовании чисел одинарной точности. Фильтр с ограниченной окном синк функцией использует *1001* точечное ядро фильтра, сформированное при помощи свертки с самим собой *501* точечного ядра фильтра с ограниченной окном синк функцией. Как показано в Главе 16, это обеспечивает большее ослабление в полосе заграждения.

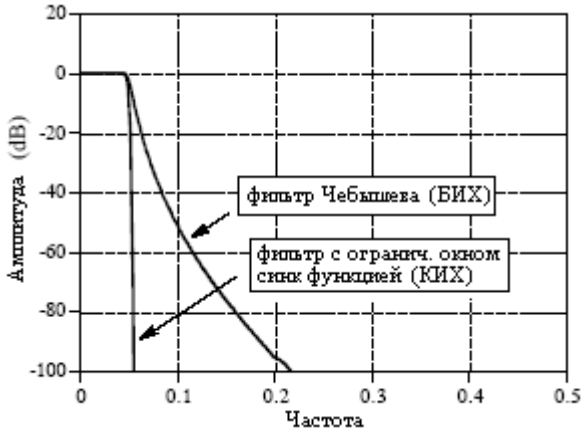


Рис. 21.4 Максимальные качественные характеристики КИХ и БИХ фильтров

Как же сопоставляются два этих фильтра с точки зрения достижения максимальных качественных характеристик? Фильтр с ограниченной окном синк функцией сокрушает фильтр Чебышева! Даже если рекурсивный фильтр был бы улучшен (большее число полюсов, многокаскадная реализация, двойная точность и т.д.), он все же, не идет ни в какое сравнение с качествами достигаемыми КИХ-фильтром. Это особенно впечатляет, когда Вы принимаете во внимание то, что фильтр с ограниченной окном синк функцией только начал бороться. Существуют сильные ограничения на максимально достижимые качества, которые может обеспечить рекурсивный фильтр. И наоборот, фильтр с ограниченной окном синк функцией можно довести до невероятных уровней. Это, конечно же, достигается только в том случае, если Вы пожелаете подождать результата. Что в свою очередь поднимает вторую критическую проблему: *скорость*.

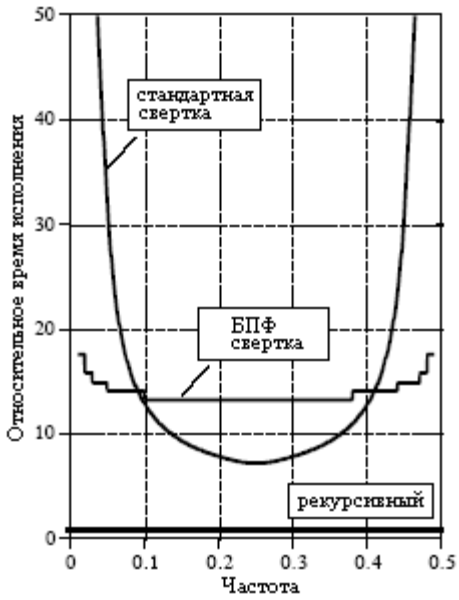


Рис. 21.5 Сравнение скоростей исполнения для КИХ и БИХ фильтров

Сравнение этих фильтров по скорости похоже на гонки Феррари с картингом. Рис. 21.5 показывает, насколько больше времени для исполнения требуется фильтру с ограниченной окном синк функцией, по сравнению с шести полюсным рекурсивным фильтром. Поскольку рекурсивный фильтр имеет более быстрый завал на низких и высоких частотах, то для того чтобы соревноваться в характеристиках (т.е. соблюсти честное сравнение), протяженность ядра фильтра с ограниченной окном синк функцией должна быть сделана *длиннее*. Это объясняет увеличенное время исполнения для фильтра с ограниченной окном синк функцией около частот близких к *0* и *0,5*. Важным моментом является то, что КИХ-фильтры, как и можно было ожидать, по величине примерно на порядок медленнее, чем сопоставимые БИХ фильтры (картинг: 15 миль в час, Феррари: 150 миль в час).

Состязание № 3: Фильтры скользящего среднего против однополюсных фильтров

Наше третье соревнование будет битвой фильтров временной области. Первым бойцом будет девяти точечный фильтр скользящего среднего. Его оппонентом в сегодняшнем состязании будет однополюсный рекурсивный фильтр, использующий двунаправленную технику. Для достижения сопоставимой частотной характеристики в однополюсном фильтре будет использовано затухание от отсчета к отсчету равное $x=0,70$. Сражение начинается на рис. 21.6, где показана частотная характеристика каждого фильтра. Ни одна из них не производит впечатления, но правда, разделение по частоте не является тем, для чего используются эти фильтры. Никаких очков ни одной из сторон.

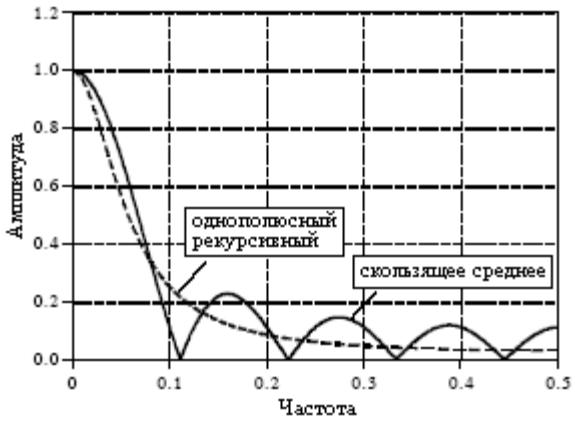


Рис. 21.6 Частотные характеристики фильтра скользящего среднего и однополюсного фильтра

На рис. 21.7 показаны переходные характеристики фильтров. На рис. 21.7a переходная характеристика фильтра скользящего среднего представляет собой прямую линию, наиболее быстрый путь перемещения от одного уровня к другому. На рис. 21.7b переходная характеристика рекурсивного фильтра более гладкая, что для некоторых приложений может быть лучше. По одному очку каждой из сторон.

В терминах качественных характеристик эти фильтры очень похожи, и выбор между ними часто делается на основе персональных предпочтений. Однако существуют два случая, где один фильтр имеет над другим небольшое превосходство. Это связано с компромиссом между временем *разработки* и временем *исполнения*. В первом случае, Вы хотите сократить время разработки и очень хотите взять более медленный фильтр. Например, у Вас может быть одноразовая задача, осуществить фильтрацию нескольких

тысяч точек. Поскольку прогон всей программы занимает всего несколько секунд, то бессмысленно тратить время на оптимизацию алгоритма. Почти всегда будут использоваться числа с плавающей запятой. Выбором может быть либо использование фильтра скользящего среднего, выполненного с помощью свертки, либо использование однополюсного рекурсивного фильтра. Победителем здесь является рекурсивный фильтр. Его будет немного проще программировать и модифицировать, и исполняться он будет намного быстрее.

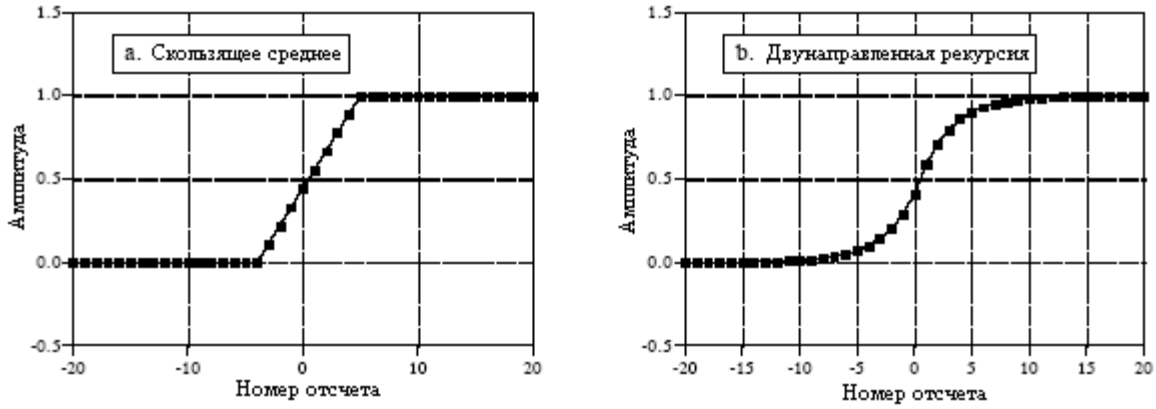


Рис. 21.7 Переходные характеристики фильтра скользящего среднего и двунаправленного однополюсного фильтра

Второй случай является прямо противоположным, Ваш фильтр должен функционировать настолько быстро насколько это возможно и для того, чтобы этого добиться, Вы хотите потратить дополнительное время на разработку. Например, этот фильтр может быть частью коммерческого продукта с возможностью прогона в *миллион* раз. Для достижения максимально возможной скорости, по-видимому, Вы будете использовать целые числа. Вашим выбором среди фильтров будет либо скользящее среднее, выполненное с помощью *рекурсии*, либо однополюсный рекурсивный фильтр, реализованный с помощью справочных таблиц или целочисленной математики. Здесь победителем является фильтр скользящего среднего. Он будет быстрее исполняться, и не восприимчив к проблемам разработки и реализации арифметики целых чисел.

ГЛАВА
22

ОБРАБОТКА ЗВУКА

Обработка звука охватывает множество разнообразных областей, причем все они включают в себя представление звука человеку-слушателю. Выделяются три области: (1) *высококачественное воспроизведение музыки*, такое как в аудио компакт-дисках, (2) *голосовая телекоммуникация*, это иное название для телефонных сетей и (3) *искусственная речь*, когда компьютеры генерируют и распознают образцы голоса человека. Тогда как эти приложения имеют различные цели и проблемы, их объединяет общий судья - ухо человека. Цифровая обработка сигнала произвела революционные изменения в этих и других областях обработки звука.

Слух человека

Ухо человека является чрезвычайно сложным органом. Ситуация осложняется еще и тем, что информация от *двух* ушей объединяется в запутанной нервной сети - мозге человека. Имейте в виду, что все последующее представляет собой лишь краткий обзор; существует множество тонких эффектов и плохо понимаемых явлений относящихся к слуху человека.

Рис. 22.1 иллюстрирует основную структуру и процессы, происходящие в ухе человека. *Внешнее ухо* состоит из двух частей: видимой оттопыренной складки кожи на прикрепленном к голове хряще, и *ушного канала* - трубки около 0,5 см в диаметре, расширяющейся приблизительно до 3 см внутри головы. Эти структуры направляют окружающие звуки к чувствительным органам *среднего и внутреннего уха*, расположенным в безопасности внутри костей черепа. Поперек конца канала уха протянут тонкий лист ткани называемый *барабанной перепонкой* или *ушным барабаном*. Звуковые волны, ударяясь о барабанную перепонку, заставляют ее вибрировать. Среднее ухо представляет собой набор косточек передающих эту вибрацию на *улитку* (внутреннее ухо), где она преобразуется в нервные импульсы. Улитка представляет собой трубку, заполненную жидкостью, в диаметре, грубо, около 2 мм и в длину 3 см. Ушная улитка свернута и напоминает маленькую раковину *улитки*, хотя на рис. 22.1 она показана прямой. Фактически термин *улитка* происходит от греческого слова *cochlea*.

Когда звуковая волна пытается пройти из воздуха в жидкость, через поверхность раздела передается только небольшая часть звука, тогда как остальная часть энергии отражается. Это происходит из-за того, что воздух обладает *низким* механическим импедансом (низким акустическим давлением и высокой скоростью частиц, являющейся результатом низкой плотности и высокой сжимаемости), в то время как *жидкость* имеет высокий механический импеданс. Выражаясь нетехническим языком, для того чтобы помахать рукой в воде требуется больше усилий, чем для того чтобы помахать в воздухе. Такая разница в механическом импедансе приводит к тому, что большая часть звука отражается от перехода воздух/жидкость.

Среднее ухо представляет собой *согласующую импедансы* сеть, которая увеличивает часть звуковой энергии, вошедшую в жидкость внутреннего уха. Например, у рыб нет барабанной перепонки или среднего уха, поскольку у них нет никакой необходимости слышать в воздухе. Большая часть преобразования импеданса происходит

за счет разницы в *площади* между барабанной перепонкой (получающей звук из воздуха) и *овальным окном* (передающим звук в жидкость, см. рис. 22.1). Барабанная перепонка имеет площадь приблизительно 60 мм^2, в то время как овальное окно, грубо, имеет площадь 4 мм^2. Поскольку давление равно силе деленной на площадь, такая разница в площади увеличивает давление звуковой волны примерно в 15 раз.

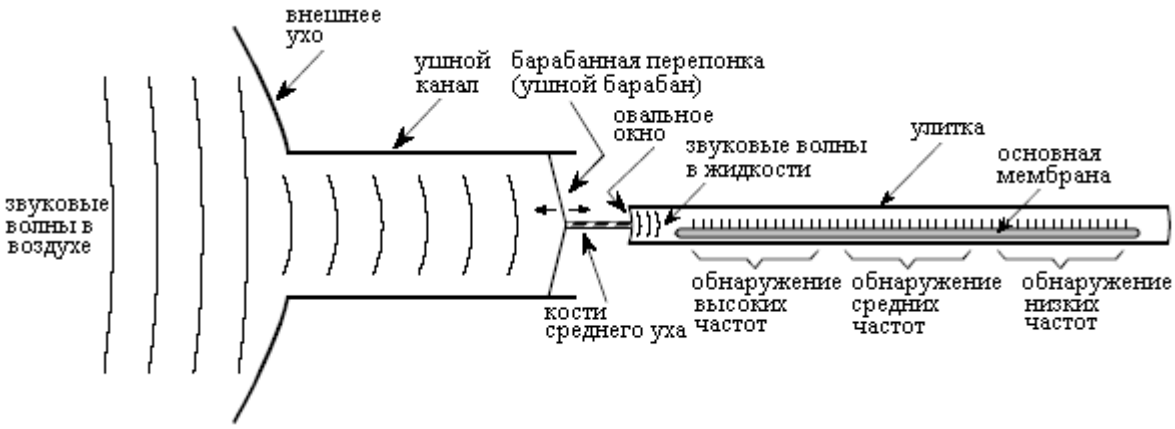


Рис. 22.1 Функциональная схема уха человека

Внутри улитки находится *основная мембрана*, поддерживающая структура для приблизительно 12000 чувствительных клеток, формирующих *слуховой нерв*. Около овального окна основная мембрана наиболее жесткая и становится более эластичной у противоположного конца, что позволяет ей действовать как *анализатору частотного спектра*. Под воздействием высокочастотного сигнала основная мембрана резонирует в тех местах, где она жесткая, это приводит к возбуждению ближайших к овальному окну нервных клеток. Аналогично, низкочастотные звуки возбуждают нервные клетки на дальнем конце основной мембраны. Это заставляет определенные волокна в улиточном нерве, откликаться на определенные частоты. Такая организация называется организацией по **принципу местоположения**, и сохраняется повсюду вдоль слухового пути в мозг.

Другая схема кодирования информации также использующаяся слухом человека, называется **принципом залпа**. Нервные клетки передают информацию посредством генерирования коротких электрических импульсов, называемых *потенциалами действия*. Нервные клетки на основной мембране могут кодировать звуковую информацию при помощи вырабатываемого в ответ на каждый период колебаний потенциала действия. Например, звуковая волна с частотой 200 герц может быть представлена нейроном, вырабатывающим 200 потенциалов действия в секунду. Однако это возможно только на частотах ниже приблизительно 500 герц - максимальной скорости, с которой нейроны могут вырабатывать потенциалы действия. Ухо человека решает данную проблему, позволяя нескольким нервным клеткам поочередно выполнять эту отдельную задачу. Например, тон с частотой 3000 герц может быть представлен с помощью *десяти* нервных клеток, попеременно выстреливающих 300 раз в секунду. Это расширяет диапазон принципа залпа приблизительно до 4 кГц, выше которого используется исключительно принцип местоположения.

Таблица 22-1 показывает соотношения между интенсивностью звука и воспринимаемой громкостью. Принято выражать интенсивность звука в логарифмическом масштабе, называемом **децибелы УМЗ** (уровень мощности звука). В этом масштабе *0* dB УМЗ это звуковая волна с мощностью 10^{-16} Вт/см^2, приблизительно самый слабый звук, обнаруживаемый ухом человека. Нормальная речь около *60* dB УМЗ, тогда как болевые повреждения уха происходят приблизительно при *140* dB УМЗ.

Разница между самыми громкими и самыми слабыми звуками, которые могут слышать люди, составляет приблизительно 120 dB, диапазон по амплитуде в один миллион. Слушатели могут обнаруживать *изменения* в громкости при изменении сигнала примерно на 1 dB (12 % изменения по амплитуде). Другими словами, от самого слабого шепота до самого оглушительного грома существует всего 120 уровней громкости, которые могут быть восприняты. Чувствительность уха удивительна; при прослушивании очень слабых звуков, амплитуда вибрации барабанной перепонки уха меньше, чем диаметр одной молекулы!

Грубо говоря, восприятие громкости соотносится с мощностью звука пропорционально 10 в степени 1/3. Например, если Вы увеличиваете мощность звука в *десять* раз, слушатели отметят, что громкость увеличилась примерно в *два* раза ($10^{1/3}$=2). Это основная проблема для устранения нежелательных звуков из окружающей среды, например, усиленного стерео в соседней квартире. Предположим, что Вы старательно покрываете 99 % вашей стены отличным звуконепроницаемым материалом, пропуская из-за дверей, углов, вентиляции и т.п., всего 1 % площади поверхности. Даже притом, что мощность звука была снижена исключительно до 1 % от ее прежнего значения, воспринимаемая громкость понизилась только приблизительно до $0{,}01^{1/3}$=0,2, или 20 %.

Обычно считают, что диапазоном слуха человека являются частоты от 20 Гц до 20 кГц, а наиболее чувствителен он к звукам на частотах между 1 кГц и 4 кГц. Например, слушатели могут обнаружить звуки на частоте 3 кГц настолько слабые как *0* dB УМЗ, но на 100 Гц потребуется 40 dB УМЗ (амплитуда увеличивается в 100 раз). При 3 кГц слушатели могут сказать, что два тона различны, если их частоты отличаются больше, чем примерно на 0,3%. При 100 герц это увеличивается до 3 %. Для сравнения, соседние клавиши на фортепьяно различаются примерно на 6 % по частоте.

Таблица 22.1

Вт/см^2	Децибел УМЗ	Пример звука
10^{-2}	140	Боль
10^{-3}	130	
10^{-4}	120	Дискомфорт
10^{-5}	110	Стук кузнечных молотков и рок-концерт
10^{-6}	100	
10^{-7}	90	OSHA ограничения для индустриального шума
10^{-8}	80	
10^{-9}	70	
10^{-10}	60	Нормальный разговор
10^{-11}	50	
10^{-12}	40	Самый слабый звук слышимый при 100 Гц
10^{-13}	30	
10^{-14}	20	Самый слабый звук слышимый при 10 кГц
10^{-15}	10	
10^{-16}	0	Самый слабый звук слышимый при 3 кГц
10^{-17}	-10	
10^{-18}	-20	

Главным преимуществом наличия *двух* ушей является способность определять *направление* на источник звука. Люди-слушатели могут определять разницу между двумя источниками звука, размещенными очень близко - всего в три градуса друг от друга на расстоянии в 10 метрах от человека. Эта информация о направлении получается двумя различными способами. Во-первых, частоты свыше приблизительно 1 кГц сильно *затеняются* головой. Другими словами, ближайшее к источнику звука ухо получает более сильный сигнал, чем ухо на противоположной стороне головы. Вторым ключом к направленности является то, что ухо на дальней стороне головы вследствие того, что оно находится на большем расстоянии от источника, слышит звук немного *позже*, чем

близлежащее ухо. Основываясь на типичном размере головы (приблизительно 22 см) и скорости звука в воздухе (приблизительно 340 метров в секунду), угловое различие в три градуса требует точности хронометрирования приблизительно 30 микросекунд. Так как такое хронометрирование требует использования принципа залпа, этот ключ к направленности используется преимущественно для звуков с частотами меньше чем приблизительно 1 кГц.

Обоим этим источникам информации о направлении сильно помогает способность повернуть голову и наблюдать происходящие изменения в сигналах. Интересное ощущение происходит тогда, когда точно такие же звуки подаются слушателю на оба уха так, как это происходит при прослушивании звукозаписи через наушники в режиме моно. Мозг заключает, что звук идет из центра головы слушателя!

Тогда как слух человека может определить *направление,* откуда приходит звук, он плохо справляется с определением *расстояния* до источника звука. Это происходит потому, что в звуковой волне мало доступных ключей, которые могут дать такую информацию. Слух человека слабо воспринимает то, что звуки высокой частоты находятся ближе, а звуки низкой частоты - дальше. Последнее связано с тем, что по мере распространения на большие расстояния высокие частоты звуковых волн рассеиваются. Другой слабый ключ к расстоянию, дающий восприятие размера помещения, содержит эхо. Например, звуки в большой аудитории будут содержать многократное эхо с интервалами около 100 миллисекунд, в то время как для небольшого офиса типичны интервалы в 10 миллисекунд. Некоторые существа решили проблему определения расстояния за счет использования активного *сонара*. Например, летучие мыши и дельфины производят щелчки и писки, которые отражаются от близко расположенных объектов. Измеряя интервал между испусканием звука и приходом его эха, эти животные могут определять местоположение объектов с разрешением около 1 сантиметра. Эксперименты показали, что некоторые люди, особенно слепые, также до некоторой степени могут использовать активное эхо для определения местоположения.

Тембр

Восприятие длительного звука, такого как нота идущая из музыкального инструмента, часто делится на три части: **громкость**, **высота** и **тембр**. Как описывалось ранее, *громкость* – это мера интенсивности звуковой волны. *Высота* – это частота основной составляющей в звуке, т.е. частота с которой повторяется форма волны. Несмотря на то, что в этих двух восприятиях имеются тонкие эффекты, они прямо соответствуют легко характеризуемым физическим величинам.

Тембр, будучи определяемым гармоническим составом сигнала, является более сложным. На рис. 22.2 иллюстрируются две формы волны, каждая из которых формируется сложением синусоидальной волны с *единичной* амплитудой и частотой 1 кГц с синусоидальной волной с амплитудой равной *одной второй* и частотой 3 кГц. Разница между двумя формами волн состоит в том, что одна из них, показанная на рис. 22.2b, содержит *проинвертированную* перед сложением высокочастотную составляющую. Говоря по-другому, третья гармоника (3 кГц) по сравнению с первой гармоникой (1 кГц) сдвинута по фазе на 180 градусов. Несмотря на очень разные во временной области формы волн эти два звуковых сигнала *идентичны*. Это связано с тем, что слух опирается на *амплитуды* частотных составляющих и очень нечувствителен к их *фазам*. *Очертания* форм волн временной области только косвенно относятся к слуху, и в звуковых системах обычно не рассматривается.

Невосприимчивость уха к фазе может быть понята через исследование того, как распространяется звук в среде. Предположим, что Вы слушаете человека, говорящего с Вами через небольшую комнату. Большая часть звуков, достигающих ваших ушей отражается от стен, потолка и пола. Поскольку характеристики распространения звука

(такие как: ослабление, отражение, и резонанс) зависят от частоты, различные частоты достигнут вашего уха по различным путям. Это означает, что по мере вашего продвижения по комнате относительная фаза каждой частоты будет изменяться. Поскольку ухо игнорирует эти изменения фазы, при изменении положения Вы воспринимаете голос как *не изменяющийся*. С физической точки зрения, фаза звукового сигнала по мере его распространения в сложной среде становится случайной. Говоря по-другому, ухо нечувствительно к фазе, поскольку она содержит мало полезной информации.

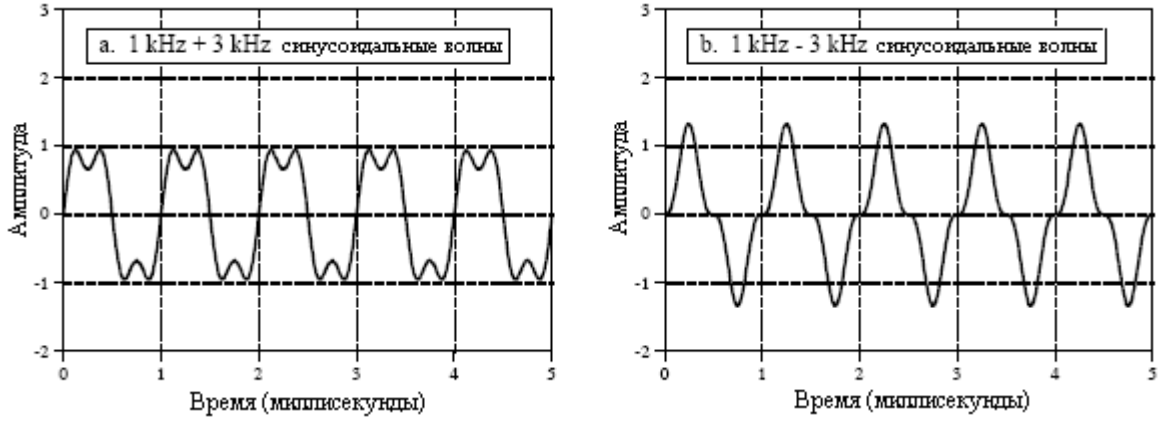


Рис. 22.2 К обнаружению фазы ухом человека

Однако нельзя говорить, что ухо полностью глухо к фазе. Это связано с тем, что фазовые изменения могут перестроить *временную ось* звукового сигнала. Примером является система чириканья (Глава 11), преобразующая импульс в сигнал значительно большей продолжительности. Хотя они отличаются только своими фазами, из-за их разницы в длительности ухо может различать эти два звука. По большому счету, это просто курьезы, а не то, что происходит в нормальной окружающей слушателя среде.

Предположим, что мы просим, скрипача сыграть, скажем, ноту *ля* находящуюся ниже ноты *до* в средине клавиатуры фортепиано. Когда форма волны демонстрируется на осциллографе, она во многом похожа на зубья пилы, показанные на рис. 22.3a. Это результат действия липкой канифоли, смазывающей волокна смычка скрипача. Поскольку смычок протягивается поперек струны, форма волны формируется следующим образом: струна прилипает к смычку, оттягивается им в сторону и в конечном итоге освобождается. Этот цикл повторяется снова и снова, создавая в результате пилообразную форму волны.

Рисунок 22.3b показывает, как этот звук воспринимается ухом, частота 220 герц плюс гармоники в 440, 660, 880 герц и т.д. Если бы эта нота была сыграна на другом инструменте, форма волны *выглядела* бы по-другому, однако ухо все еще слышало бы частоту в 220 герц плюс гармоники. Поскольку два инструмента дают одинаковую основную частоту для этой ноты, они звучат похоже, и говорят, что у них одинаковая *высота*. Так как относительная амплитуда *гармоник* разная, они не будут звучать идентично, и о них будут говорить, что у них разный *тембр*.

Часто говорят, что тембр определяется очертанием формы волны. Это справедливо, но слегка вводит в заблуждение. Восприятие тембра является результатом обнаружения ухом гармоник. Хотя содержание гармоник определяется очертанием формы волны, нечувствительность уха к фазе делает эту взаимосвязь весьма односторонней. То есть специфическая форма волны будет иметь только один тембр, в то время как специфический тембр имеет бесконечное число всевозможных форм волн.

Ухо слишком привыкло слышать основную частоту плюс гармоники. Если слушателям представить комбинацию синусоидальных волн с частотами 1 кГц и 3 кГц,

они будут говорить, что это звучит естественно и приятно. Если использовать синусоидальные волны с частотами 1 кГц и 3,1 кГц, это будет звучать неприятно.

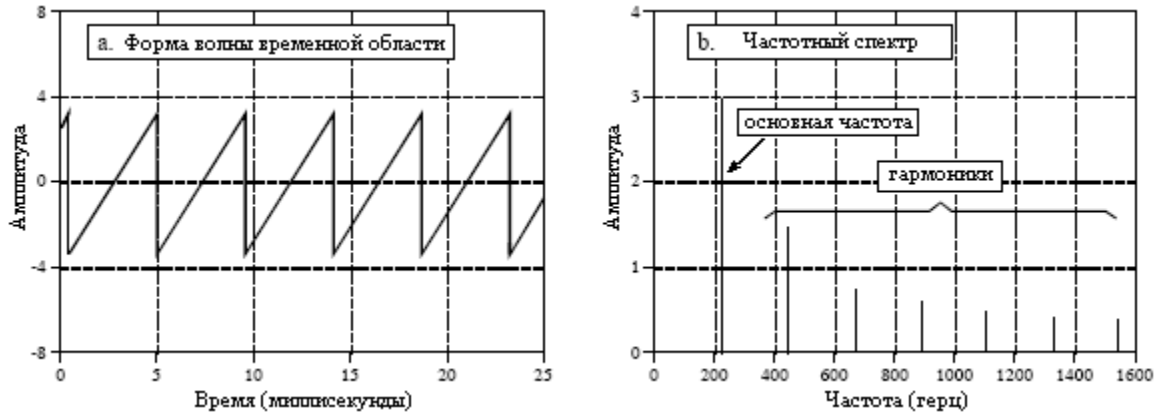


Рис. 22.3 Форма волны скрипки

Как показано на клавиатуре фортепиано на рис. 22.4, это является основой стандартной музыкальной шкалы. Нажатие самой дальней левой клавиши на пианино производит звук с основной частотой 27,5 герца плюс гармоники с частотами 55, 110, 220, 440, 880 герц и т.д. (здесь присутствуют также гармоники лежащие между этими частотами, но в данном обсуждении они не имеют значения). Эти гармоники соответствуют основным частотам, производимым другими клавишами на клавиатуре. Характерно, что каждая *седьмая* белая клавиша является гармоникой от самой дальней левой клавиши. То есть, восьмая клавиша слева имеет основную частоту 55 герц, 15-я клавиша имеет основную частоту 110 герц и т.д. Будучи гармониками друг друга, при игре эти клавиши звучат похоже и гармоничны при игре в унисон. По этой причине их *всех* называют нотой *ля*. Точно таким же образом белые клавиши находящиеся сразу же справа от каждой *ля* называются *си*, и все они являются гармониками друг друга. Этот порядок повторяется для семи нот: *ля*, *си*, *до*, *ре*, *ми*, *фа* и *соль*.

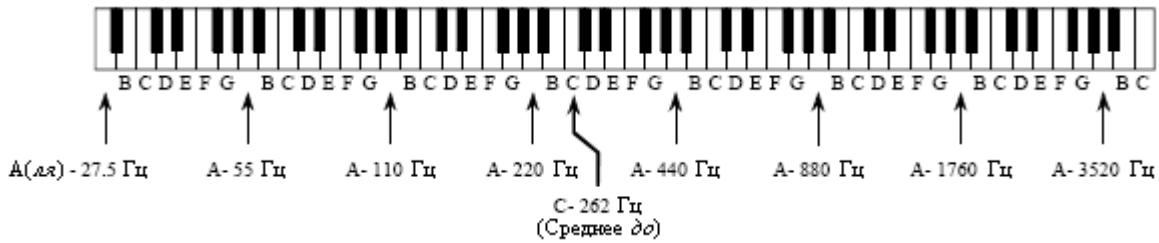


Рис. 22.4 Клавиатура фортепиано

Термин **октава** означает *коэффициент два по частоте*. На пианино одна октава содержит восемь белых клавиш, что объясняет ее название (*octo* по латыни *восемь*). Другими словами, после каждых семи белых клавиш частота пианино удваивается, а вся клавиатура охватывает немногим более семи октав. Диапазон слуха человека вообще ограничивается от 20 герц до 20 кГц, что соответствует около 1/2 октавы влево и две октавы вправо от клавиатуры пианино. Так как октавы основаны на удвоении частоты через каждое фиксированное число клавиш, то они являются *логарифмическим* представлением частоты. Это является важным потому, что звуковая информация в основном распределяется таким же образом. Например, сколько звуковой информации переносится в октаве между 50 герцами и 100 герцами, столько же и в октаве между 10 кГц и 20 кГц. Даже притом, что пианино охватывает всего около 20 % частот, которые

могут слышать люди (4 кГц из 20 кГц), оно может производить более 70 % звуковой информации, которую люди могут воспринимать (7 из 10 октав). Аналогично, самая высокая частота, которую может обнаружить человек в процессе его старения, снижается приблизительно от 20 кГц до 10 кГц. Однако это потери всего 10% слуховых способностей (одна октава из десяти). Как будет показано далее, это логарифмическое распределение информации непосредственно влияет на требуемую *частоту дискретизации* звуковых сигналов.

Качество звука против скорости передачи данных

Существуют два вопроса, на которые необходимо дать ответ при проектировании цифровой аудиосистемы: 1) насколько хорошо она должна звучать? и (2), какой может быть допустимая скорость передачи данных? Обычно ответ на эти вопросы попадает в одну из трех категорий. Прежде всего, **музыка с высоким качеством воспроизведения (Hi-Fi)**, где качество звука имеет первостепенную важность, и почти любая скорость передачи данных будет приемлема. Во-вторых, **телефонная связь**, требующая естественного звучания речи *и* низкой скорости передачи данных для снижения стоимости системы. В третьих, **сжатая речь**, где очень важно сокращение скорости передачи данных, и может допускаться некоторая неестественность в качестве звука. Это включает военную связь, сотовые телефоны, и речь для голосовой почты и средств мультимедиа, хранящаяся в цифровой форме.

Таблица 22.2

Требуемое качество звука	Ширина полосы	Частота дискретиз.	Число битов	Скор. перед. данных (бит/сек)	Комментарий
Музыка с высоким качеством воспроизведения (компакт диски)	5 Гц - 20 кГц	44,1 кГц	16 битов	706К	Удовлетворяет даже самых отявленных аудиофилов. Лучше чем человеческий слух.
Речь телефонного качества	200 Гц - 3,2 кГц	8 кГц	12 битов	96К	Хорошее качество для речи, но очень бедное для музыки.
(с компандированием)	200 Гц - 3,2 кГц	8 кГц	8 битов	64К	Нелинейное АЦП снижает скорость передачи данных на 50%. Очень типичная техника
Речь закодированная при помощи линейного прогно-зирующего кодирования	200 Гц - 3,2 кГц	8 кГц	12 битов	4К	ЦОС техника сжатия речи. Очень низкие скорости передачи данных, бедное качество звука

Таблица 22.2 показывает компромисс между качеством звука и скоростью передачи данных для этих трех категорий. Музыкальные системы с высоким качеством воспроизведения осуществляют дискретизацию достаточно быстро (44,1 кГц) и с достаточной точностью (16 битов), так что фактически они могут фиксировать все звуки, которые люди способны услышать. Такое великолепное качество звука дается ценой высокой скорости передачи данных 44,1 кГц × 16 битов = 706К бит/сек. Это - чистое решение проблемы "в лоб".

Тогда как музыка требует ширины диапазона в 20 кГц, естественно звучащая речь требует всего около 3,2 кГц. Даже притом, что частотный диапазон был сокращен всего до 16% (3,2 кГц из 20 кГц), сигнал все еще содержит 80% первоначальной звуковой информации (8 из 10 октав). Телекоммуникационные системы обычно работают с частотой дискретизации около 8 кГц, что допускает естественно звучащую речь, но

сильно снижает качество музыки. Вы вероятно уже знакомы с таким различием в качестве звука: ЧМ радиостанции передают с шириной полосы почти в 20 кГц, в то время как АМ радиостанции ограничены полосой приблизительно в 3,2 кГц. На АМ станции голоса звучат нормально, а музыка слабо и неудовлетворительно.

В только голосовых системах снижена также и точность - с 16 битов до 12 битов на отсчет со слегка заметными изменениями качества звука. Если размер шага квантования сделать неравномерным, то точность может быть снижена всего до 8 битов в отсчете. Это широко распространенная процедура, называемая **компандированием**, и будет обсуждена позже в этой главе. Частота дискретизации в 8 кГц с АЦП точностью 8 битов на отсчет, в результате дает скорость передачи данных 64К бит/сек. Для естественно звучащей речи это решение проблемы скорости передачи данных "*в лоб*". Заметьте, что речь требует менее 10% от скорости передачи данных для музыки с высоким качеством воспроизведения.

Скорость передачи данных 64К бит/сек является результатом прямого применения теории дискретизации и квантования к звуковым сигналам. Методы для дальнейшего снижения скорости передачи данных основываются на *сжатии* потоков данных за счет устранения присущей сигналам речи избыточности. Сжатие данных является темой Главы 27. Одним из наиболее эффективных способов сжатия звуковых сигналов является **линейное прогнозирующее кодирование (LPC)**, у которого есть несколько разновидностей и подгрупп. В зависимости от требуемого качества речи LPC может снизить скорость передачи данных до такой небольшой величины, как 2-6К бит/сек. Позже в этой главе мы вернемся к LPC в связи с *синтезом речи*.

Высококачественное воспроизведение звука

Аудиофилы требуют предельного качества звука, а все другие факторы рассматривают как второстепенные. Если Вам нужно описать какого образно, одним словом, то это будет - *убийственного*. Вместо того чтобы просто соответствовать способностям уха человека, такие системы разработаны с целью *выйти* за пределы возможностей слуха. Это единственный способ быть уверенным в том, что воспроизводимая музыка является нетронутой. Цифровой звук был представлен миру **лазерным компакт-диском**, или **CD**. Это была революция в музыке; качество звука CD систем сильно превосходило старые системы наподобие виниловых пластинок и магнитных пленок. На переднем крае этой технологии была ЦОС.

На рис. 22.5 показана поверхность лазерного компакт-диска в том виде, какой ее можно увидеть через мощный микроскоп. Основной является блестящая поверхность (отражающая свет) с цифровой информацией хранимой в виде последовательности темных питов (ямок – прим. перев.), подсвечиваемых на поверхности лазером. Информация расположена на одной дорожке спирально идущей, в противоположность дорожке фонографа, от внутренней стороны к внешней. По мере считывания информации от внутренней стороны спирали к внешней, частота вращения CD изменяется приблизительно от 480 до 210 оборотов в минуту, что делает скорость сканирования постоянной и равной 1,2 метра в секунду. (Для сравнения дорожка фонографа вращается с постоянной скоростью такой, как 33, 45 или 78 оборотов в минуту.) При воспроизведении оптический датчик определяет, отражает поверхность свет или не отражает, генерируя соответствующую двоичную информацию.

Как показано геометрией на рис. 22.5, CD хранит около 1 бита на микрометре квадратном, что соответствует 1 миллиону битов на миллиметре квадратном и 15 миллиардам битов на диске. Это почти такие же размерные характеристики, какие используются при производстве интегральных микросхем и с хорошим разрешением. Одним из свойств света является то, что он не может быть сфокусирован меньше, чем приблизительно половина длины волны или 0,3 микрометра. Поскольку оба, и

интегральные схемы и лазерные диски создаются оптическими средствами, нечеткость света ниже 0,3 микрометра ограничивает минимально достижимые характеристики, которые можно использовать.

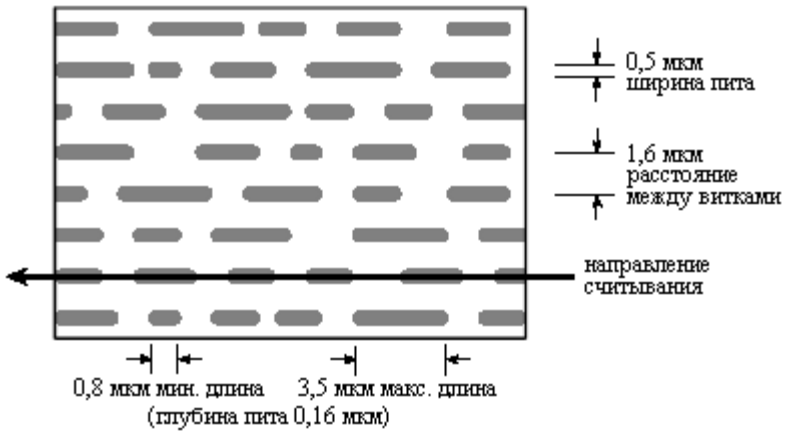


Рис. 22.5 Поверхность компакт диска

На рис. 22.6 показана блок схема типичной системы воспроизведения компакт-диска. Скорость передачи данных без их обработки – 4,3 миллиона битов в секунду, что соответствует 1 биту на каждые 0,28 мкм длины дорожки. Однако это вступает в противоречие с указанной геометрией CD; каждый пит должен быть не короче, чем 0,8 мкм и не длиннее, чем 3,5 мкм. Другими словами, каждая двоичная *единица* должна быть частью группы от 3 до 13 *единиц*. Это дает преимущества в сокращении уровня ошибок при оптическом считывании, но как же Вам заставить двоичные данные осуществлять такое странное группирование?

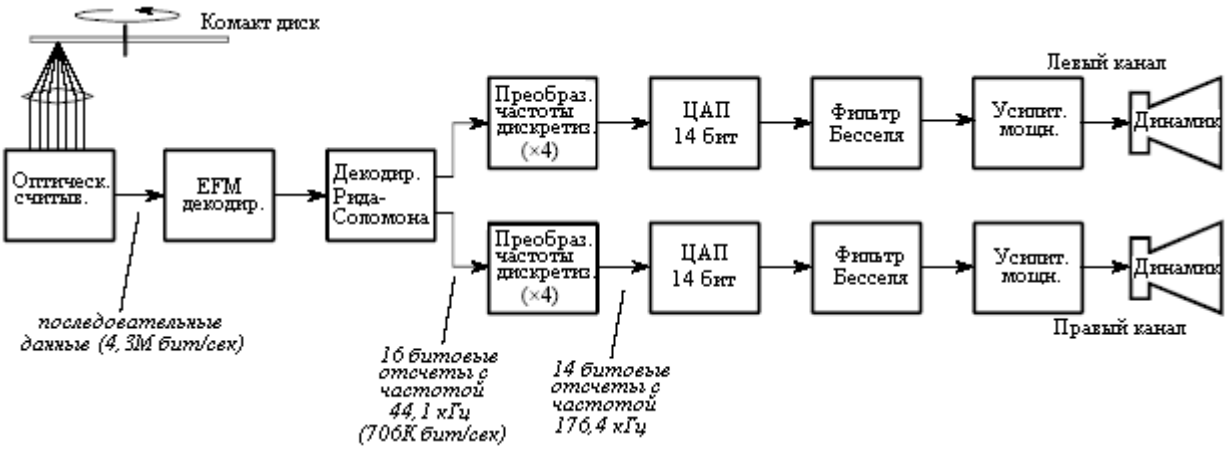


Рис. 22.6 Блок-схема проигрывателя компакт дисков

Ответом является схема кодирования называемая **модуляцией восемь в четырнадцать** (EFM). Вместо того чтобы непосредственно сохранить байт данных на диск, 8 битов пропускаются через поисковую таблицу, из которой выталкиваются 14 битов. Эти 14 битов обладают желаемыми характеристиками группирования и сохраняются на лазерном диске. После воспроизведения двоичные значения, считанные с диска, пропускают через инверсную поисковую таблицу EFM, в результате чего каждая 14 битовая группа преобразуется обратно в правильные 8 битов.

Вдобавок к EFM данные кодируются еще и в формате называемом **двухуровневый код Рида-Соломона**. Последнее включает в себя объединение левого и правого стерео

каналов вместе с данными для обнаружения и исправления ошибок. Цифровыми ошибками, обнаруживаемыми во время воспроизведения, являются: либо *исправляемые*, при использовании избыточных данных в схеме кодирования; либо *скрытые*, при интерполяции между соседними отсчетами; либо *статистические*, при установке нулевого значения отсчетов. Данные схемы кодирования в результате дают *утроенную* скорость передачи данных, т.е. 1,4М бит/сек для звуковых стерео сигналов по сравнению с 4,3М бит/сек, сохраняемых на диске.

После обнаружения и исправления ошибок звуковые сигналы представлены 16 битовыми отсчетами с частотой дискретизации 44,1 кГц. В наиболее простой системе эти сигналы могли бы быть пропущены через 16 битовый ЦАП за которым следует низкочастотный аналоговый фильтр. Однако это потребует высококачественной аналоговой электроники пропускающей все частоты ниже 20 кГц и заграждающей все частоты выше 22,05 кГц, 1/2 от частоты дискретизации. Более обычным способом является использование **мультичастотной** техники, т.е. преобразовать цифровые данные к более высокой частоте дискретизации до ЦАП. При преобразовании обычно используется коэффициент 4, преобразующий 44,1 кГц в 176,4 кГц. Это называется **интерполяцией** и может быть истолковано как двухэтапный процесс (хотя в действительности он может быт выполнен и не таким способом). Прежде всего, между исходными отсчетами размещаются по три отсчета с нулевым значением дающие более высокую частоту дискретизации. В частотной области это дает эффект дублирования спектра от 0 до 22,05 кГц три раза: от 22,05 кГц до 44,1 кГц, от 44,1 кГц до 66,15 кГц и от 66,15 кГц до 88,2 кГц. На втором этапе для удаления новых добавленных частот используется эффективный *цифровой* фильтр.

Увеличение частоты дискретизации делает интервал между отсчетами меньше, что дает более сглаженную кривую на выходе ЦАП. Сигнал продолжает содержать частоты лежащие между 20 Гц и 20 кГц; однако, частота Найквиста была увеличена в четыре раза. Последнее означает, что аналоговому фильтру необходимо пропустить только частоты, лежащие ниже 20 кГц, при блокировании частот лежащих выше 88,2 кГц. Это обычно выполняется трех полюсным фильтром Бесселя. Почему же используется фильтр *Бесселя,* если ухо не чувствительно к фазе? Убийственного, помните?

Поскольку отсчетов в четыре раза больше, число битов на отсчет может быть сокращено с 16 битов до 15 битов без потери качества звука. Коррекция *sin(x)/x*, необходимая для компенсации хранения нулевого порядка ЦАП, может быть частью либо аналогового, либо цифрового фильтра.

Звуковые системы более чем с одним каналом, называются **стерео** (от Греческого слова *пространственный*, или *трехмерный*). Множественные каналы посылают звук слушателю из различных направлений, обеспечивая более точное воспроизведение оригинальной музыки. Музыка проигрываемая через моно воспроизводящую систему (один канал) часто звучит искусственно и слабо. Для сравнения, хорошее стерео воспроизведение заставляет слушателя ощущать то, что музыканты находятся всего в нескольких шагах от него. С 1960-х годов для высококачественного воспроизведения музыки используются два канала (левый и правый), тогда как в кино используются четыре канала (левый, правый, центральный и окружающий). В ранних стерео записях (скажем Битлз или Мамас и папас) некоторых певцов можно часто услышать только в одном или в другом канале. Это быстро развилось в более искушенное **микширование**, когда звук от нескольких микрофонов в студии звукозаписи объединяется в два канала. Микширование является искусством, нацеленным на обеспечение слушателя восприятием эффекта *присутствия*.

Четырехканальный звук, используемый в кино называется Долби стерео, а версия для домашнего кинотеатра называется Долби сэррaунд про лоджик ("Dolby" и "Pro Logic" являются торговыми марками корпорации Dolby Laboratories Licensing). Четыре канала закодированы в стандартные левый и правый каналы, что позволяет воспроизводить

музыку и обычным двухканальным стерео системам. Для воссоздания четырехканального звука при воспроизведении используется Долби декодер. Левый и правый каналы, из громкоговорителей, расположенных с каждой из сторон киноэкрана или телевизионного экрана, аналогичны каналам обычной двух канальной стерео системы. Громкоговоритель центрального канала обычно расположен прямо над или под экраном. Его цель состоит в том, чтобы воспроизвести речь и другие визуально связанные звуки, прочно удерживая их в центре экрана в независимости от местонахождения зрителя/слушателя. Окружающие громкоговорители расположены слева и справа от слушателя и в большой аудитории могут включать до 20 громкоговорителей. Окружающий канал воспроизводит только средние частоты (скажем от 100 Гц до 7 кГц) и *задерживается* от 15 до 30 миллисекунд. Эта задержка заставляет слушателя воспринимать речь, как идущую с экрана, а не со стороны. То есть слушатель слышит речь идущую спереди, сопровождаемую задержанной версией речи идущей с разных сторон. Мозг слушателя интерпретирует задержанный сигнал, как отражение от стен, и игнорирует его.

Компандирование

В телекоммуникации важной является скорость передачи данных, поскольку она прямопропорциональна *стоимости* передачи сигнала. Экономия битов это - тот же самое, что и экономия денег. **Компандирование** является общим методом снижения скорости передачи данных звуковых сигналов посредством создания *неодинаковых* уровней квантования. Как упоминалось ранее, самый громкий звук, который можно стерпеть (120 dB УМЗ) приблизительно в миллион раз больше по амплитуде самого слабого звука, который можно услышать (0 dB УМЗ). Однако ухо не может различать звуки, отличающиеся друг от друга меньше чем примерно на 1 dB (12% по амплитуде). Другими словами, существует всего около 120 различных уровней громкости, которые можно различить, логарифмически расположенных вдоль диапазона амплитуды в один миллион.

Это является важным для оцифровки звуковых сигналов. Если уровни квантования расположены равномерно, то для получения речи телефонного качества должны быть использованы 12 битов. Однако если уровни квантования сделаны *неодинаковыми*, то для соответствия характеристикам слуха человека необходимо всего 8 битов. Это страшно интуитивно: если сигнал маленький, уровни должны быть очень близки друг к другу; если сигнал большой, может использоваться больший интервал.

Компандирование может быть выполнено тремя способами: (1) пропустить аналоговый сигнал через нелинейную цепь, перед тем как он достигнет линейного 8 битового АЦП, (2) использовать 8 битовый АЦП, имеющий внутри неравномерно расположенные уровни или (3) использовать линейный 12 битовый АЦП, сопровождаемый цифровой поисковой таблицей (12 битов на входе, 8 на выходе). Каждый из этих трех вариантов требует ту же самую нелинейность только в разных местах: в аналоговой цепи, в АЦП или в цифровой схеме.

Для компандирующих кривых используются два почти одинаковых стандарта: **закон μ255** (называемый также **законом мю**), применяемый в Северной Америке и **закон** "**A**", применяемый в Европе. Оба используют логарифмическую нелинейность, поскольку именно она преобразует промежуток, доступный человеческому уху в линейный интервал. Кривые, используемые в законе μ255 и законе "A" в форме уравнения, задаются как:

$$y = \frac{\ln(1+\mu x)}{\ln(1+\mu)} \qquad \text{для } 0 \le x \le 1, \quad \mu = 255 \tag{22.1}$$

$$y = \frac{1 + \ln(Ax)}{1 + \ln(A)} \qquad \text{для } 1/A \le x \le 1,$$

$A=87,6$ (22.2)

$$y = \frac{Ax}{1 + \ln(A)} \qquad \text{для } 0 \le x \le 1/A.$$

На рис. 22.7 эти уравнения изображены графически для значений входной переменной *x*, находящихся между –1 и +1 и дающих значения выходной переменной также между -1 и +1. Уравнениями 22.1 и 22.2 обработаны только положительные входные значения; части кривых для отрицательных входных значений найдены из симметрии. Как показано на рис. 22.7a, кривые для закона μ255 и закона "A" почти одинаковы. Единственное существенное различие находится вблизи начала координат, показанного на рис. 22.7b, где закон μ255 - плавная кривая, а закон "A" быстро переходит в прямую линию.

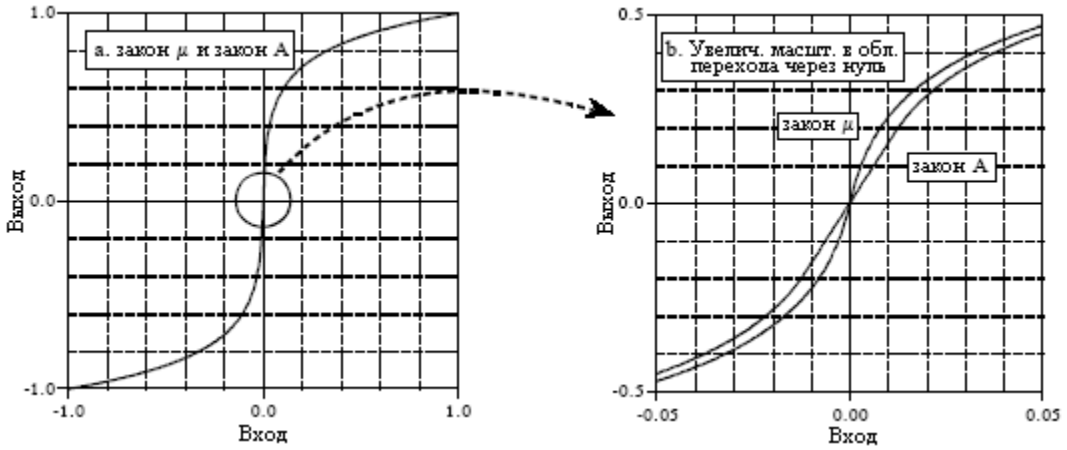


Рис. 22.7 Компандирующие кривые

Для аналоговой электроники создание устойчивой нелинейности является трудной задачей. Одним из методов является использование логарифмической зависимости между током и напряжением, приложенным к *p-n* переходу диода, и добавление цепей, корректирующих ужасный температурный дрейф. Большинство компандирующих схем используют другую стратегию: аппроксимировать нелинейность набором прямых линий. Типичной схемой является аппроксимация логарифмической кривой группой 16 прямых сегментов, называемых **кордами**. Первый бит 8 битового выходного сигнала показывает, положителен или отрицателен входной сигнал. Следующие три бита показывают, которая из 8 положительных и 8 отрицательных корд используется. Последние четыре бита разбивают каждую корду на 16 равноотстоящих приращений. Как и большинство интегральных схем, чипы компандирования имеют сложную и запатентованную внутреннюю конструкцию. Вместо того чтобы переживать о том, что происходит там, внутри чипа, обратите основное внимание на цоколевку выводов и техническое описание.

Синтез и распознавание речи

Компьютерное генерирование и распознавание речи являются грозными проблемами; было испробовано большое число подходов при достижении всего лишь

умеренного успеха. Это активная область исследований ЦОС и, несомненно, останется таковой на многие годы вперед. Вы будете очень разочарованы, если Вы ожидаете, что в этом разделе будет описано, как создавать цепи синтеза и распознания речи. Здесь будет представлено только краткое введение к типичным подходам. Прежде чем начать, следует отметить, что большинство коммерческих продуктов, которые воспроизводят речь голосом человека, не *синтезируют* ее, а просто проигрывают записанный в цифровом виде фрагмент речи человека. Этот подход обладает хорошим качеством звука, но ограничен предварительно записанными словами и фразами.

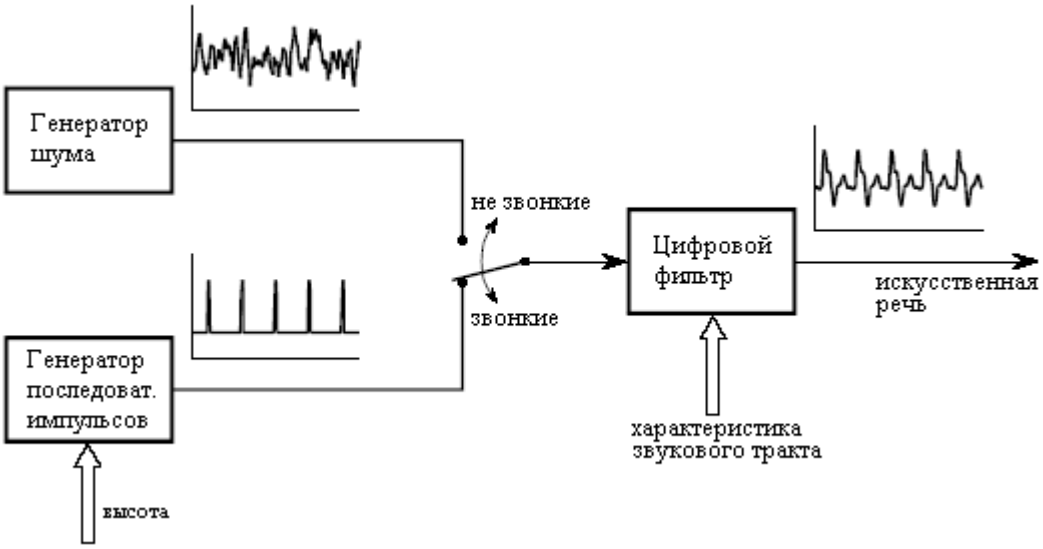


Рис. 22.8 Модель речи человека

Почти все методы синтеза и распознания речи базируются на модели формирования речи человека, показанной на рис. 22.8. Большинство звуков речи человека могут быть классифицированы либо как **звонкие**, либо как **фрикативные**. Звонкие звуки получаются тогда, когда воздух выдыхается из легких через голосовые связки и выходит изо рта и/или носа. Голосовые связки это две тонких складки ткани, вытянутые поперек воздушного потока, сразу за Адамовым яблоком (кадыком – прим. перев.). В ответ на изменяющееся напряжение мускул, голосовые связки вибрируют на частотах между 50 и 1000 Гц, создавая периодические попыхивания воздуха, вылетающие в горло. Примером звонких звуков являются гласные. На рис. 22.8 звонкие звуки представлены генератором последовательности импульсов, причем высота (то есть, основная частота формы волны) является регулируемым параметром.

Для сравнения, *фрикативные* звуки порождаются как произвольные шумы, не от вибрации голосовых связок. Это происходит тогда, когда воздушный поток почти перекрывается языком, губами, и/или зубами, создавая воздушную турбулентность вблизи сближения органов речи. Фрикативные звуки включают: *с*, *ф*, *ш*, *з*, *в* и *х*. В модели на рис. 22.8 фрикативные звуки представлены *генератором шума*.

Оба эти источника звука видоизменяются акустическими полостями, сформированными из языка, губ, рта, горла и носовых проходов. Поскольку распространение звука через эти структуры является линейным процессом, оно может быть представлено линейным фильтром с соответственно выбранным импульсным откликом. В большинстве случаев в модели используется *рекурсивный* фильтр с рекурсивными коэффициентами, определяющими характеристики фильтра. Из-за того, что акустические полости имеют размеры порядка нескольких сантиметров, частотная характеристика, прежде всего, представляет собой ряд резонансов в килогерцовом диапазоне. На жаргоне обработки звука, эти резонансные пики называются

формантными частотами. Изменяя относительное положение языка и губ, формантные частоты могут быть изменены и по частоте и по амплитуде.

На рис. 22.9 показан типичный способ изображения речевых сигналов, **голосовая спектрограмма** или **отпечаток голоса**. Звуковой сигнал разбивается на короткие сегменты, скажем от 2 до 40 миллисекунд, а затем для нахождения частотного спектра каждого сегмента используется БПФ. Эти спектры помещаются рядом друг с другом и преобразуются в черно-белое изображение с градациями серого (маленькие амплитуды становятся светлыми, а большие амплитуды становятся темными). Это дает графический способ наблюдения того, как изменяется во времени частотное содержимое речи. Выбираемая длина сегмента является компромиссом между *разрешением по частоте* (улучшается при удлинении сегмента) и *разрешением по времени* (улучшается при укорочении сегмента).

Как демонстрируется на примере буквы *a* в слове *rain*, во временной области звонкие звуки имеют периодическую форму волны, показанную на рис. 22.9a, а их частотный спектр представляет собой ряд систематически разделенных гармоник, показанных на рис. 22.9b. Для сравнения буква *s* в слове *storm* показывает, что фрикативные звуки во временной области имеют зашумленный сигнал, как на рис. 22.9c, и зашумленный спектр, изображенный на рис. 22.9d. Эти спектры показывают также конфигурацию формантных частот для обоих звуков. Заметим также, что изображение время - частота для слова *rain*, когда его произносят, оба раза выглядит схоже.

Спустя короткий период, скажем 25 миллисекунд, речевой сигнал может быть аппроксимирован определением трех параметров: (1) выбора возбуждения либо периодического, либо случайного шумового, (2) частоты периодической волны (если выбрано соответствующее возбуждение) и (3) коэффициентов цифрового фильтра, используемых для имитации характеристик звукового тракта. Затем, за счет непрерывного обновления этих трех параметров приблизительно 40 раз в секунду может синтезироваться непрерывная речь. Этому подходу обязан один из ранних коммерческих успехов ЦОС: “*Произноси и пиши*”, широко продаваемая электроника, помогающая обучению детей. Качество звука этого типа синтеза речи бедно, звучание слишком механическое и совсем не похоже на речь человека. Однако ему требуется очень низкая скорость передачи данных, обычно всего несколько Кбит/сек.

Этот подход является также основой для **линейного прогнозирующего кодирования** (LPC) - метода сжатия речи. Цифровая запись речи человека разбивается на короткие сегменты, и каждый из них характеризуется тремя параметрами в соответствии с моделью. Обычно для этого требуется около дюжины байтов на сегмент, или от 2 до 6 Кбайт/сек. По мере необходимости информация сегмента передается или записывается в память, а затем восстанавливается синтезатором речи.

Алгоритмы распознания речи идут на шаг дальше, пытаясь в извлеченных параметрах распознать образцы. Распознавание обычно включает сравнение информации сегмента с шаблонами предварительно записанных звуков, в попытке идентифицировать произнесенные слова. Проблема состоит в том, что этот метод работает не очень хорошо. Он полезен для некоторых приложений, но очень сильно уступает способностям человека слушателя. Для того чтобы понять, почему распознавание речи является для компьютеров таким сложным, представьте кого-то неожиданно говорящего следующее предложение:

Больший пробег медицинский покупайте собак почти удачный когда.

Конечно, Вы не поймете смысла этого предложения потому, что его здесь нет. Что более важно, Вы вероятно даже не поймете некоторые отдельные слова, которые были произнесены. Это основа к пониманию того, как люди воспринимают и понимают речь. Слова узнаются по их звучанию, а также из *контекста* предложения и из *ожиданий* слушателя. Например, представьте, что Вы слышите два предложения:

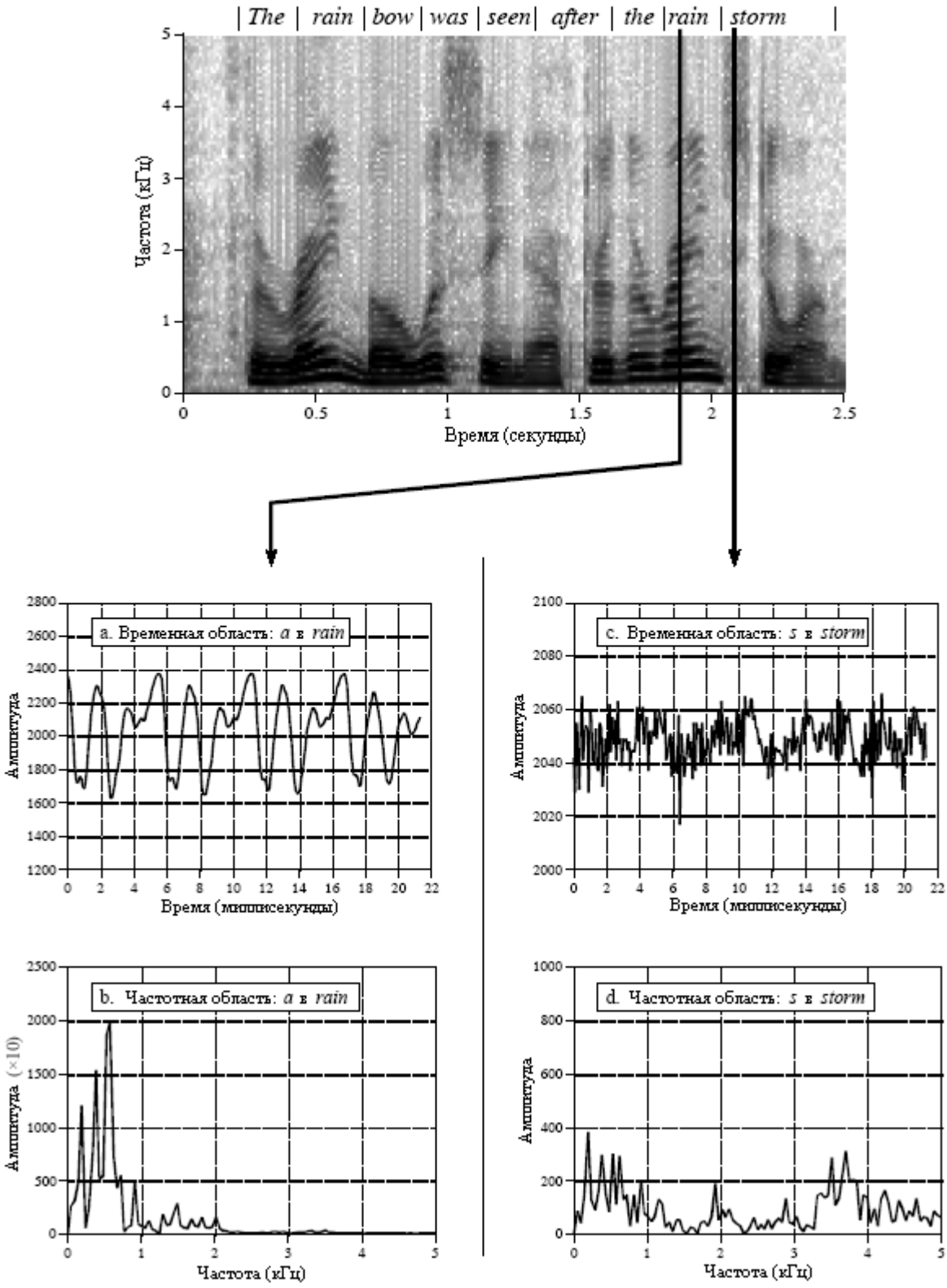


Рис. 22.9 Спектрограмма голоса

На хеллоуин ребенок носился с пионом.

Во время войны он был американским шпионом.

Даже если при передаче подчеркнутых слов были произведены точно те же самые звуки, из контекста слушатели *слышат* правильные слова. Из ваших накопленных знаний о мире Вы знаете, что дети не становятся секретными агентами, а люди в военное время не будут расхаживать с цветами. Обычно это не является сознательным актом, но представляет собой неотъемлемую часть слуха человека.

Большинство алгоритмов распознавания речи опираются только на звук отдельных слов, а не на их контекст. Они пытаются *распознавать слова*, а не *понимать речь*. Это является их огромным недостатком по сравнению с человеком-слушателем. Системам распознавания речи присущи три досадных недостатка: (1) распознаваемая речь должна иметь между словами отчетливые паузы. Это устраняет необходимость иметь в алгоритме дело с фразами, которые звучат похоже, но составлены из разных слов (т.е. *с пионом* и *шпионом*). Для людей, привыкших говорить непрерывным потоком, это медленно и неудобно. (2) Часто словарь ограничивается всего несколькими сотнями слов. Это означает, что алгоритму, для того чтобы найти лучшее соответствие приходится просматривать только ограниченный набор слов. При увеличении словаря увеличивается и время распознавания, и процент ошибок. (3) Алгоритм следует *обучать* для каждого говорящего. Данное требует от всякого, кто использует систему, произносить каждое слово до его распознавания, часто на это требуется от пяти до десяти повторов. Такая персонифицированная база данных сильно увеличивает точность распознания слова, но она неудобна и требует много времени.

Награда за развитие успешной технологии распознания речи огромна. Речь является самым быстрым и наиболее эффективным способом человеческого общения. Распознание речи обладает потенциалом замены письма, печатания, ввода с клавиатуры и электронного управления, выполняемого при помощи выключателей и кнопок. Чтобы быть принятой коммерческим рынком, ей требуется работать чуть-чуть лучше. Прогресс в распознавании речи, вероятно, придет из областей искусственного интеллекта и нейронных сетей, а также и непосредственно из ЦОС. Не думайте об этом как о технических *трудностях*; думайте об этом как о технических *возможностях*.

Нелинейная обработка звука

Цифровая фильтрация различными способами может улучшить звуковые сигналы. Например, для отделения частот представляющих главным образом сигнал, от частот представляющих главным образом шум может быть использована *Винеровская фильтрация* (смотрите Главу 17). Подобным образом, *обратная свертка* может скомпенсировать нежелательную *свертку*, так как это делается при восстановлении старых записей (тоже обсуждалось в Главе 17). Эти виды линейных методов составляют костяк ЦОС. Для обработки звука полезными являются также и несколько *нелинейных* методов. Здесь будут кратко описаны два.

Первый нелинейный метод используется для уменьшения в речевых сигналах широкополосного шума. Этот вид шума включает: шипение магнитной ленты, электронный шум в аналоговых цепях, ветер, дующий в микрофон, приветствующие крики толпы и т.д. Линейная фильтрация здесь слабо полезна, поскольку частоты в шуме полностью перекрывают частоты в голосовом сигнале, оба покрывают диапазон от 200 герц до 3,2 кГц. Как могут быть разделены два сигнала, когда они перекрываются и во временной области, *и* в частотной области?

Вот как это делается. В коротком сегменте речи, амплитуды частотных составляющих сильно *неодинаковые*. В качестве примера, на рис. 22.10а показан

частотный спектр16 миллисекундного сегмента речи (т.е. 128 отсчетов при частоте дискретизации 8 кГц). Наибольшая часть сигнала содержится здесь в нескольких частотах с большими амплитудами. Напротив, на рис. 22.10b иллюстрируется спектр только случайного шума; он очень нерегулярный, но более однородно распределен в области низких амплитуд.

Ну а теперь ключевая концепция: если присутствуют и сигнал, и шум, то, глядя на *амплитуду* каждой из частот, оба они могут быть частично разделены. Если амплитуда большая, то вероятнее всего это сигнал, и поэтому должен быть сохранен. Если амплитуда маленькая, то это в значительной степени можно отнести к шуму, и поэтому его следует отбросить, т.е. свести к нулю. Частотные составляющие среднего размера прилаживаются между этими двумя крайностями каким-нибудь сглаживающим способом.

Другой способ, взглянуть на этот метод, как на *изменяющийся во времени фильтр Винера*. Как Вы помните, частотная характеристика фильтра Винера пропускает частоты, которые являются главным образом сигналом, и подавляет частоты, которые являются главным образом шумом. Это требует *заранее* знания спектров сигнала и шума, так, чтобы можно было определить частотную характеристику фильтра. Эта нелинейная техника использует ту же самую идею, за исключением того, что частотная характеристика фильтра Винера пересчитывается для каждого сегмента, базируясь на спектре *этого сегмента*. Другими словами, частотная характеристика фильтра изменяется от сегмента к сегменту в соответствии с характеристиками самого сигнала.

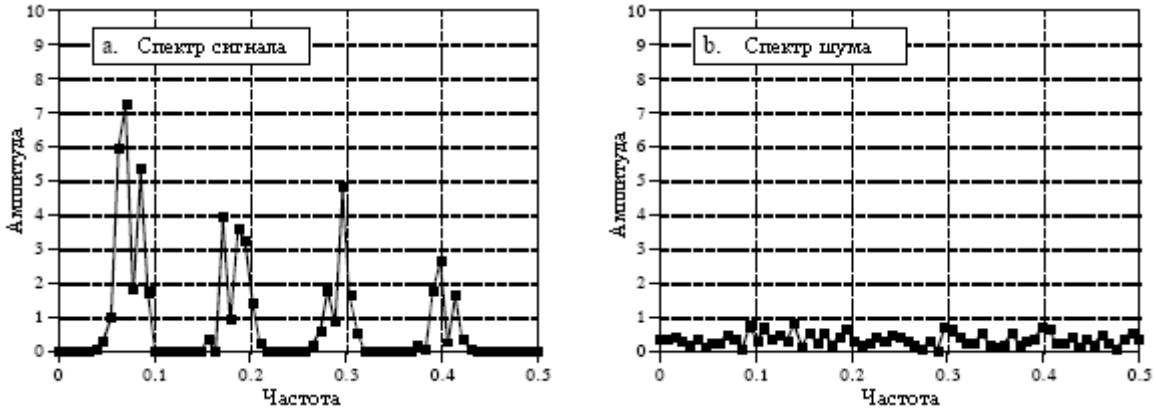


Рис. 22.10 Спектр речи и шума

Одной из трудностей в реализации этого (и других) нелинейного метода является то, что метод суммирования перекрытий для фильтрации длинных сигналов не является справедливым. Поскольку частотная характеристика изменяется, форма волны во временной области каждого сегмента, больше не будет лежать на одном уровне в соседних сегментах. Это может быть преодолено, если вспомнить, что звуковая информация кодируется в изменяющихся во времени частотных образцах, а не в конфигурации формы волны временной области. Типичный подход состоит в том, чтобы разделить первоначальный сигнал временной области на *перекрывающиеся* сегменты. После обработки, к каждому из перекрывающихся сегментов, прежде чем они будут повторно объединены, применяется сглаживающее окно. Это обеспечивает плавный переход частотного спектра от одного сегмента к следующему.

Вторая нелинейная техника называется **гомоморфной** обработкой сигнала. Этот термин буквально означает: *такая же структура*. Сложение - это не единственный способ, которым шум и помехи могут быть объединены с интересующим Вас сигналом; умножение и свертка также являются обычными средствами смешивания сигналов вместе. Если сигналы объединяются нелинейным способом (т.е. каким-либо другим способом кроме сложения), они не могут быть разделены при помощи линейной фильтрации.

Гомоморфные методы пытаются разделить сигналы, объединенные нелинейным способом, заставляя проблему *становиться* линейной. То есть проблема преобразуется к *такой же структуре*, как и линейная система.

Например, рассмотрим звуковой сигнал, передаваемый посредством амплитудно-модулированной радио волны. По мере изменения атмосферных условий амплитуда принимаемого сигнала увеличивается и уменьшается, приводя к медленному изменению принимаемого звукового сигнала во времени. Это может быть смоделировано как *перемножение* звукового сигнала, представленного как *a*[ ], с медленно изменяющимся сигналом *g*[ ], представляющим изменяющееся усиление. Данная задача обычно обрабатывается электронной схемой называемой *автоматической регулировкой усиления* (АРУ), но она может быть также скорректирована и нелинейной ЦОС.

Как показано на рис. 22.11, входной сигнал *a*[ ] × *g*[ ] пропускается через логарифмическую функцию. В соответствии с тождеством $\log(x \cdot y) = \log x + \log y$ это приводит к тому, что два сигнала объединяются при помощи сложения, т.е. $\log a[\,] + \log g[\,]$. Другими словами, *логарифм* является гомоморфным преобразованием преобразующим нелинейную задачу *умножения* в линейную задачу *сложения*.

Затем, сложенные сигналы разделяются обычным линейным фильтром, то есть некоторые частоты пропускаются, в то время как другие задерживаются. Для АРУ сигнал усиления *g*[ ] будет состоять из очень низких частот, намного ниже полосы от 200 герц до 3,2 кГц для звуковых частот. Логарифм этих сигналов будет иметь более сложные спектры, но идея та же самая: фильтр верхних частот используется для того, чтобы исключить из сигнала изменяющиеся составляющие усиления. Действительно, $\log a[\,] + \log g[\,]$ преобразуется в $\log a[\,]$. На последнем этапе при помощи показательной функции (антилогарифм, или e^x) избавляются от логарифма, получая желаемый выходной сигнал *a*[ ].

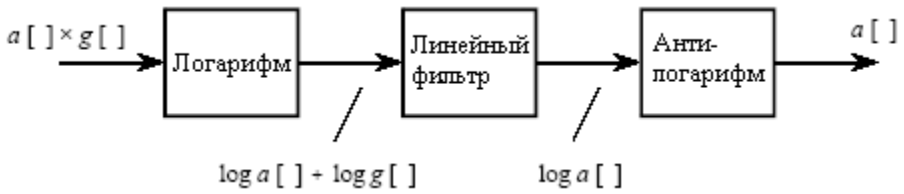


Рис. 22.11 Гомоморфное разделение перемноженных сигналов

На рис. 22.12 показана гомоморфная система для разделения сигналов, которые были объединены *сверткой*. Приложением, где это оказалось полезным, является устранение эха из звуковых сигналов. То есть производится свертка звукового сигнала с импульсной характеристикой состоящей из дельта функции плюс сдвинутой и смасштабированной дельта функции. Гомоморфное преобразование для свертки состоит из двух этапов, преобразующего свертку в умножение, *преобразования Фурье*, сопровождаемого *логарифмом*, превращающим умножение в сложение. Как и ранее, сигналы затем разделяются при помощи линейной фильтрации, после чего избавляются от гомоморфного преобразования.

Интересным поворотом на рис. 22.12 является то, что линейная фильтрация имеет дело с сигналами частотной области в той же манере, в которой обычно обрабатываются сигналы временной области. Другими словами, временная и частотная области обменялись своим естественным применением. Например, если для выполнения этапа линейной фильтрации использовалась БПФ свертка, то перемножаемые "спектры" появились бы во *временной области*. Этот обмен ролями породил странный жаргон. Например, *кепстр* (трансформировано из *спектр*) представляет собой преобразование Фурье логарифма преобразования Фурье. Подобным образом имеются фильтры с *длинной полосой пропускания* и *короткой полосой пропускания*, вместо фильтров с нижней

полосой пропускания и верхней полосой пропускания. Некоторые авторы даже используют термины *сачтотный аланиз* и *лифтрация*.

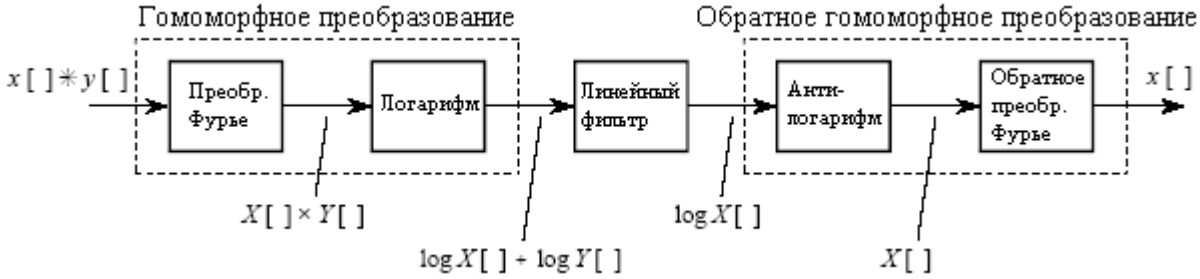


Рис. 22.12 Гомоморфное разделение сигналов объединенных сверткой

Помните, что это лишь упрощенные описания утонченных алгоритмов ЦОС; гомоморфная обработка полна тонких деталей. Например, логарифм должен уметь обрабатывать, как отрицательные, так и положительные значения во входном сигнале, поскольку они характерны для звуковых сигналов. Это требует применения *комплексного логарифма* более продвинутой концепции, чем логарифм, используемый в каждодневной науке и технике. Когда линейная фильтрация ограничивается только фильтром *нулевой фазы*, комплексный логарифм находится взятием простого логарифма от абсолютного значения сигнала. После прохождения через фильтр нулевой фазы, отфильтрованному сигналу снова приписывается знак исходного сигнала.

Другой проблемой является *совмещение имен*, которое происходит тогда, когда берется логарифм. Например, представьте оцифровку непрерывной *синусоидальной волны*. В соответствии с теоремой о дискретизации, на один период достаточно двух или более отсчетов. Теперь рассмотрим оцифровку логарифма этой непрерывной синусоидальной волны. Острые углы, чтобы захватить форму волны, т.е. предотвратить совмещение имен, требуют на период намного больше отсчетов. Требуемая частота дискретизации после применения логарифма может легко стать в 100 раз большей, чем прежде. В дальнейшем, применяется ли логарифм к непрерывному сигналу или к его цифровому представлению значения не имеет; результат будет тот же самый. Совмещение имен не прекратится, если частота дискретизации не будет достаточно высокой, чтобы захватить острые углы, возникшие из-за нелинейности. В результате звуковые сигналы, вместо лишь стандартных 8 кГц, могут потребовать дискретизации при 100 кГц или более.

Даже если эти детали решены, нет никакой гарантии, что линеаризованные сигналы *смогут* быть разделены линейным фильтром. Это в силу того, что спектры линеаризованных сигналов могут перекрываться, даже если не перекрываются спектры исходных сигналов. Например, представьте сложение двух синусоидальных волн, одну при 1 кГц и одну при 2 кГц. Так как эти сигналы в частотной области не перекрываются, они могут быть полностью разделены линейной фильтрацией. Теперь представьте, что эти две синусоидальных волны перемножены. Используя гомоморфную обработку, берется логарифм объединенного сигнала, что дает логарифм одной синусоидальной волны плюс логарифм другой синусоидальной волны. Проблема состоит в том, что логарифм синусоидальной волны содержит множество гармоник. Поскольку гармоники от этих двух сигналов накладываются друг на друга, их полное разделение не возможно.

Несмотря на эти препятствия, гомоморфная обработка преподносит важный урок: сигналы следует обрабатывать в манере *совместимой* с тем, как они образованы. Говоря по-другому, первый шаг в любой задаче ЦОС - понять, каким образом в обрабатываемых сигналах представлена информация.

ГЛАВА
23

ФОРМИРОВАНИЕ И ДЕМОНСТРАЦИЯ ИЗОБРАЖЕНИЯ

Изображения являются описанием того, как параметр изменяется по поверхности. Например, обычные видимые изображения являются результатом изменений интенсивности света по двумерной плоскости. Однако свет не является единственным параметром, используемым в научных изображениях. Например, изображение может быть сформировано из *температуры* интегральной схемы, *скорости крови* в артерии пациента, *рентгеновского излучения* от отдаленной галактики, *движения почвы* во время землетрясения и т.д. Такие экзотические изображения обычно преобразуются в обычные картинки (т.е. световые изображения), так чтобы они могли быть оценены глазом человека. Данная первая глава по обработке изображений описывает, как формируются и преподносятся человеку-наблюдателю цифровые изображения.

Структура цифрового изображения

На рис. 23.1 иллюстрируется структура цифрового изображения. Данный пример изображения планеты Венера получен микроволновым радаром от орбитального космического зонда. Микроволновое изображение требуется потому, что плотная атмосфера не пропускает видимый свет, делая обычную фотографию невозможной. Показанное изображение представлено *40000* отсчетов организованных в двумерный массив *200* столбцов на *200* строк. Так же, как и в случае одномерных сигналов, эти строки и столбцы могут быть пронумерованы от *0* до *199* или от *1* до *200*. На жаргоне обработки изображений каждый отсчет называется **пиксель**, сокращение от фразы: picture element (*элемент картинки)*. Каждый *пиксель* в данном примере представляет собой отдельное число между *0* и *255*. При получении изображения Венеры это число относилось к количеству энергии микроволн, отраженному от соответствующей местности на поверхности планеты. Для того чтобы отобразить все это в форме *видимого изображения* значение каждого пикселя преобразуется по **шкале серого**, где *0* является черным, *255* является белым, а промежуточные значения представляют собой оттенки серого.

В изображениях информация закодирована в **пространственной области**, для изображения это эквивалент временной области. Другими словами, характеристики изображений представляются не *синусоидами*, а *контурами*. Это означает, что число пикселей и расстояние между ними определены тем, насколько маленькие характерные черты должны быть увидены, а не формальными ограничениями теоремы о дискретизации. В изображениях *может* происходить совмещение имен, но к нему в основном относятся, как к раздражающему фактору, нежели чем как к главной проблеме. Например, костюм в тонкую полоску по телевидению выглядит ужасно, поскольку частота повторения рисунка выше, чем частота Найквиста. Частоты, у которых произошло совмещение имен, появляются в виде светлых и темных полос, перемещающихся по одежде при изменении положения человека (это явление называется *муар* – прим. перев.).

"Типичное" цифровое изображение состоит примерно из *500* строк на *500* столбцов. Это изображение с качеством, встречающимся на телевидении, в приложениях

для персональных компьютеров и общенаучных исследованиях. Изображения с меньшим количеством пикселей, скажем *250* на *250*, рассматриваются как имеющие необычно слабое разрешение. Часто оно характерно для новых видов изображений; по мере же становления технологии, добавляется большее число пикселей. Изображения с таким низким разрешением зримо выглядят неестественно, и часто можно видеть отдельные пиксели. На другом конце находятся изображения с более чем *1000* на *1000* пикселей, считающиеся исключительно хорошими. Это качество лучшей компьютерной графики, телевидения высокой четкости и 35 миллиметровых кинофильмов. Существуют также приложения, нуждающиеся в еще более высоком разрешении, где требуется несколько тысяч пикселей на одну сторону: цифровые рентгеновские изображения, космические фотографии и глянцевая реклама в журналах.

Сильнейшей мотивацией к использованию изображений с низким разрешением является то, что здесь нужно обрабатывать *меньшее число* пикселей. Это не является тривиальным; одной из наиболее трудных проблем в обработке изображений является управление массивным количеством данных. Например, одна секунда цифрового звука требует около восьми *килобайт*. Для сравнения, одна секунда телевидения требует около восьми *мегабайт*. Передача изображения 500 на 500 пикселей с помощью модема со скоростью с 33,6 килобита в секунду требует почти минуту! Переход к размеру изображения в 1000 на 1000 *учетверяет* эти проблемы.

В обработке изображения обычно используется 256 **уровней серого** (уровней квантования), что соответствует одному байту на пиксель. Для этого существует несколько причин. Во-первых, отдельный байт удобен для управления данными, ведь именно так обычно данные хранятся в компьютере. Во-вторых, большое число пикселей в изображении компенсирует до некоторой степени ограниченное число уровней квантования. Например, представьте группу смежных пикселей чередующихся по значению между цифровыми числами 145 и 146. Глаз человека воспринимает эту область, как область с яркостью 145,5. Другими словами, изображение очень *раскачено*. В-третьих, и это наиболее важно, размер шага яркости в 1/256 (0,39%) раза меньше, чем может воспринимать глаз. Изображение, представленное человеку-наблюдателю при использовании большего числа, чем 256 уровней, улучшено не будет.

Однако некоторые изображения должны быть сохранены с большим числом битов, чем 8 на пиксель. Помните, что большинство изображений, которые встречаются в ЦОС, представляют собой невидимые параметры. Для надлежащего охвата тонких деталей сигнала полученное изображение должно быть в состоянии использовать преимущества большего количества уровней квантования. Здесь важно не ждать того, что глаз человека увидит всю информацию, содержащуюся в этих точно отстоящих друг от друга уровнях. Мы рассмотрим вопросы, касающиеся этой проблемы в последующем обсуждении яркости и контрастности.

Значение каждого пикселя в цифровом изображении представляет небольшую *область* непрерывного изображения, которое было оцифровано. Например, представьте, что орбитальный зонд Венеры по мере его продвижения над планетой по орбите, снимает отсчеты каждые 10 метров по поверхности планеты. Это определяет квадратный **интервал между отсчетами** и **сетку дискретизации,** в которой каждый пиксель представлен площадкой 10 на 10 метров. Теперь, представьте, что происходит в отдельном измерении микроволнового отражения. Космический зонд излучает высоко сфокусированный всплеск микроволновой энергии, ударяющийся о поверхность, представляющую собой, например, круг диаметром 15 метров. Таким образом, каждый пиксель содержит информацию о таком круглом участке вне зависимости от размера сетки дискретизации.

Такая область непрерывного изображения, которая формирует значение пикселя, называется **апертурой дискретизации**. Размер апертуры дискретизации часто связан с внутренними возможностями конкретной используемой системы изображения. Например,

микроскопы ограничены качеством оптики и длиной волны света, электронные камеры ограничены случайной диффузией электронов в видео датчике и т.д. В большинстве случаев сетка дискретизации делается приблизительно такой же, как и апертура дискретизации системы. Разрешение конечного цифрового изображения будет ограничиваться, прежде всего, наибольшим из двух: сетки дискретизации или апертуры дискретизации. Мы вернемся к этой теме в Главе 25 при обсуждении пространственного разрешения цифровых изображений.

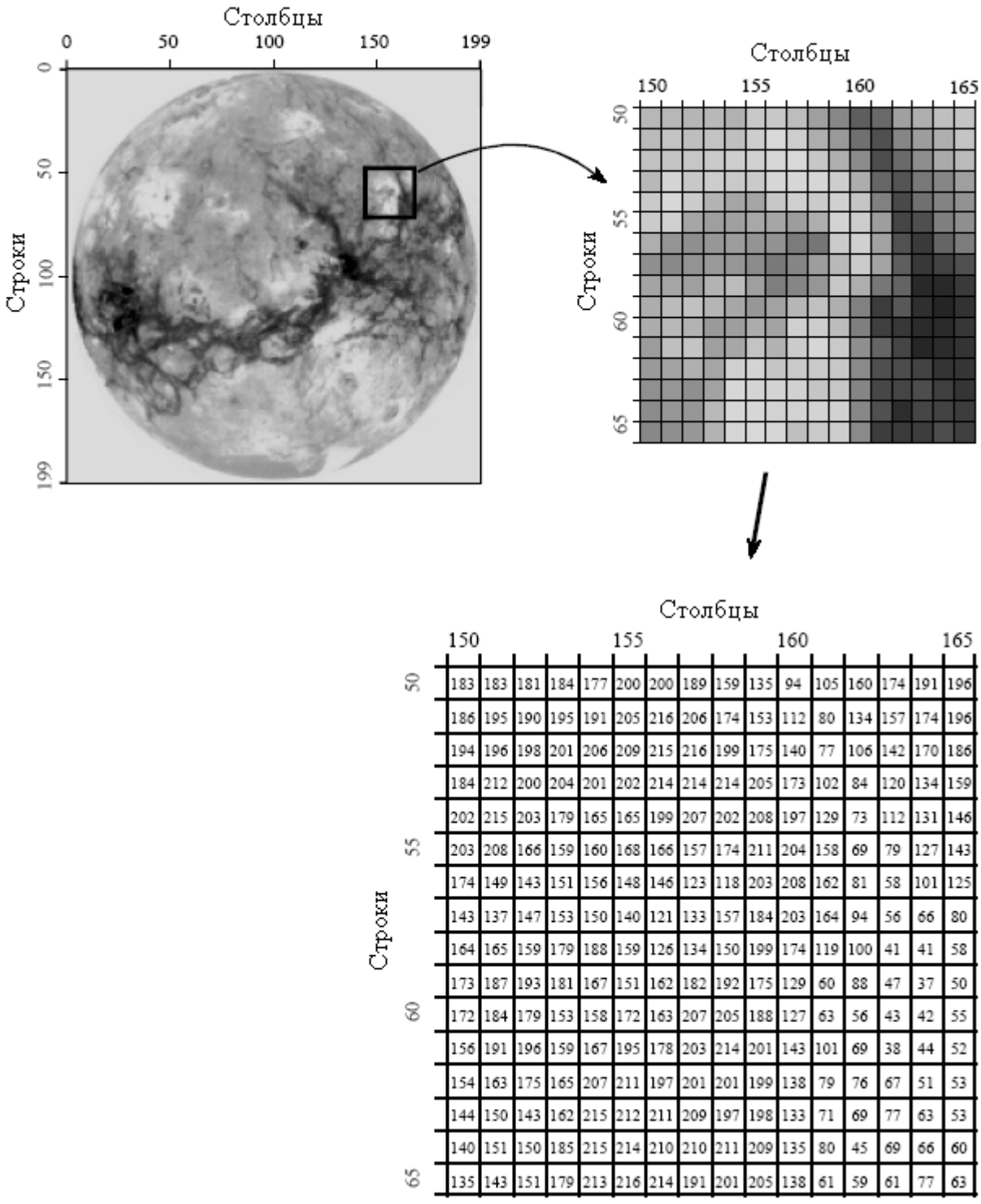


Рис. 23.1 Структура цифрового изображения

Цвет добавляется к цифровым изображениям за счет использования для каждого пикселя трех чисел, представляющих интенсивность трех первичных цветов: красного, зеленого и синего. Смешивание этих трех цветов дает все возможные цвета, которые может воспринять глаз человека. Для хранения интенсивности каждого из этих трех цветов часто используется отдельный байт, что позволяет уловить в изображении 256×256×256 = 16,8 миллионов различных цветов.

Когда цель заключается в том, чтобы представить зрителю реальную картину мира, наподобие как в телевидении или в фотографии, цвет является очень важным. Однако в науке и технике изображения используются не для этого. Здесь целью является анализ двумерного сигнала посредством использования в качестве *инструмента* визуальной системы человека. Для этого достаточно черно-белого изображения.

Камеры и глаза

Строение и работа глаза очень похожа на электронную камеру, поэтому вполне естественно обсуждать их вместе. Оба базируются на двух основных компонентах: набор линз и датчик изображения. Набор линз улавливает часть света исходящего от объекта и фокусирует его на датчике изображения. Затем датчик изображения преобразует световой рисунок в видео сигнал либо электронный, либо нейронный.

На рис. 23.2 показана работа линзы. В этом примере изображение фигуристки сфокусировано на экран. Термин *фокус* означает, что существует взаимно однозначное соответствие каждой точки на фигуристке с соответствующей точкой на экране. Например, рассмотрим область размером *1* мм × *1* мм на кончике пальца носка. При ярком свете эту площадь размером *1* мм^2 каждую секунду бомбардирует, грубо, около 100 триллионов фотонов света. В зависимости от характеристик поверхности от 1 до 99 процентов этих падающих фотонов будут отражены в произвольном направлении. Через линзу пройдет только небольшая часть этих отраженных фотонов. Например, через линзу диаметром один сантиметр расположенную от объекта на расстоянии 3 метров пройдет всего одна миллионная часть отраженного света.

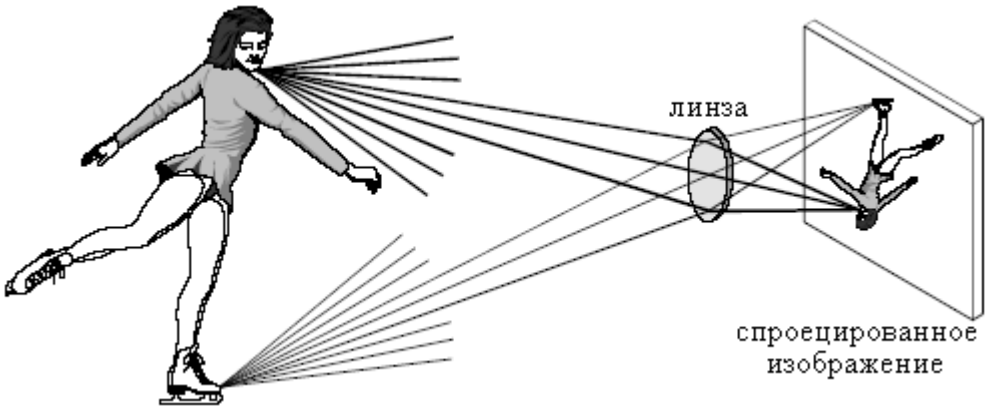


Рис. 23.2 Фокусировка света линзой

Преломление в линзе изменяет направление индивидуальных фотонов в зависимости от места и угла, где они столкнулись с переходом воздух стекло. Такое изменение направления заставляет расходящийся из одной точки свет возвратиться в одну точку на проекционном экране. Все фотоны, отразившиеся от носка и прошедшие через линзу, возвращаются вместе обратно на "носок" в спроецированном изображении. Подобным образом часть света, пришедшая от *любой* другой точки на объекте, будет проходить через линзу, и фокусироваться на соответствующую точку в спроецированном изображении.

На рис. 23.3 и 23.4 иллюстрируются соответственно основное устройство электронной камеры и глаза человека. Оба представляют собой светонепроницаемые корпуса с прикрепленной на одном конце линзой и датчиком изображения на другом. Камера заполнена воздухом, в то время как глаз заполнен прозрачной жидкостью. Каждая система линз имеет два регулируемых параметра **фокус** и **диаметр диафрагмы**.

Если линза сфокусирована не должным образом, каждая точка на объекте будет проецироваться в круглый участок на датчике изображения, что делает изображение расплывчатым. В камере фокусировка достигается физическим перемещением линзы вперед или назад от датчика изображения. Для сравнения, глаз содержит две линзы выпуклость в передней части глазного яблока, называемая роговой оболочкой, и регулируемая линза внутри глаза. Роговая оболочка осуществляет основное преломление света, ее форма и местоположение не изменяются. Регулирование фокусировки выполняется внутренней линзой, гибкой структурой, которая может деформироваться под действием *ресничных мышц*. Когда эти мышцы сокращаются, линза искривляется, создавая резкую фокусировку объекта.

В обеих системах для управления тем, какая часть линзы открыта для прохождения света, а следовательно, и тем, какова яркость изображения проецируемого на датчик изображения, используется *диафрагма*. Диафрагма глаза сформирована из непрозрачной мышечной ткани, которая для увеличения *зрачка* (увеличения количества света) может сокращаться. Диафрагма в камере - это набор механических элементов, выполняющий такие же функции.

В оптических системах параметры взаимодействуют друг с другом многими неожиданными способами. Например, рассмотрим, как влияют на *четкость* полученного изображения количество доступного света и чувствительность датчика света. Это влияние связано с тем, что *диаметр диафрагмы* и *время выдержки* регулируют передачу соответствующего количества света от наблюдаемой сцены к датчику изображения. Если света доступно больше чем необходимо, диаметр диафрагмы должен быть уменьшен, что дает большую *глубину резкости* (граница расстояния от камеры, в пределах которого объект остается в фокусе). Большая глубина резкости обеспечивает более четкое изображение в тех случаях, когда объекты находятся на различных расстояниях. С другой стороны, обилие света позволяет сократить время выдержки, что приводит к уменьшению смазанности изображения от тряски камеры и движения снимаемого объекта. Оптические системы полны взаимообменов такого вида.

Регулируемая диафрагма необходима как в камере, так и в глазе, поскольку в окружающей среде диапазон интенсивностей света значительно больше, чем непосредственно может обработать световой датчик. Например, разница интенсивности света между солнечным светом и лунным светом составляет около одного миллиона. Добавим к этому, что отражающая способность может изменяться между *1 %* и *99 %*, в результате диапазон интенсивности света почти в *сто миллионов*.

Динамический диапазон электронной камеры, определяемый как наибольший сигнал, который может быть измерен, деленный на внутренний шум прибора, обычно составляет от 300 до 1000. С другой стороны, максимально вырабатываемый сигнал равен 1 вольту, а среднеквадратический шум в темноте около 1 милливольта. Отверстие диафрагмы объективов типичных камер изменяется приблизительно в 300 раз. Это дает типичную электронную камеру с динамическим диапазоном в несколько сотен тысяч. Совершенно очевидно, что та же камера и объектив используемые при ярком свете в темную ночь будут бесполезны.

Для сравнения глаз работает с динамическим диапазоном, который почти перекрывает большинство изменений окружающей среды. Удивительно, что диафрагма не является основным способом достижения такого огромного динамического диапазона. От темноты до света отверстие зрачка изменяется всего примерно в 20 раз. Чтобы иметь возможность работать, нервные клетки обнаруживающие свет постепенно меняют свою

чувствительность, сохраняя динамический диапазон. Например, после того как Вы вошли в темный зал кинотеатра для того, чтобы приспособиться к низкой освещенности, вашим глазам понадобится всего несколько минут.

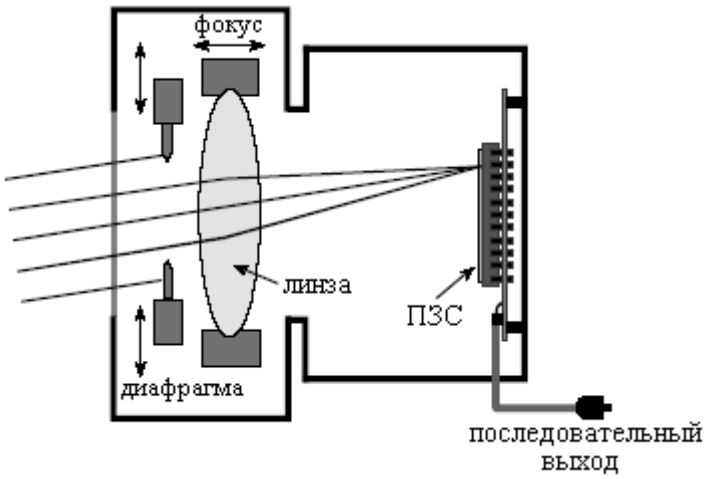


Рис. 23.3 Схема электронной камеры

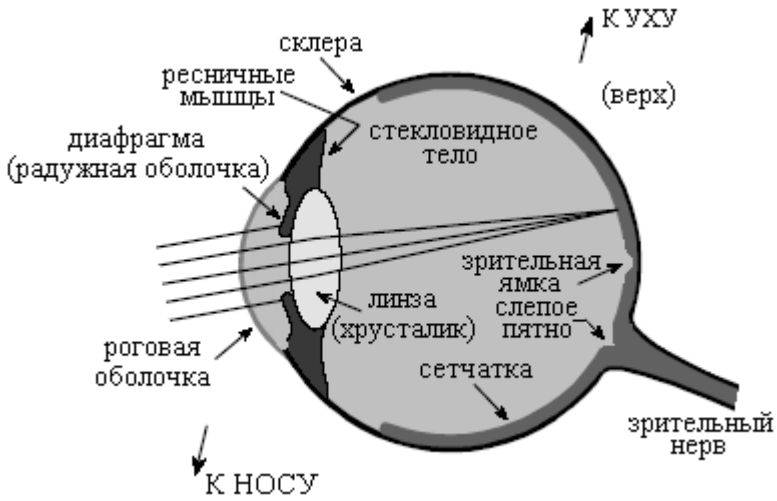


Рис. 23.4 Схема глаза человека

Один способ, которым ЦОС может улучшать изображения при помощи снижения динамического диапазона наблюдателя, следует рассмотреть. То есть, мы не желаем очень светлых и очень темных участков в одном и том же изображении. Отраженное изображение сформировано из двух сигналов: двумерного образа *освещенности* сцены, умноженного на двумерный образ *отражающей способности* сцены. Образ отражающей способности имеет динамический диапазон менее 100, поскольку все обычные материалы отражают от *1 %* до *99 %* падающего света. Именно здесь содержится большая часть *информации изображения*, такой как: где в сцене расположены объекты и каковы характеристики их поверхности. Для сравнения, сигнал освещенности зависит от источников света находящихся вокруг объектов, но не от самих объектов. Сигнал освещенности может иметь динамический диапазон порядка нескольких миллионов, хотя наиболее типично от 10 до 100 в пределах отдельного изображения. Сигнал освещенности несет мало интересной информации, но может ухудшить конечное изображение из-за увеличения его динамического диапазона. ЦОС может улучшить эту ситуацию с помощью подавления сигнала освещенности, позволяя сигналу отражающей способности

доминировать в изображении. Подход к осуществлению этого алгоритма представляется в следующей главе.

Светочувствительная поверхность, покрывающая тыльную сторону глаза, называется **сетчаткой**. Как показано на рис. 23.5, сетчатка может быть разделена на три основных слоя специализированных нервных клеток: один для преобразования света в нервные сигналы, один для обработки изображения и один для передачи информации к ведущему в мозг зрительному нерву. Почти у всех животных эти слои, как кажется, расположены *наоборот*. То есть, светочувствительные клетки расположены в последнем слое, что требует прохождения света через другие слои до момента его обнаружения.

Существует два типа клеток, обнаруживающих свет: **палочки** и **колбочки**, названные так по их внешнему виду под микроскопом. Палочки специализируются на работе с очень слабым светом таким, как при ночном небе. Вблизи темноты видимость кажется очень *зашумленной*, то есть изображение кажется, заполнено непрерывно меняющимся зернистым рисунком. Это является результатом слабого сигнала изображения, а не результатом недостатков глаза. В глаз попадает так мало света, что можно увидеть случайное обнаружение отдельных фотонов. Это называется *статистическим шумом* и встречается во всех системах получения изображений при слабом свете, наподобие военных систем ночного видения. Глава 25 еще раз вернется к этой теме. Так как палочки не могут обнаруживать цвет, видимость при слабом свете является черно-белой.

Рецепторы колбочек специализируются на распознавании цвета, но могут работать только при достаточном количестве света. В глазу имеются три типа колбочек: чувствительные к красному цвету, чувствительные к зеленому цвету и чувствительные к синему цвету. Это следствие того, что они содержат различные *зрительные пигменты*, химические вещества, поглощающие разные длинны волн (цвета) света. На рис. 23.6 показаны длинны волн света, которые возбуждают каждый из этих трех рецепторов. Это называется **RGB** (акроним от Red – красный, Green - зеленый и Blue – синий – прим. перев.) **кодированием,** и представляет то, в каком виде информация покидает глаз через зрительный нерв. Благодаря нейронной обработке на нижних уровнях мозга, человеческое восприятие цвета устроено сложнее. RGB кодирование преобразуется в другую схему кодирования, в которой цвета классифицируются как: красный *или* зеленый, синий *или* желтый и светлый *или* темный.

RGB кодирование – важное ограничение человеческого зрения; длинны волн, существующие в окружающей среде, объединяются всего в три широкие категории. Для сравнения, специализированные камеры могут разделять оптический спектр на сотни или тысячи индивидуальных цветов. Например, они могли бы быть использованы для классификации клеток на злокачественные и здоровые, понимания физики далекой звезды или для того чтобы увидеть замаскированных солдат, прячущихся в лесу. Почему при обнаружении цвета глаз так ограничен? Очевидно, что все, что необходимо людям для выживания - это найти *красное* яблоко среди *зеленых* листьев на фоне *синего* неба.

Грубо, палочки и колбочки шириной *3* мкм плотно упакованы по всей поверхности сетчатки *3* см на *3* см. В результате сетчатка составлена из массива, грубо, $10000 \times 10000 = 100$ миллионов рецепторов. Для сравнения, зрительный нерв имеет всего около одного миллиона нервных волокон соединенных с этими клетками. В среднем каждое волокно зрительного нерва через соединительный слой соединено, грубо, со 100 световыми рецепторами. В дополнение к объединению информации, соединительный слой улучшает изображение, заостряя края и подавляя составляющую освещенности сцены. Эта биологическая обработка изображения будет обсуждена в следующей главе.

Прямо в центре сетчатки находится небольшая область называемая **зрительной ямкой** (от латинского *fovea* - яма), которая используется для видения с высоким разрешением (см. рис. 23.4). Зрительная ямка отличается от остальной части сетчатки по нескольким аспектам. Во-первых, зрительный нерв и соединительные слои отодвинуты в

сторону, что позволяет расположить рецепторы прямо под падающим светом. В результате чего зрительная ямка выглядит как маленькая впадина на сетчатке. Во-вторых, в зрительной ямке расположены только колбочки, и они упакованы более плотно, чем в остальной части сетчатки. Такое отсутствие палочек в зрительной ямке объясняет, почему ночное зрение часто лучше тогда, когда смотришь на объект *искоса*, а не прямо. В-третьих, каждое волокно зрительного нерва находится под влиянием всего нескольких колбочек, обеспечивая хорошую способность локализации. Зрительная ямка удивительно мала. На нормальном для чтения расстоянии зрительная ямка видит площади около 1 мм в диаметре меньше чем размер одной буквы! Разрешающая способность эквивалентна сетке *20×20* пикселей в пределах этой области.

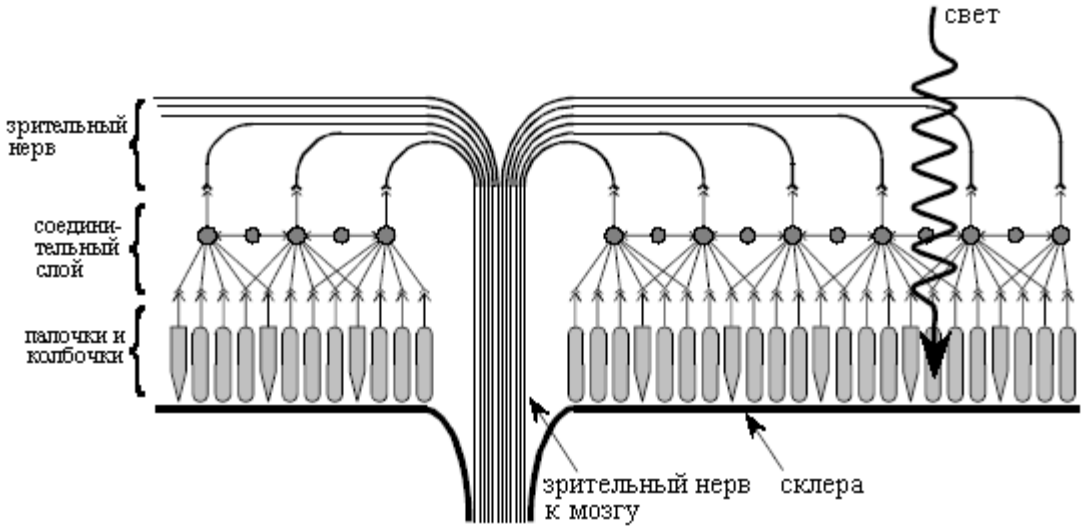


Рис. 23.5 Сетчатка глаза человека

С помощью резких движений глаза, называемых **саккадированными**, человеческое зрение преодолевает ограничения вызванные маленьким размером зрительной ямки. Эти отрывистые движения позволяют обладающей высоким разрешением зрительной ямке быстро сканировать рассматриваемую область в поисках подходящей информации. Кроме того, саккадированние обеспечивает палочкам и колбочкам непрерывное изменение светового рисунка. Это важно по причине естественной способности адаптации сетчатки к изменению уровня интенсивности света. Действительно, если глаз принудительно зафиксировать на одной и той же сцене, детали и цвета через несколько секунд потускнеют.

Наиболее обычными датчиками изображения, используемыми в электронных камерах, являются **приборы с зарядовой связью** (**ПЗС**). ПЗС представляет собой интегральную схему, которая в 1980-х вытеснила большинство камер на вакуумных трубках, точно так же, как за двадцать лет до этого транзисторы вытеснили вакуумные усилительные лампы. Сердцем ПЗС является тонкая пластина из кремния (подложка – прим. перев.) площадью обычно около 1 см^2. Как показано на поперечном разрезе, на рис. 23.7, нижняя сторона пластины покрыта тонким слоем метала, соединенного с потенциалом земли. Верхняя сторона покрыта тонким электрическим изолятором и периодически повторяющимися электродами. Наиболее обычным типом ПЗС является прибор с **трехфазным считыванием**, в котором каждые третьи из электродов соединяются вместе. Используемый кремний носит название *p-типа*, что означает, что у него избыток носителей положительного заряда называемых *дырками*. В этом обсуждении о дырке можно думать, как о положительно заряженной частице, свободно перемещающейся по полупроводнику. На этом рисунке дырки представлены символом "+".

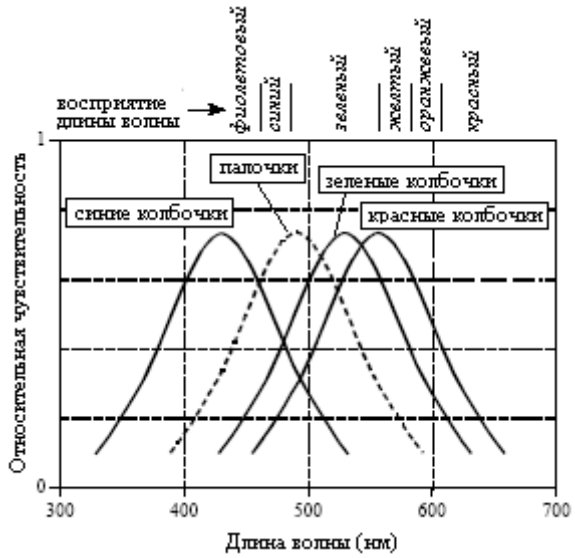


Рис. 23.6 Спектральная характеристика глаза

На рис. 23.7a к одной из трех фаз приложен потенциал *+10* вольт, в то время как две другие фазы удерживаются в *0* вольт. Поскольку положительный потенциал отталкивает положительные заряды, это заставляет дырки двигаться прочь от каждого третьего электрода. Последнее приводит к формированию области под этими электродами называемой **ямой**, сокращенная версия от физического термина *потенциальная яма*.

Каждая яма в ПЗС представляет собой эффективный датчик света. Как показано на рис. 23.7b, энергия одного фотона света, попадающего на кремний, затрачивается на формирование двух заряженных частиц одного электрона и одной дырки. Дырка уходит прочь, оставляя в яме застрявший электрон, удерживаемый положительным напряжением на электроде. На этой иллюстрации электроны представлены символом "-". Во время **периода экспонирования** световой рисунок, падающий на ПЗС, преобразуется в рисунок зарядов внутри ям в ПЗС. Источникам более тусклого света требуются более длительные периоды экспонирования. Например, период экспонирования для стандартного телевидения 1/60-я секунды (в стандарте NTSC – прим. перев.), тогда как астрофотография может аккумулировать свет в течение многих часов.

Считывание электронного изображения очень хитроумно; аккумулированные в каждой яме электроны *выталкиваются* на выходной усилитель. Как показано на рис. 23.7c, положительное напряжение подается на линии двух других фаз. Это приводит к расширению каждой ямы вправо. Как показано на рис. 23.7d, следующим шагом является снятие напряжения с первой фазы, что приводит к сжатию ям. Что перемещает аккумулированные электроны в яму, находящуюся справа от того места, где они находились ранее. Повторяя с тремя линиями фаз эту пульсирующую последовательность, аккумулированные электроны выталкиваются вправо, пока не достигнут **чувствительного усилителя заряда**. Это - причудливое название для конденсатора, за которым следует буфер с единичным коэффициентом усиления. Как только электроны вытолкнуты из последней ямы, они попадают на конденсатор, на котором они создают падение напряжения. Для достижения высокой чувствительности, конденсаторы выполняются чрезвычайно маленькими, обычно менее *1* пф. Такой конденсатор и усилитель являются частью интегральной ПЗС и выполняются на том же кристалле кремния. Сигнал, выходящий из ПЗС, представляет собой последовательность уровней напряжения пропорциональных количеству света, упавшему на последовательно расположенные ямы.

На рис. 23.8 показано, как с ПЗС считывается двумерное изображение. По завершении периода экспонирования, аккумулированный в каждой яме заряд, каждый раз перемещается вверх по столбу на одну строку. Например, все ямы в 15 строке сначала перемещаются в строку 14, затем в строку 13, затем в строку 12 и т.д. Каждый раз строки смещаются на одну позицию вверх, а все ямы из строки с номером *1* передаются в **горизонтальный регистр**. Горизонтальный регистр представляет собой специализированную группу ям для быстрого перемещения заряда в горизонтальном направлении к чувствительному усилителю заряда.

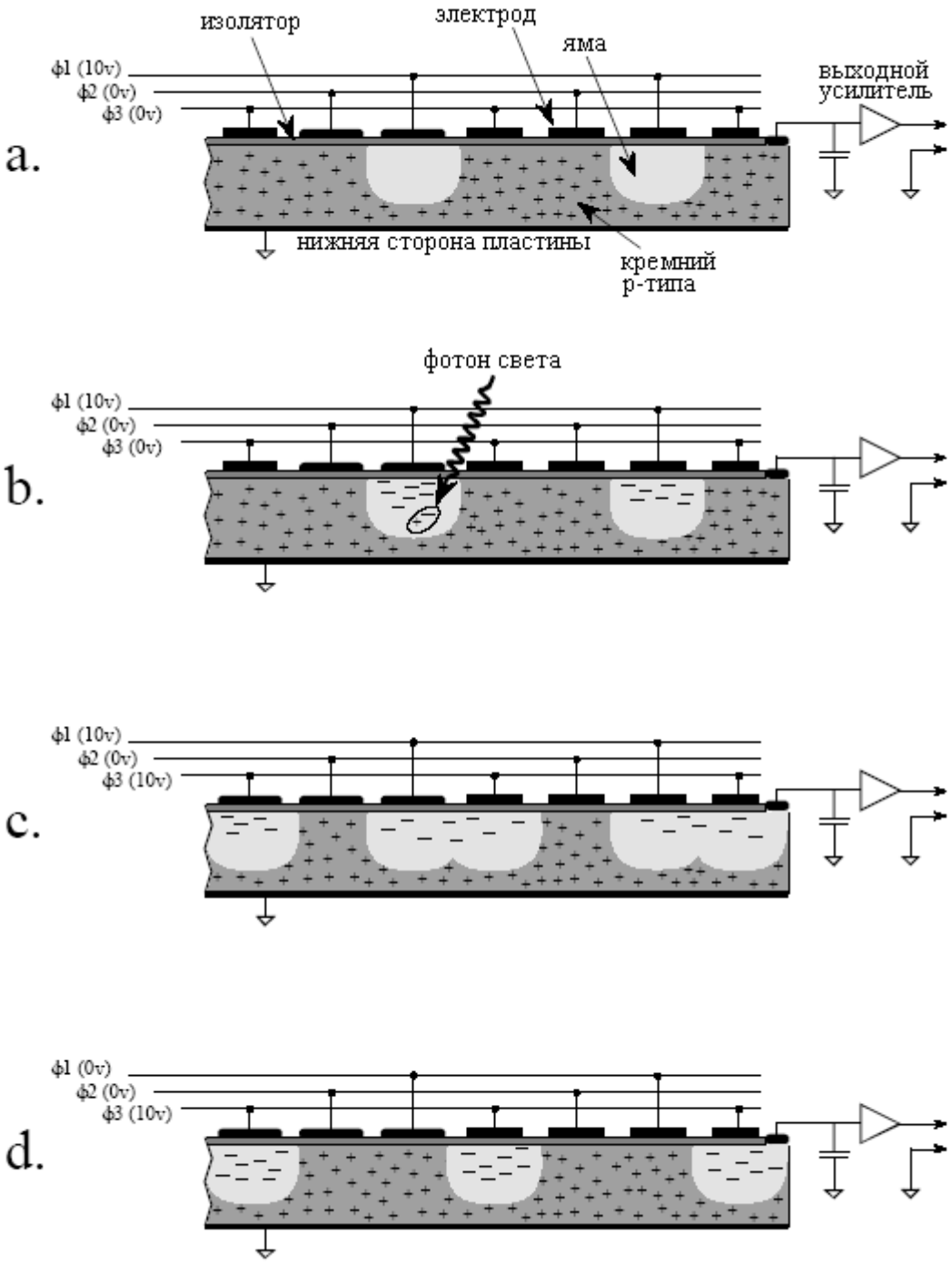


Рис. 23.7 Работа прибора с зарядовой связью (ПЗС)

Заметьте, что такая архитектура преобразует двумерный массив в последовательный поток данных в особом порядке. Первым считываемым пикселем

является пиксель в левом верхнем углу изображения. Затем слева направо продолжается считывание первой строки, а далее слева направо продолжается считывание последующих строк. Настоящее почти всегда соблюдается при преобразовании двумерного массива (изображения) в последовательные данные и называется **рядом старшего порядка**.

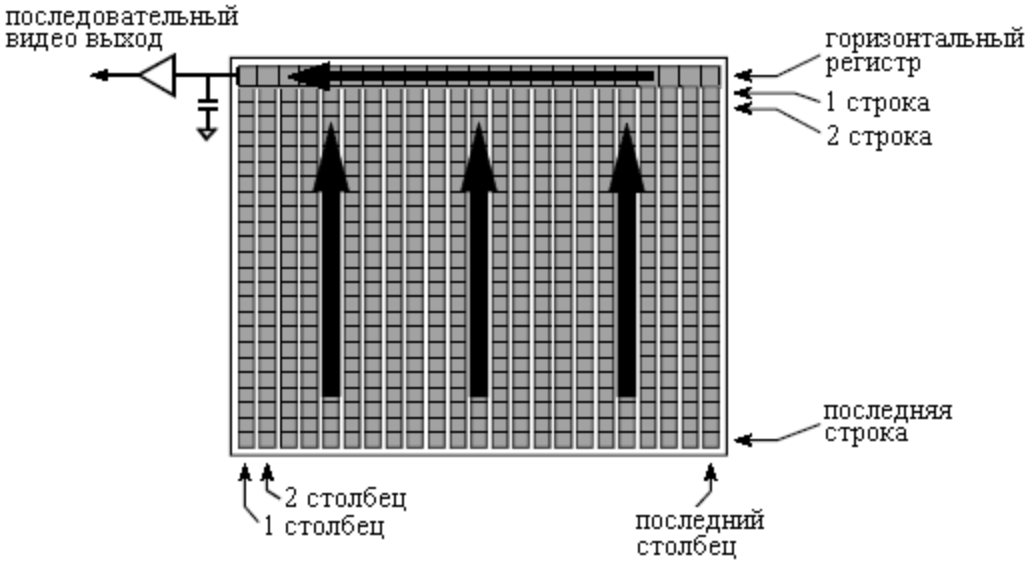


Рис. 23.8 Архитектура ПЗС

Телевизионные видео сигналы

Уже почти более 50 лет стандартный телевизионный сигнал все еще остается одним из наиболее обычных способов передачи изображения. На рис. 23.9 показано, как выглядит телевизионный сигнал на осциллографе. Это называется **композитным видео сигналом**, что означает, что импульсы вертикальной и горизонтальной синхронизации (развертки) смешены с информацией действительной картинки. Эти импульсы используются в телевизионных приемниках для синхронизации цепей отклонения по вертикали и горизонтали для достижения соответствия показываемому видео. Каждая секунда стандартного видео содержит 30 полных изображений, обычно называемых **кадрами**. Видео инженеры сказали бы, что каждый кадр содержит 525 строк. Это число слегка вводит в заблуждение, поскольку видео информацию содержит всего от 480 до 486 этих строк; оставшиеся от 39 до 45 строк зарезервированы для синхроимпульсов, удерживающих телевизионные цепи в синхронизме с видео сигналом.

В стандартном телевидении для снижения *мерцания* показываемого изображения используется **чересстрочный** формат. Это означает, что в каждом кадре сначала передаются нечетные строки, а затем четные. Группа нечетных строк называется **нечетным полукадром**, а группа четных строк называется **четным полукадром**. Поскольку каждый кадр состоит из двух полукадров, видеосигнал передает 60 полукадров в секунду. Каждый полукадр начинается со сложной последовательности вертикальных синхроимпульсов длящихся 1,3 миллисекунды. За ней следуют либо четные, либо нечетные строки видео. Каждая строка длится 63,5 микросекунды, включая горизонтальный синхроимпульс длительностью 10,2 микросекунды, отделяющий одну строку от другой. В пределах каждой строки аналоговое напряжение соответствует шкале серого изображения с более яркими участками, располагающимися в сторону *повышения* от синхроимпульсов. Таким образом, синхроимпульсы располагаются за пределами уровня черного. На жаргоне видео говорят, что синхроимпульсы *чернее черного*.

Рис. 23.9 Композитный видеосигнал

Аппаратные средства, используемые для аналого-цифрового преобразования видео сигналов, называются **захват кадра**. Обычно они выполняется в виде электронной платы, вставляемой в компьютер и соединяемой с камерой при помощи коаксиального кабеля. По команде полученной от программного обеспечения захватчик кадра ждет начала следующего кадра, обозначенного вертикальными синхроимпульсами. Во время следования двух полукадров осуществляется многократная дискретизация каждой строки видео, обычно на строку приходится 512, 640 или 720 отсчетов при 8 битах на один отсчет. Эти отсчеты записываются в память в виде одной строки цифрового изображения.

Такой способ получения цифрового изображения приводит к важной разнице между вертикальным и горизонтальным направлениями. Каждая строка в цифровом изображении соответствует одной строке в видеосигнале и, следовательно, одной строке ям в ПЗС. К сожалению, столбцы получаются не очень прямыми. В ПЗС, в зависимости от конкретно используемого прибора, каждая строка содержит между приблизительно 400 и 800 ям (столбцов). При считывании строки ям из ПЗС, результирующая строка видео фильтруется в сглаженный аналоговый сигнал такой, как на рис. 23.9. Другими словами видеосигнал становится независимым от того, сколько столбцов находится в ПЗС. Разрешение в горизонтальном направлении ограничивается тем, насколько быстро может измениться аналоговый сигнал. Для цветного телевидения обычно устанавливается, что это 3,2 МГц, что дает время нарастания около 100 наносекунд, т.е. около 1/500-й от видео строки в 53,2 микросекунды.

При оцифровке изображения в захватчике кадров оно снова преобразуется в столбцы. Однако эти столбцы в цифровом изображении не имеют никакого отношения к столбцам в ПЗС. Число столбцов в цифровом изображении зависит исключительно от того, сколько раз захватчик кадра производил снятие отсчетов в каждой строке. Например, в ПЗС может быть 800 ям в строке, в то время как в оцифрованном изображении может быть всего 512 пикселей (т.е. столбцов) на строку.

Число столбцов в цифровом изображении также важно и по другой причине. Стандартное телевизионное **изображение** имеет **формат** 4 к 3, т.е. его ширина слегка больше, чем высота. В кино более широкий формат - 25 к 9. В ПЗС для научных приложений часто используется формат 1 к 1, т.е. идеальный квадрат. В любом случае формат ПЗС фиксирован расположением электродов и не может быть изменен. Однако формат оцифрованного изображения зависит от числа отсчетов на строку. Это становится проблемой при демонстрации изображения, как на видеомониторе, так и в виде твердой копии. Если формат изображения воспроизведен не должным образом, изображение выглядит расплющенным по горизонтали или вертикали.

Описанный здесь 525 строчный видеосигнал называется **NTSC** (National Television Systems Committee – Национальный комитет телевизионных систем), стандарт предопределил путь в далеком 1954. Это система, используемая в Соединенных Штатах и в Японии. В Европе приняты два подобных стандарта называемых **PAL** (Phase Alternation by Line – перемена фазы по строкам) и **SECAM** (Séquence de Couleurs Avec Mémoire –

поочередность цветов с памятью). Основная концепция здесь та же самая, просто другие числа. Как PAL, так и SECAM работают с 25 кадрами в секунду в чересстрочном формате при 625 строках на кадр. Так же, как и в NTSC, некоторые из этих строк зарезервированы за вертикальной синхронизацией, что дает около 576 строк несущих информацию картинки. Другие более тонкие различия касаются того, как к сигналу добавляются цвет и звук.

Наиболее простой способ передачи цветного телевидения состоял бы в том, чтобы иметь три отдельных аналоговых сигнала по одному на каждый из трех цветов, обнаруживаемых человеческим глазом: красный, зеленый и синий. К сожалению, историческое развитие телевидения отвергло такую простую схему. Цветной телевизионный сигнал развивался так, чтобы дать существовавшим черно-белым телевизорам возможность использоваться без их модификации. Это было реализовано за счет сохранения того же самого сигнала с информацией о яркости, но с добавлением отдельного сигнала с информацией о цвете. На жаргоне видео яркость называется *яркостный сигнал*, в то время как цвет называется *сигналом цветности*. Сигнал цветности содержится в несущей волне с частотой 3,58 МГц, добавленной к черно-белому видеосигналу. Звук добавлен таким же образом, на несущей волне с частотой 4,5 МГц. Телевизионный приемник разделяет три этих сигнала, индивидуально обрабатывает каждый, а яркостный сигнал и сигнал цветности снова объединяет при окончательном отображении.

Другие способы получения и демонстрации изображения

Не во всех изображениях полный кадр получается сразу. Другим очень типичным способом получения изображения является **построчное сканирование**. Оно включает использование датчика, содержащего одномерный массив пикселей, скажем 2048 пикселей в длину и 1 пиксель в ширину. По мере перемещения объекта мимо датчика строка за строкой считывается изображение. Построчное сканирование используется в факсимильных машинах и сканерах багажа в аэропортах. Как вариант объект может оставаться неподвижным, а датчик перемещаться. Последнее очень удобно, когда датчик уже установлен на перемещающемся объекте, например как на самолете, снимающем фотографии находящейся под ним поверхности. Преимущество построчного сканирования заключается в том, что *скорость* обменивается на *простоту* датчика. Например, на сканирование целой страницы текста у факсимильной машины может уйти несколько секунд, но результирующее изображение будет содержать тысячи строк и столбцов.

Еще более простой подход получать изображение **точки за точкой**. Например, микроволновое изображение Венеры было получено один пиксель за один раз. Другим примером является *микроскоп со сканирующим зондом*, способный давать изображение отдельных атомов. Небольшой зонд, часто на своем острие состоящий всего из одного отдельного атома, чрезвычайно близко приближается к отображаемой поверхности. Между образцом и зондом могут быть обнаружены квантово механические эффекты, позволяющие точно остановить зонд от поверхности образца. Затем зонд перемещается по поверхности образца на постоянном от него расстоянии, отслеживая пики и впадины. В конечном изображении значение каждого пикселя представляет собой возвышение соответствующего места над поверхностью образца.

Распечатанные изображения делятся на две категории: по **шкале серого** и по **полутонам**. Каждый пиксель в изображении по шкале серого представляет собой оттенок серого между черным и белым так, как на фотографии. Для сравнения, каждый пиксель изображения в полутонах формируется из многих отдельных *точек* каждая из которых либо полностью черная, либо полностью белая. Оттенки серого получаются за счет изменения различного количества этих белых и черных точек. Например, представьте

лазерный принтер с разрешением 600 точек на дюйм. Для получения 256 уровней яркости между черным и белым каждый пиксель соответствовал бы матрице 16 на16 печатных точек. Черные пиксели формируются, когда все эти 256 точек черные. В то время как белые пиксели формируются, когда все эти 256 точек белые. Средне серый состоит из половины белых точек и половины черных. Поскольку отдельные точки слишком малы, чтобы их можно было увидеть при рассматривании с нормального расстояния, происходит обман зрения и, кажется, что сформирована шкала серого.

Изображения в полутонах проще обрабатываются на принтерах, включая и светокопировальные машины. Их недостатком является то, что качество изображения часто хуже, чем у картинок по шкале серого.

Регулирование яркости и контрастности

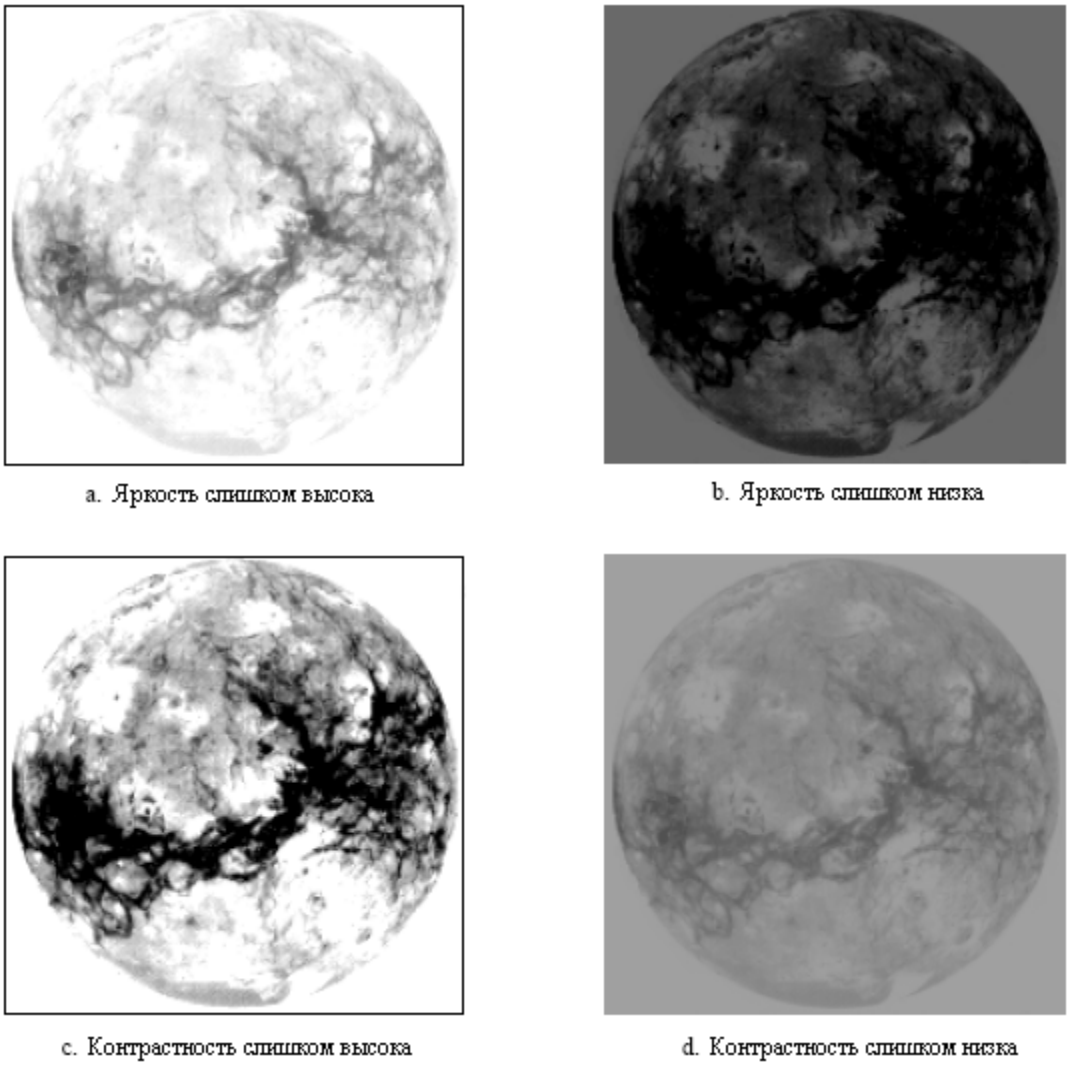
Для того, чтобы изображение было легко рассматривать, оно должно иметь надлежащую **яркость** и **контрастность**. Яркость относится к полной освещенности или затемненности объекта. Контрастность представляет собой *разницу* в яркости между объектами или областями. Например, белый заяц бегущий по снежному полю имеет *слабую* контрастность, в то время как черная собака по сравнению с тем же самым белым фоном обладает *хорошей* контрастностью. На рис. 23.10 показаны четыре возможных способа, которыми могут быть отрегулированы яркость и контрастность. Если яркость слишком высока, как на рис. 23.10a, самые белые пиксели насыщены и разрушают детали в этих областях. На рис. 23.10b показано обратное, когда установленная яркость слишком мала и насыщены черные пиксели. На рис. 23.10c показан случай установки высокой контрастности приводящий к тому, что черное становится слишком черным, а белое становится слишком белым. Наконец, на рис. 23.10d контрастность установлена слишком низкой; все пиксели средне серого оттенка, что приводит к исчезновению объектов друг в друге.

На рис. 23.11 и 23.12 *яркость* и *контрастность* иллюстрируются более детально. Тестовое изображение, показанное на рис. 23.12, имеет шесть различных уровней яркости и контрастности. На рис. 23.11 показана структура испытательного изображения, некоторого массива 80×32 пикселя в котором каждый пиксель имеет значение между 0 и 255. Фон испытательного изображения заполнен случайным шумом равномерно распределенным между 0 и 255. Значения пикселей в трех квадратах слева направо соответственно 75, 150 и 225. Каждый квадрат содержит два треугольника только со слегка отличающимися от окружающих их значениями пикселей. Другими словами, в изображении имеется темная область с бледными деталями, область изображения средней яркости с бледными деталями, и в изображении есть яркая область с бледными деталями.

На рис. 23.12 показано, как регулировка контрастности и яркости позволяет визуализировать в изображении различные особенности. На рис. 23.12a яркость и контрастность установлены на *нормальном* уровне, что показывают **ползунковые регуляторы B** и **C**, расположенные слева от изображения. А теперь обратите Ваше внимание на график, помещенный рядом с каждым изображением и называемый **выходным преобразованием**, **выходной таблицей поиска**, или **гамма кривой**. Этот график управляет аппаратными средствами ЭВМ, показывающими изображение. Значение каждого пикселя в хранимом изображении, представляющее собой число между 0 и 255, пропускается через данную таблицу поиска, для того чтобы получить другое число между 0 и 255. Это новое цифровое число управляет схемой интенсивности видео так, что числа от 0 до 255 преобразуются соответственно в оттенки от черного до белого. То есть таблица поиска отображает хранимые числа в яркость изображения.

На рис. 23.12a показано, как выглядит изображение, когда выходное преобразование устанавливается таким образом, чтобы не оказывать на изображение *никакого* влияния, т.е. цифровой выход идентичен цифровому входу. Каждый пиксель в

зашумленном фоне представляет собой случайный оттенок серого, равномерно распределенного между черным и белым. Три квадрата выглядят ясно отличаемыми друг от друга, как темный, средней яркости и светлый. Проблема в том, что треугольники внутри каждого квадрата невозможно легко заметить; чтобы отличить эти области от их окрестности контрастность для глаза слишком низка.



a. Яркость слишком высока

b. Яркость слишком низка

c. Контрастность слишком высока

d. Контрастность слишком низка

Рис. 23.10 Варианты регулировки яркости и контрастности

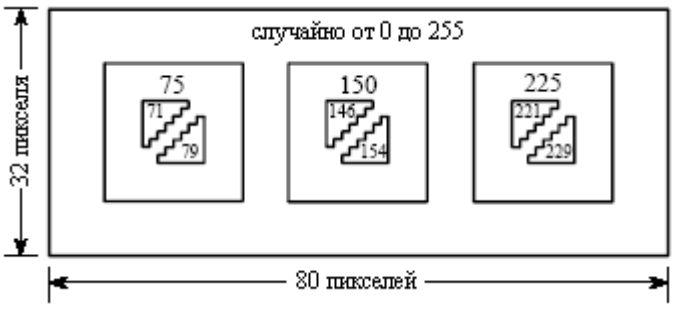


Рис. 23.11 Испытательное изображение для регулировки и оценки яркости и контрастности

На рис. 23.12b и рис. 23.12c показан эффект от изменения яркости. Сдвиг выходного преобразования *влево* увеличивает яркость, в то время как сдвиг выходного преобразования *вправо* уменьшает яркость. Увеличение яркости делает *каждый* пиксель в изображении более светлым. И наоборот, уменьшение яркости делает *каждый* пиксель в изображении более темным. Такие изменения в изображении могут улучшить визуальное восприятие чрезмерно темных или светлых областей изображения, но если пойти слишком далеко, будут **поглощать** изображение. Например, все пиксели в самом правом квадрате на рис. 23.12b отображаются с самой полной интенсивностью, т.е. 255. На рис. 23.12c показан противоположный эффект, где все пиксели в самом левом квадрате отображаются как наиболее черные, т.е. цифровое число ноль. Поскольку все пиксели в этой области имеют одно и то же значение треугольники полностью пропали. Заметьте также, что *ни один* из треугольников на рис. 23.12b или 23.12c увидеть нелегче, чем на рис. 23.12a. Изменение яркости слабо помогает (если вообще помогает) отличать слабо контрастные объекты от их окружения.

На рис. 23.12d показано отображение, оптимизированное для просмотра пикселей со значением около цифрового числа 75. Оптимизация осуществлена за счет повышения *контрастности*, выразившегося в увеличении *наклона* выходного преобразования. Например, хранимые значения пикселей 71 и 75 становятся при отображении 100 и 116, увеличивая контрастность в четыре раза. Значения пикселей между 46 и 109 отображаются в диапазоне от наиболее черных до наиболее белых. Ценой заплаченной за такое увеличение контрастности стало то, что значения пикселей от 0 до 45 насытились до черного, а значения пикселей от 110 до 255 насыщается до белого. Как показано на рис. 23.12d, увеличившаяся контрастность позволила треугольникам в левом квадрате стать видимыми ценой насыщения среднего и правого квадратов.

На рис. 23.12e показан эффект от еще большего увеличения контрастности, дающий всего 16 ненасыщенных из 256 отображаемых хранимых уровней. Яркость тоже была увеличена таким образом, чтобы центром 16 используемых уровней стало цифровое число 150. Теперь детали в центральном квадрате просматриваются очень хорошо, однако почти все другое в изображении насыщено. Например, взгляните на шум вдоль границ изображения. Здесь очень мало пикселей с серым промежуточным оттенком, почти каждый пиксель либо чисто черный, либо чисто белый. Такая техника использования высокой контрастности для разглядывания всего нескольких уровней, иногда называется **растягивание шкалы серого**.

Регулирование контрастности является способом *изменения масштаба* на небольшом диапазоне значений пикселей. Регулирование яркости *центрирует* масштабируемый участок интересующих значений пикселей. Большинство систем цифровых изображений допускают регулировку яркости и контрастности именно таким образом, и часто дают графическое отображение выходного преобразования (как на рис. 23.12). Для сравнения, регуляторы яркости и контрастности в телевизорах и мониторах представляют собой *аналоговые цепи* и могут работать по-разному. Например, управление контрастностью монитора может достигаться регулированием усиления аналогового сигнала, тогда как управление яркостью может достигаться добавлением или вычитанием постоянного смещения. Мораль такова, не удивляйтесь, если подобные аналоговые средства управления работают не так, как Вы предполагали.

Преобразование шкалы серого

Последнее изображение на рис. 23.12f отличается от остальных. Здесь, вместо того чтобы у кривой был наклон в *одном* диапазоне входных значений, кривая имеет наклон в *двух* диапазонах. Это позволяет дисплею одновременно показывать треугольники в обоих левом и правом квадратах. Конечно, это приводит к насыщению значений пикселей, которые *не* находятся вблизи данных цифровых чисел. Обратите внимание, что

ползунковые регуляторы для контрастности и яркости на рис. 23.12f не показаны, это отображение за пределами того, что могут обеспечить регулировки яркости и контрастности.

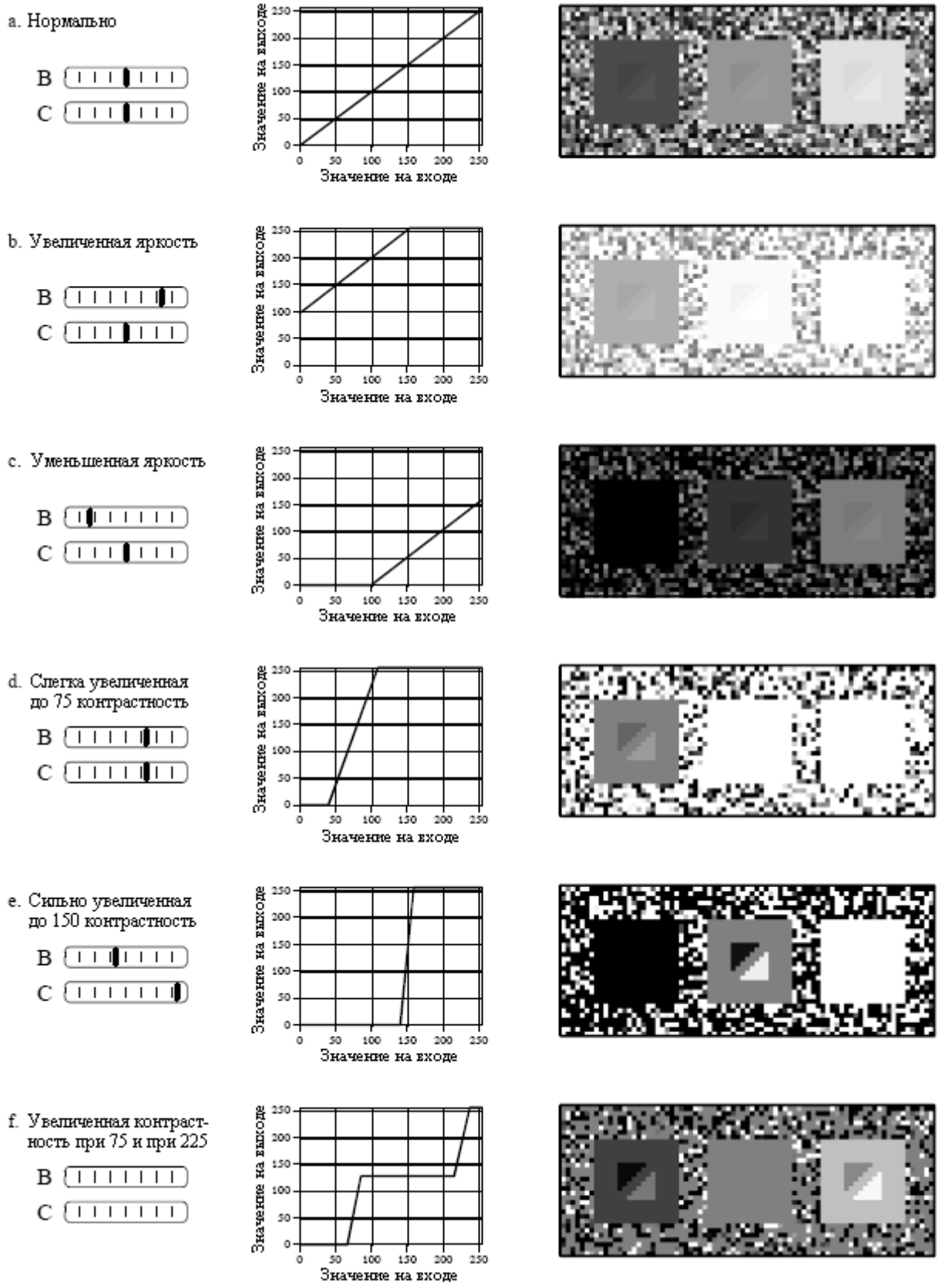


Рис. 23.12 Визуализация различных особенностей в изображении

Дальнейшее развитие этого подхода приводит к мощной технике улучшения внешнего вида изображения: **преобразованию шкалы серого**. Идея состоит в том, чтобы увеличить контрастность значений нужных пикселей за счет значений пикселей, не представляющих интереса. Это выполняется за счет определения относительной важности каждого из возможных от 0 до 255 значений пикселя. Чем важнее значение, тем больше делается его контрастность в отображаемом изображении. Систематический способ реализации этой процедуры будет показан на примере.

a. Исходное ИК изображение

b. С преобразованием шкалы серого

Рис. 23.13 Обработка преобразованием шкалы серого

Изображение на рис. 23.13 было получено в полной темноте при помощи камеры на ПЗС, чувствительной к инфракрасному излучению в нижней части спектра. Отображаемым параметром является температура: чем горячее объект, тем больше он испускает инфракрасной энергии, и тем ярче выглядит на изображении. Это объясняет очень темный (холодный) фон, серое (теплое) тело и белую (горячую) решетку радиатора грузовика. Эти системы очень хороши для военных и полиции; Вы можете видеть другого парня даже тогда, когда он сам себя не может видеть! Из-за неравномерного распределения значений пикселей изображение на рис. 23.13a трудно разглядеть. Большая часть изображения настолько темна, что нельзя рассмотреть детали в сцене. С другой стороны решетка радиатора близка к насыщенному белому.

Гистограмма этого изображения, показанная на рис. 23.14a показывает, что фон, человек и решетка радиатора имеют приемлемо различные значения. В этом примере мы будем увеличивать контрастность фона и решетки радиатора за счет всего остального, включая и тело человека. Такая стратегия представлена на рис. 23.14b. Примем, что самые низкие значения пикселей, т.е. фон, будут иметь относительную контрастность равную двенадцати. Тогда как самые высокие значения пикселей, т.е. решетка радиатора, будут иметь относительную контрастность равную шести. Тело будет иметь относительную контрастность, равную единице при ступенчатом переходе между областями. Все эти значения определяются методом проб и ошибок.

Преобразование шкалы серого, вытекающее из такой стратегии и отмеченное, как выполненное *вручную*, показано на рис. 23.14c. Оно найдено взятием бегущей суммы (т.е. дискретного интеграла) от кривой на рис. 23.14b, а затем нормализовано таким образом, чтобы с правой стороны оно было равно 255. Почему для нахождения требуемой кривой

необходимо брать *интеграл*? Подумайте об этом следующим образом: Контрастность для определенного значения пикселя равна наклону выходного преобразования. То есть мы хотим, чтобы кривая на рис. 23.14b была производной (наклоном) кривой на рис. 23.14c. Это означает, что кривая на рис. 23.14c должна быть интегралом от кривой на рис. 23.14b.

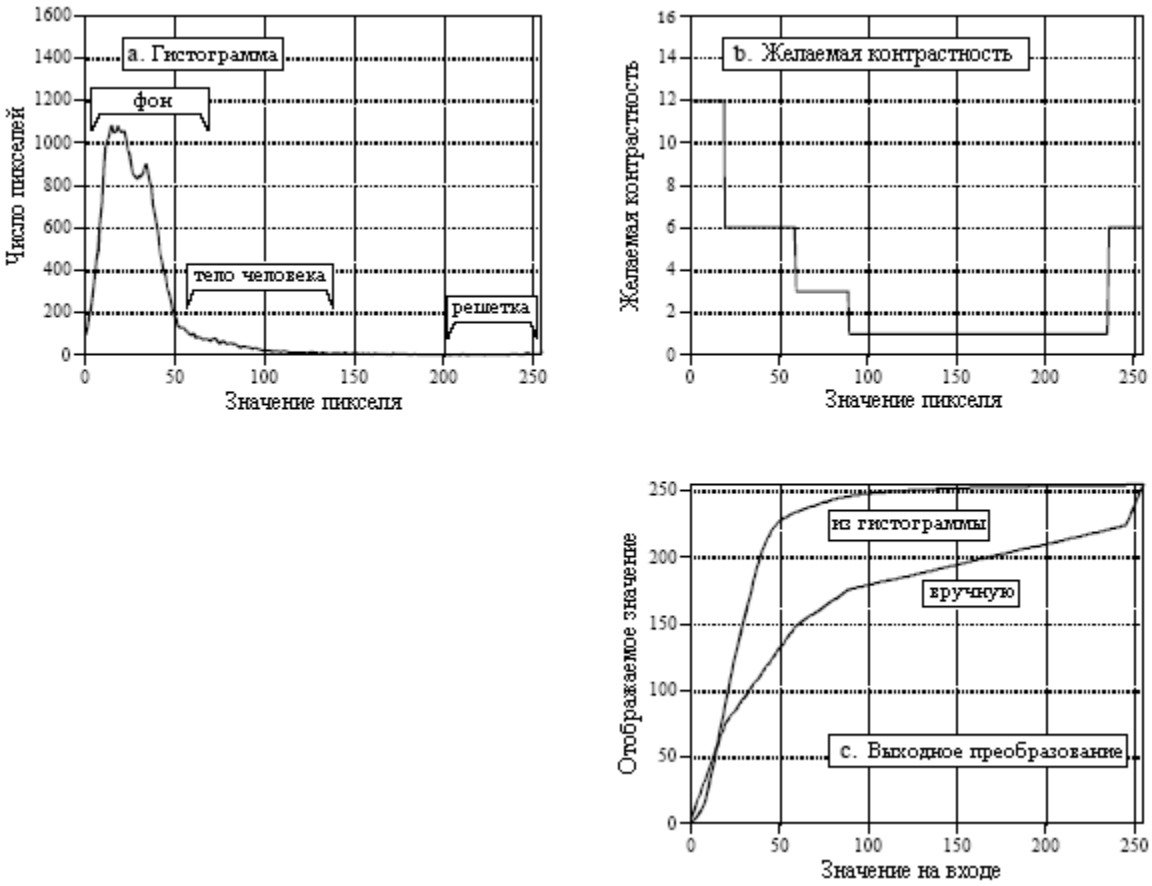


Рис. 23.14 Развитие преобразования шкалы серого

Пропуская изображение, приведенное на рис. 23.13a, через это, определенное вручную преобразование шкалы серого, получим изображение, показанное на рис. 23.13b. Фон стал *светлее*, решетка радиатора стала *темнее*, и обе обладают более хорошей контрастностью. Эти улучшения достигаются ценой ухудшения контрастности тела, дающей менее детализированное изображение нарушителя (хотя оно не может получиться еще хуже, чем в оригинальном изображении).

Преобразования шкалы серого могут значительно улучшить визуальное восприятие изображения. Проблема в том, что они могут потребовать большого числа проб и ошибок. Способом автоматизировать эту процедуру является **уравнивание гистограммы**. Обратите внимание, что гистограмма на рис. 23.14a и кривая взвешивания контрастности на рис. 23.14b имеют туже самую общую форму. Уравнивание гистограммы вслепую использует гистограмму в качестве кривой взвешивания контрастности, исключая необходимость в суждениях человека. То есть, вместо генерирования кривой вручную, выходное преобразование находится при помощи интегрирования и нормализации *гистограммы*. В результате самую большую контрастность получают те значения, которые имеют наибольшее число пикселей.

Уравнивание гистограммы интереснейшая математическая процедура, поскольку она максимизирует *энтропию* изображения, меру количества информации, передаваемой фиксированным числом битов. Недостатком уравнивания гистограммы является то, что оно ошибочно смещает значения *ряда* пикселей к некоторой величине без учета

важности пикселей с этими значениями в изображении. Например, решетка радиатора грузовика и человек нарушитель на рис. 23.13 являются наиболее заметными особенностями. Несмотря на это, уравнивание гистограммы почти полностью игнорировало бы эти объекты, потому что они содержат относительно небольшое число пикселей. Уравнивание гистограммы легко и быстро. Просто запомните, что если оно не работает хорошо, то созданная вручную кривая, вероятно, будет работать намного лучше.

Деформирование

Одной из проблем фотографирования поверхности планеты является искажение изображения, возникающее в связи с наличием у сферической формы кривизны. Например, предположим, что Вы используете телескоп, для фотографирования квадратной области вблизи центра планеты, как это показано на рис. 23.15a. Через несколько часов планета повернется вокруг своей оси и будет выглядеть так, как это показано на рис. 23.15b. Ранее сфотографированная область выглядит сильно искаженной, поскольку вблизи горизонта планеты она сильно искривлена. Каждое из этих двух изображений содержит полную информацию о сфотографированной области, просто фотографии сделаны в разных ракурсах. Весьма обычно получить фотографию выглядящую так, как на рис. 23.15a, но в действительности желать получить фотографию, как на рис. 23.15b, или наоборот. Например, при картографировании поверхности планеты спутник может сделать тысячи фотографий прямо сверху, как на рис. 23.15a. Для того чтобы сделать общую картину планеты выглядящей естественно, наподобие изображения Венеры на рис. 23.1, изображение каждой фотографии должно быть искажено и помещено в соответствующее место. С другой стороны рассмотрим спутник погоды, наблюдающий за ураганом не находящимся непосредственно под спутником. Здесь нет другого выбора, кроме как получить косое изображение, как на рис. 23.15b. Затем изображение преобразуется к виду, как если бы оно было снято прямо сверху, как на рис. 23.15a.

Такие пространственные преобразования называются **деформированием**. Наиболее часто деформирование применяется в космической фотографии, но существуют и другие применения. Например, масса датчиков изображения на вакуумных трубках обладают разным пространственным искажением. Сюда входят и применяемые военными приборы ночного видения, и используемые в области медицины датчики рентгеновского излучения. Цифровое деформирование (или если Вы предпочитаете *обратное деформирование*) может быть использовано для коррекции присущих этим приборам искажений. Любят деформировать изображения и художники спецэффектов в кинематографе. Например, техника, называемая трансформацией, в пределах нескольких кадров постепенно деформирует один объект в другой. Это может создать иллюзию наподобие превращения ребенка во взрослого или человека превращающегося в оборотня.

Деформирование берет *исходное изображение* (двумерный массив) и генерирует *деформированное изображение* (другой двумерный массив). Это выполняется с помощью циклического прохода через каждый пиксель в деформируемом изображении и выяснении: Каково надлежащее значение пикселя, которое должно быть сюда помещено? Заданным конкретным вычисляемым строке и столбцу в деформируемом изображении соответствуют надлежащие строка и столбец в исходном изображении. Для выполнения алгоритма значение пикселя из исходного изображения преобразуется в деформированное изображение. На жаргоне обработки изображений строка и столбец в исходном изображении, *откуда приходит* пиксель, называется **адрес убытия**. Преобразование каждого пикселя исходного изображения в пиксель деформированного изображения представляет собой простую часть процедуры. Трудную часть представляют вычисления ассоциируемого с каждым пикселем в деформируемом изображении *адреса убытия*. Обычно это чисто математическая задача и может быть весьма запутанной. Простое растягивание изображения в горизонтальном и вертикальном направлении значительно

легче и для нахождения адреса убытия включает только перемножение номеров строк и/или столбцов.

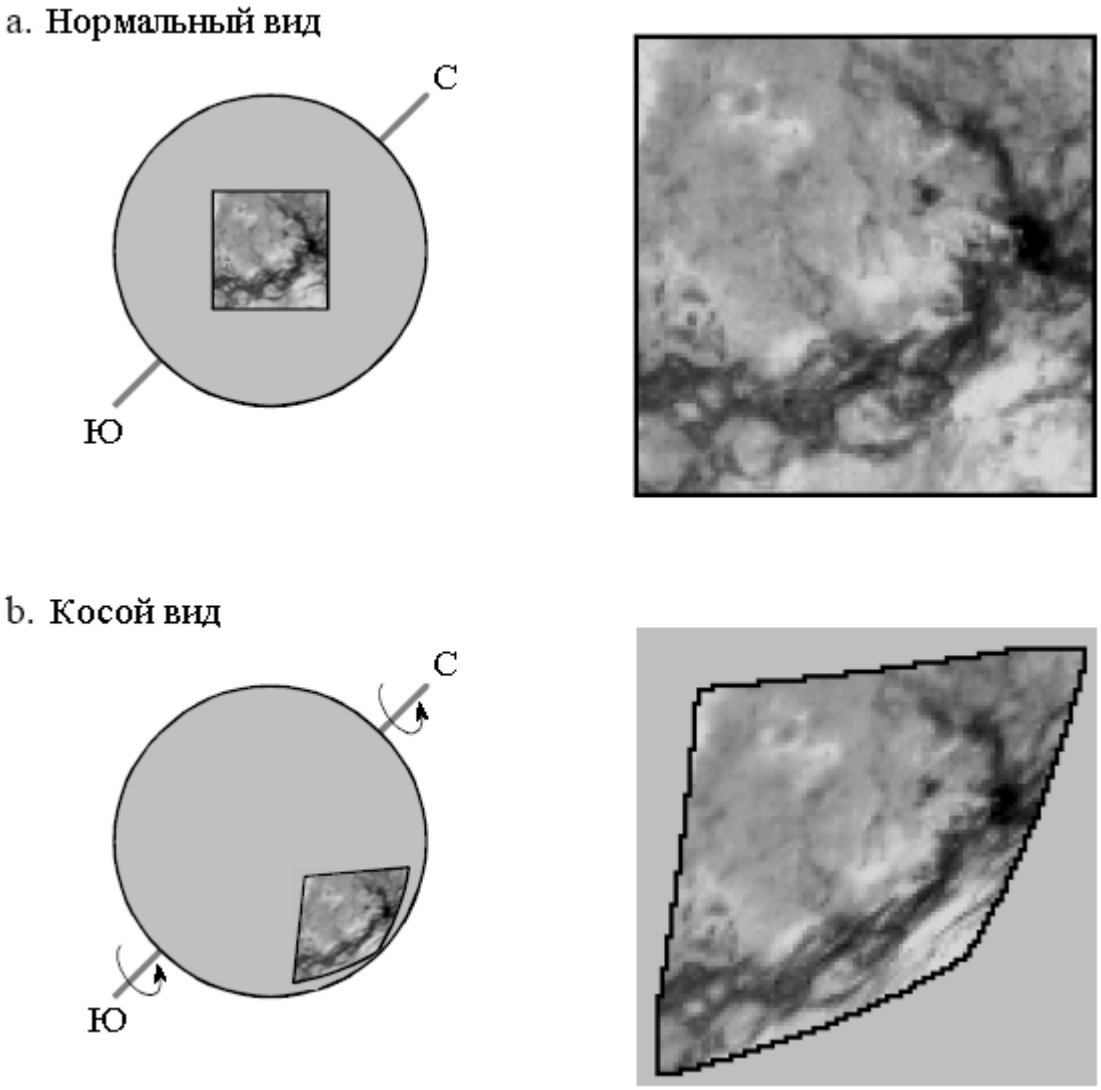


Рис. 23.15 Деформирование изображения

Одним из методов используемых при деформировании является **подпиксельная интерполяция**. Например, предположим, что Вы разработали систему уравнений, которая преобразует адрес строки и столбца в деформируемом изображении в адрес убытия в исходном уравнении. Посмотрим, что может случиться, когда Вы пытаетесь найти значение пикселя на пересечении 10 строки и 20 столбца в деформируемом изображении. Вы подставляете в свои уравнения следующую информацию: *строка = 10*; *столбец = 20*; и получаете: *строка убытия = 20,2*; *столбец убытия = 14,5*. Дело в том, что Ваши вычисления будут, вероятно, использовать плавающую запятую, и поэтому адреса убытия не будут целыми числами. Наиболее простым способом, который здесь можно использовать является алгоритм **ближайшего соседа**, то есть просто округлить адрес до ближайших целых чисел. Это просто, но может привести к большой зернистости по краям объектов, где пиксели могут оказаться немного не на своих местах.

Билинейная интерполяция требует чуть больших усилий, но дает значительно лучшее изображение. На рис. 23.16 показано, как она работает. Вам известно значение четырех пикселей, *окружающих* дробный адрес, т.е. значение пикселей в строках 20 и 21, и столбцах 14 и 15. В этом примере мы примем в качестве значений пикселей следующие: 91, 210, 162 и 95. Проблема состоит в том, чтобы осуществить интерполяцию между

этими четырьмя значениями. Это делается за два этапа. Во-первых, осуществляется интерполяция в *горизонтальном* направлении между столбцами 14 и 15. Это дает два промежуточных значения 150,5 в строке 20 и 128,5 в строке 21. Во-вторых, осуществляется интерполяция между этими промежуточными значениями в *вертикальном* направлении. Это дает билинейно интерполированное значение пикселя равное 139,5, которое затем предается в деформированное изображение. Почему интерполяция осуществляется в горизонтальном направлении *и лишь затем* в вертикальном направлении, а не наоборот? Это не имеет значения; конечный результат будет тем же самым в независимости от того, какой порядок будет использоваться.

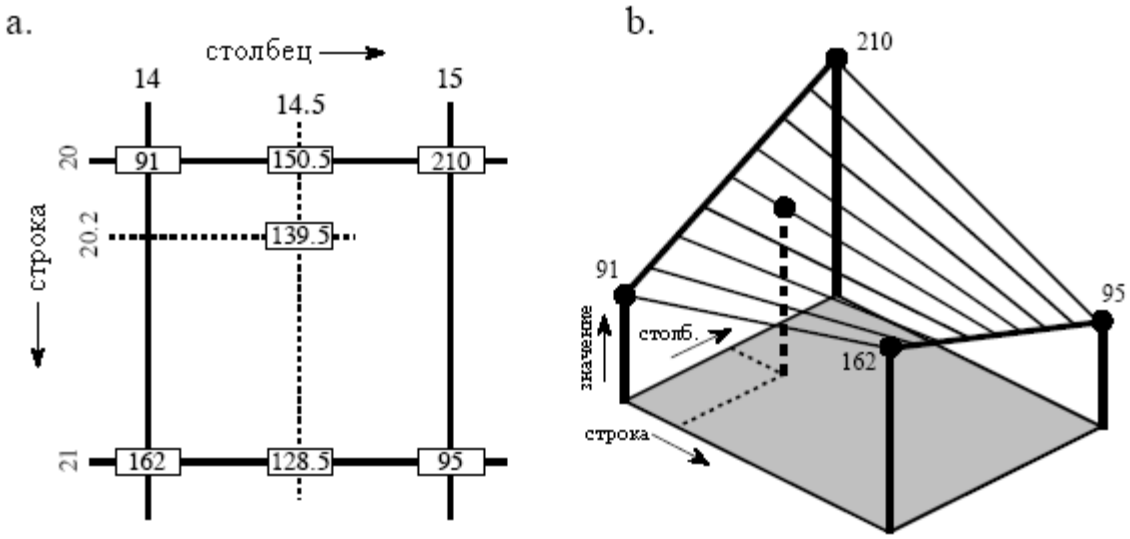


Рис. 23.16 Подпиксельная интерполяция

ГЛАВА
24

ЛИНЕЙНАЯ ОБРАБОТКА ИЗОБРАЖЕНИЯ

Линейная обработка изображения базируется на тех же самых двух методах, что и традиционная ЦОС: *свертка* и *анализ Фурье*. Наиболее важной из обоих является свертка, поскольку информация в изображении закодирована в пространственной области, а не в частотной области. Линейная фильтрация может улучшить изображения многими способами: заострить края объектов, снизить случайный шум, скорректировать неодинаковую освещенность, выполнить обратную свертку для коррекции нерезкости и смазанности движения и т.д. Эти процедуры осуществляются сверткой исходного изображения с соответствующим ядром фильтра, создающим отфильтрованное изображение. Серьезной проблемой свертки изображений является огромное число вычислений, необходимое для ее осуществления, часто приводящих к неприемлемо длительному времени исполнения. Эта глава представляет стратегию разработки ядра фильтра для различных задач обработки изображения. Описываются так же два важных метода снижения времени исполнения: *свертка посредством разделения* и *БПФ свертка*.

Свертка

Свертка изображения работает таким же образом, что и одномерная свертка. Изображения, например, могут рассматриваться как сумма *дельта импульсов*, т.е. смасштабированных и сдвинутых дельта функций. Аналогично *линейные системы* характеризуются тем, как они отвечают на дельта импульсы, т.е. их *импульсными характеристиками*. Как Вы и должны предполагать, выходное изображение, получаемое от системы, равно входному изображению, *свернутому* с импульсной характеристикой системы.

Двумерная дельта функция представляет собой изображение, состоящее из всех нулей за исключением отдельного пикселя *в строке 0*, *столбце 0*, имеющего значение равное *единице*. А теперь, условимся, что индексы строк и столбцов могут принимать как отрицательные, так и положительные значения, таким образом, *единица* находится в центре огромного моря нулей. При прохождении дельта функции через линейную систему единственная ненулевая точка будет преобразовываться в некоторый другой двумерный образец. Поскольку единственной вещью, которая может произойти с точкой, является то, что она станет *размытой*, импульсная характеристика на жаргоне обработки изображений часто называется **функцией размыва точки** (**ФРТ**).

Превосходный пример этой концепции дает глаз человечка. Как описывалось в предыдущей главе, первый слой сетчатки преобразует изображение, представленное в виде светового рисунка, в изображение, представленное в виде рисунка нервных импульсов. Второй слой сетчатки *обрабатывает* это нейронное изображение и передает его в третий слой - волокна, формирующие зрительный нерв. Представьте, что изображение, спроецированное на сетчатку, представляет собой очень маленькое пятнышко света посреди темного фона. То есть поданный в глаз *дельта импульс*. Условимся, что система линейна, а процесс обработки изображения, происходящий в сетчатке, может быть определен наблюдением изображения появившегося, в зрительном

нерве. Другими словами, мы хотим найти *функцию размыва точки* от обработки. Позже в этой главе мы еще вернемся к предположению о линейности глаза.

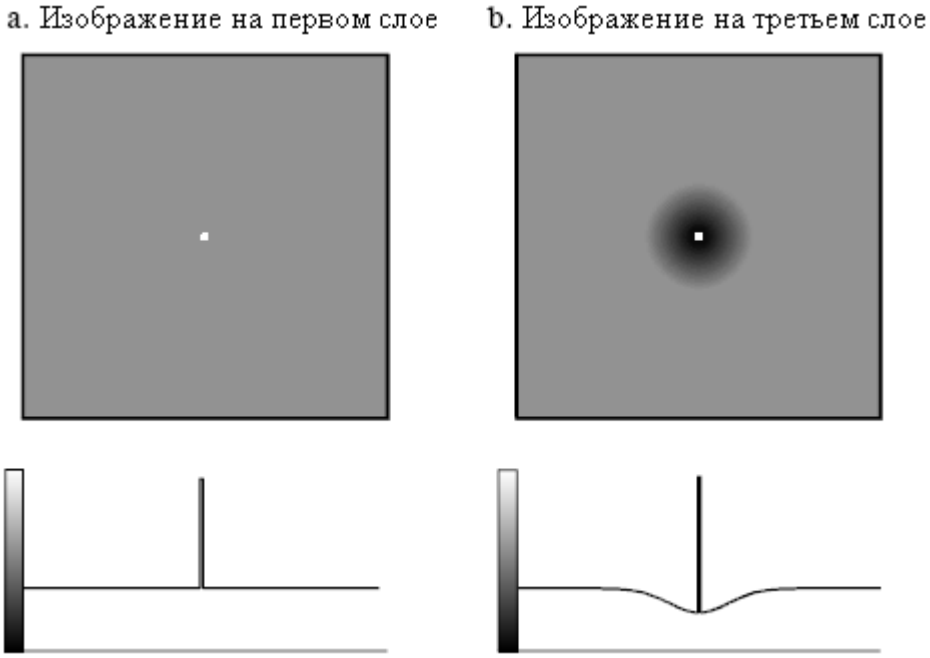


Рис. 24.1 ФРТ глаза

Этот эксперимент представлен на рис. 24.1. На рис. 24.1a показан падающий на сетчатку дельта импульс, тогда как на рис. 24.1b показано появляющееся в зрительном нерве изображение. Средний слой глаза пропускает яркий лучик, но создает вокруг него область с повышенной *затемненностью*. Глаз реализует это посредством процесса, известного под названием *поперечное поглощение*. Если в среднем слое активизировалась нервная клетка, то это уменьшает способность ее ближайших соседей стать активными. Когда глазом рассматривается полное изображение, каждая точка в рассматриваемом изображении добавляет к появляющемуся в зрительном нерве изображению смасштабированную и сдвинутую версию такого импульсного отклика. Другими словами, для получения передаваемого в мозг нейронного изображения осуществляется *свертка* наблюдаемого изображения с данной ФРТ. Возникает очевидный вопрос: каким образом свертка наблюдаемого изображения с данной ФРТ улучшает способность глаза понимать мир?

Люди и другие животные используют зрение для опознавания близлежащих объектов таких, как враги, пища и возлюбленные. Это осуществляется отделением одной области изображения от другой на основе различий в яркости и цветах. Другими словами, первым шагом в распознавании объектов является идентификация их *краев*, разрыва отделяющего объект от фона. Средний слой сетчатки помогает этой задаче посредством заострения краев в наблюдаемом изображении. На рис. 24.2 в качестве иллюстрации того, как это работает, показано изображение, медленно изменяющееся от темного к белому, дающее размытый и слабо различимый край. На рис. 24.2a показан профиль интенсивности этого изображения, образец попадающей в глаз яркости. На рис. 24.2b показан профиль яркости, появляющийся в зрительном нерве, изображение, передаваемое в мозг. Обработка в сетчатке делает края между светлыми и темными областями, выглядящими более резкими, усиливая впечатление того, что две области различны.

Перерегулирование в отклике на край создает интересную оптическую иллюзию. Сразу за краем появившаяся темная область кажется необычно темной, а появившаяся светлая область кажется необычно светлой. Получившиеся светлые и темные зоны

получили название **полос Маха** по имени Эрнста Маха (1838-1916) австрийского физика который впервые описал их.

Рис. 24.2 Полосы Маха

Как и в случае одномерных сигналов, свертка изображения может рассматриваться с двух сторон: со стороны входа и со стороны выхода. Со стороны входа каждый пиксель во входном изображении добавляет к выходному изображению смасштабированную и сдвинутую версию функции размыва точки. При рассмотрении со стороны выхода каждый пиксель в выходном изображении находится под влиянием группы пикселей из входного сигнала. Для одномерных сигналов такой областью влияния является перевернутая *слева направо* импульсная характеристика. Для сигналов изображения это перевернутая *слева направо* и *сверху вниз* ФРТ. Поскольку большинство используемых в ЦОС ФРТ симметричны относительно горизонтальной и вертикальной осей, такой переворот ни к чему не приводит и может игнорироваться. Позже в этой главе мы будем рассматривать несимметричные ФРТ, требующие принимать переворот во внимание.

На рис. 24.3 показано несколько типичных ФРТ. На рис. 24.3a показана **пилюлеобразная** ФРТ круглая вершина с прямыми сторонами. Например, если линза камеры не будет сфокусирована надлежащим образом, каждая точка в изображении будет проецироваться на датчик изображения в виде круглого пятна (вернитесь назад к рис. 23.2 и рассмотрите эффект от перемещения проекционного экрана от линзы вперед или назад). Другими словами, пилюлеобразная ФРТ - функция размыва точки не сфокусированных линз.

На рис. 24.3b показан **Гауссиан**, являющийся ФРТ систем изображения, ограниченных случайными дефектами. Например, изображение от телескопа из-за турбулентности в атмосфере становится размытым, что является причиной того, что каждая точка света в конечном изображении становится Гауссианом. Датчики изображения наподобие ПЗС и сетчатки часто ограничены рассеиванием света и/или

электронов. Центральная предельная теорема диктует, что результатом этих типов вероятностных процессов является Гауссово пятно.

Пилюлеобразная ФРТ и Гауссиан применяются для обработки изображений точно так же, как *фильтр скользящего среднего* применяется для обработки одномерных сигналов. Изображение, свертка которого осуществляется с этими ФРТ, будет выглядеть размыто и иметь слабо выраженные края, но с более низким случайным шумом. За их действие в пространственной области они получили название **сглаживающих фильтров**, или **низкочастотных** за их действие в частотной области. **Квадратная** ФРТ, показанная на рис. 24.3c, также может использоваться как сглаживающий фильтр, но у нее нет круговой симметрии. Это приводит к различию размытости изображения в диагональных направлениях по сравнению с горизонтальным и вертикальным направлениями. В зависимости от приложения, это может быть важно или не важно.

Противоположностью сглаживающего фильтра является **фильтр**, **улучшающий края** или **высокочастотный фильтр**. Для перехода к высокочастотным фильтрам от низкочастотных используется метод спектральной инверсии, обсужденный в Главе 14. Как показано на рис. 24.3d, ядро фильтра, улучшающего края, формируется посредством *изменения знака* ФРТ сглаживающего фильтра и добавлением дельта функции в центре. Примером этого типа фильтра является обработка изображения, происходящая в сетчатке.

На рис. 24.3e показана двумерная синк функция. Одномерная обработка сигналов использует ограниченную окном синк функцию для разделения полос частот. Поскольку изображения не содержат информации, закодированной в частотной области, синк функция редко используется в качестве ядра фильтра изображения, хотя она находит свое применение в некоторых теоретических задачах. Синк функция может быть трудна для использования, поскольку ее хвосты очень медленно уменьшаются по амплитуде (*1/x*), что означает, что ее следует рассматривать как бесконечно широкую. Для сравнения, хвосты Гауссиана уменьшаются очень быстро (e^{-x^2}) и, в конечном счете, могут быть безболезненно усечены.

Во всех этих ядрах фильтров в строках и столбцах используются *отрицательные* индексы, позволяющие центрировать ФРТ относительно *строки = 0* и *столбца = 0*. В одномерной ЦОС отрицательные индексы часто устраняются за счет смещения ядра фильтра вправо до тех пор, пока индексы ненулевых отсчетов не станут положительными. Такой сдвиг смещает выходной сигнал на туже величину, что обычно не имеет значения. Для сравнения, сдвиг между входным и выходным изображениями обычно неприемлем. Таким образом, в обработке изображений отрицательные индексы являются нормой для ядер фильтров.

Проблема со сверткой изображения состоит в том, что сюда вовлечено большое число вычислений. Например, при свертке изображения 512 на 512 пикселей с ФРТ 64 на 64 пикселя требуется более *миллиарда* умножений и сложений (т.е. 64×64×512×512). Большое время исполнения может сделать методы непрактичными. Для ускорения этих вещей используются три подхода.

Первой стратегией является использование очень маленькой ФРТ, часто всего 3×3 пикселя. Стратегия реализуется посредством циклического прохода через каждый отсчет в выходном изображении, используя оптимизированный код для умножения и аккумулирования соответствующих девяти пикселей из входного изображения. С небольшой 3×3 ФРТ может быть достигнут удивительный итог обработки, поскольку она достаточно велика для того, чтобы повлиять на *края* в изображении.

Второй подход используется тогда, когда требуется большая ФРТ, но ее форма не является критической. Для этого необходимо *сепарабельное* ядро фильтра, свойство позволяющее осуществлять свертку изображения, как ряд одномерных операций. Это может улучшить время исполнения в *сотни* раз.

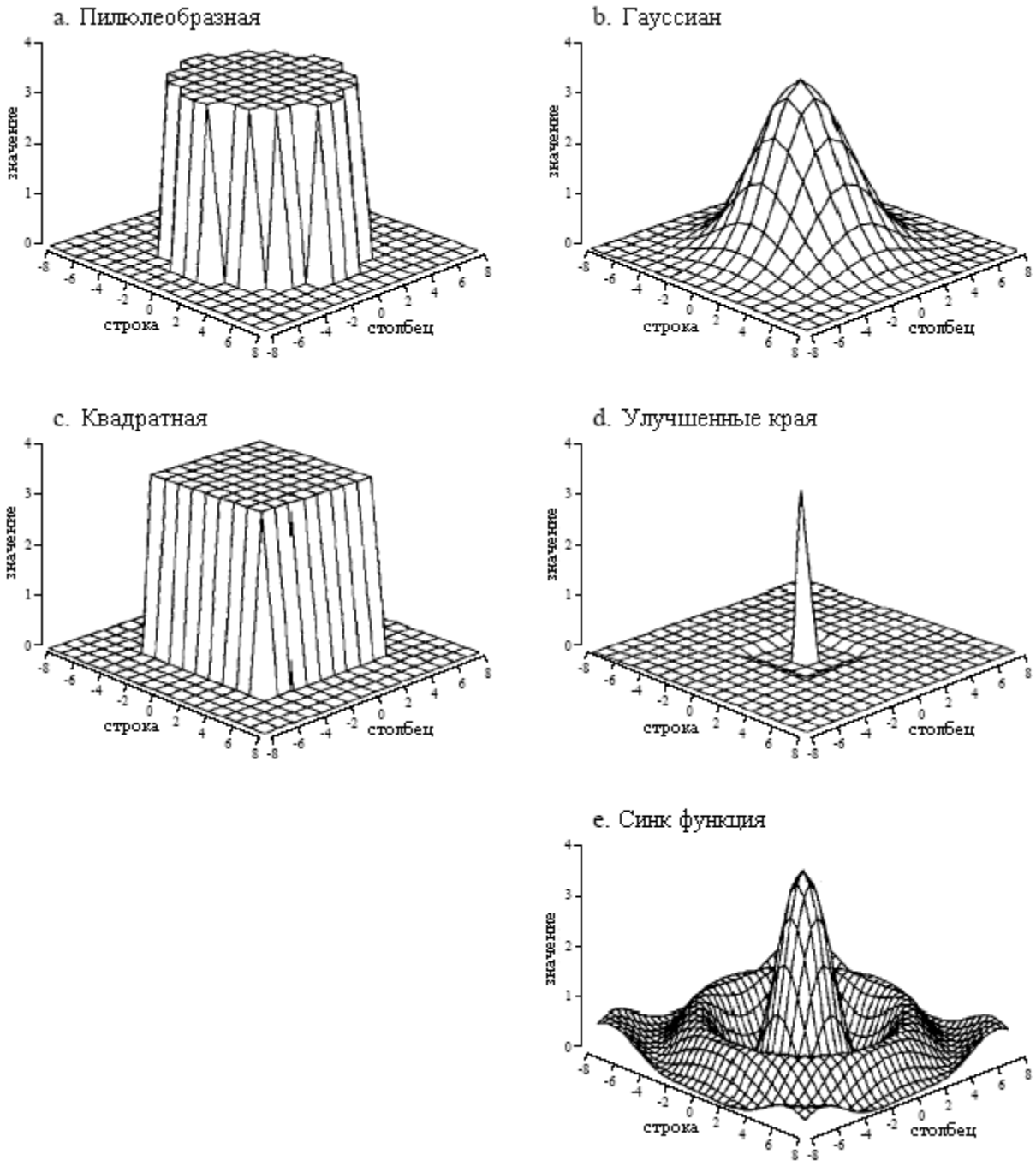


Рис. 24.3 Типичные функции размыва точки

Третьей стратегией является БПФ свертка, используемая тогда, когда ядро фильтра большое и имеет специфическую форму. Даже при улучшении скорости обеспечиваемой высокоэффективным БПФ, время исполнения будет отвратительным. Давайте подробнее рассмотрим детали этих трех стратегий и примеры того, как они используются в обработке изображения.

3×3 модификация краев

На рис. 24.4 показано несколько операций 3×3. На рис. 24.4a показано изображение, полученное рентгеновским сканером багажа в аэропорту. При осуществлении свертки этого изображения с дельта функцией 3×3 (*единица*, окруженная 8

нулями), изображение не изменяется. Само по себе это не интересно, здесь формируется основа для других ядер фильтров.

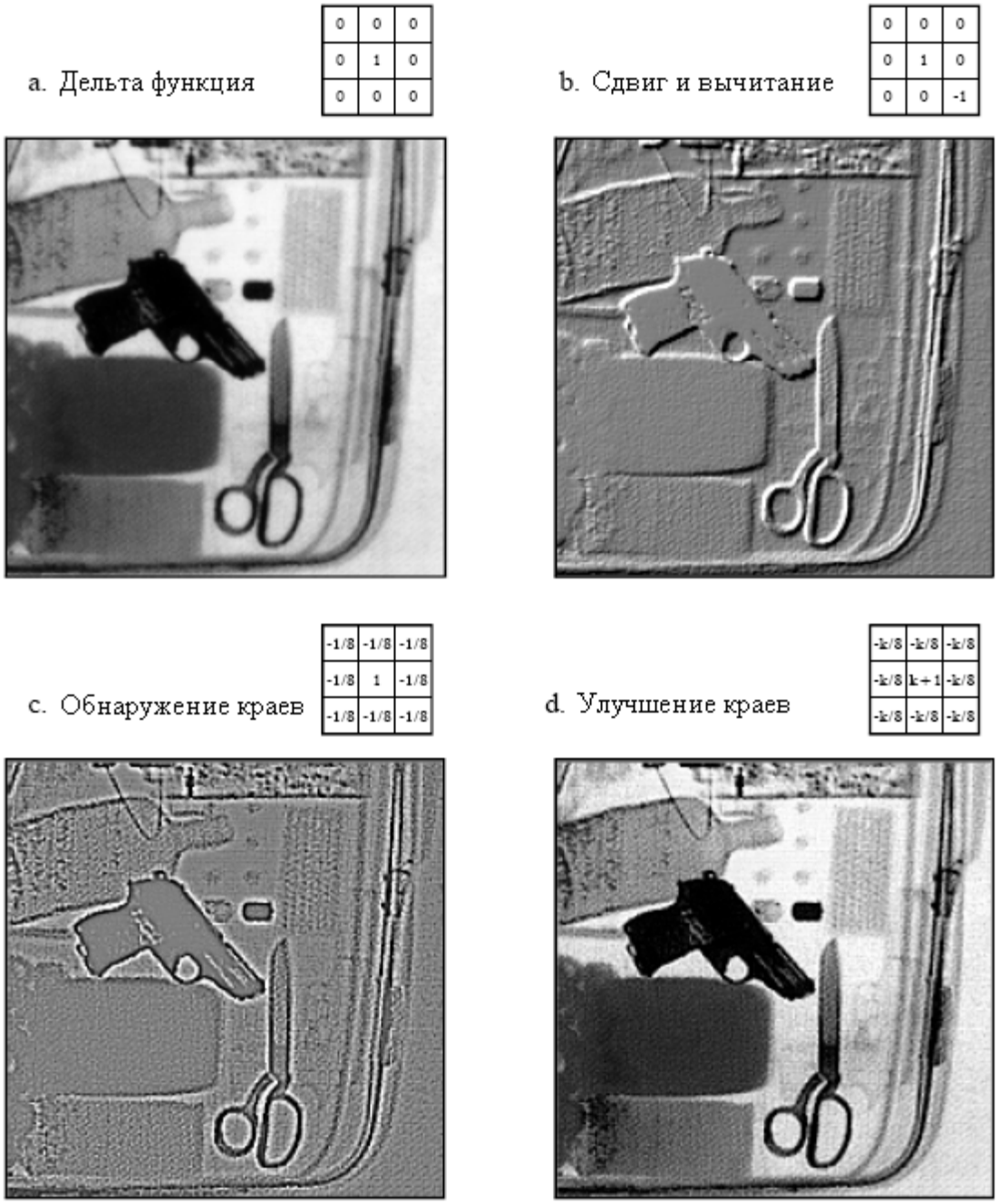


Рис. 24.4 3×3 модификация краев

На рис. 24.4b показано изображение, свернутое с единицей, минус единицей и 7 нулями. Эта операция называется **сдвиг и вычитание**, поскольку *сдвинутая* версия изображения (соответствует –1) *вычитается* из исходного изображения (соответствует 1). Такая обработка дает оптическую иллюзию того, что некоторые объекты становятся ближе или дальше чем фон, создавая трехмерный или рельефный эффект. Мозг интерпретирует изображения, как если бы освещение шло *сверху*, нормальный способ представления мира. Если края объекта яркие вверху и темные у основания, объект воспринимается как выступающий над фоном. Чтобы увидеть другой интересный эффект, переверните картинку сверху вниз, и объект будет *вдавлен* в фон.

На рис. 24.4c показана ФРТ **обнаруживающая край** и результирующее изображение. Каждый край в исходном изображении трансформируется в узкие темные и светлые полосы, которые идут параллельно с исходным краем. Изменяя порог этого изображения, можно изолировать либо темную, либо светлую полосу, получая простой алгоритм для обнаружения краев в изображении.

Типичный метод обработки изображения показан на рис. 24.4d: **улучшение краев**. Иногда его называют операцией *заострения*. На рис. 24.4a объекты обладают хорошей контрастностью (соответствующим уровнем затемненности и освещенности), но имеют очень размытые края. На рис. 24.4c объекты абсолютно не контрастны, но обладают очень отчетливыми краями. Стратегия состоит в том, чтобы умножить изображение с *хорошими краями* на постоянную k и добавить его к изображению с *хорошей контрастностью*. Это равносильно свертке исходного изображения с ФРТ 3×3, показанной на рис. 24.4d. Если установить k равным нулю, ФРТ становится дельта функцией, и изображение не изменяется. По мере увеличения k резкость границ изображения улучшается. Для изображения на рис. 24.4d было использовано значение $k=2$: две части изображения на рис. 24.4c к одной части изображения на рис. 24.4a. Эта операция имитирует способность глаза заострять края, позволяя более легко отделять объекты от фона.

Свертка с любой из этих ФРТ может привести к появлению отрицательных значений пикселей в конечном изображении. Даже если программа может обрабатывать отрицательные значения пикселей, выводящий изображение дисплей этого не может. Наиболее обычным способом обойти это является добавление смещения к каждому из вычисленных пикселей, как это сделано в данном изображении. В качестве альтернативы можно усечь выходные значения вышедшие за пределы диапазона.

Свертка разделением

Это техника выполнения быстрой свертки, использующая **сепарарабельность** ФРТ. Говорят, что ФРТ *сепарабельна*, если ее можно разбить на два одномерных сигнала: вертикальную и горизонтальную проекции. На рис. 24.5 показан пример сепарабельного изображения: квадратная ФРТ. Определенно, что значение каждого пикселя в изображении равно соответствующей точке на горизонтальной проекции, умноженной на соответствующую точку на вертикальной проекции. В математической форме это выражается как:

$$x[r,c] = vert[r] \times horz[c], \qquad (24.1)$$

где $x[r,c]$ – двумерное изображение, а $vert[r]$ и $horz[c]$ – одномерные проекции. Очевидно, что большинство изображений не удовлетворяют этому требованию. Например, пилюлеобразная ФРТ не сепарабельна. Однако существует *бесконечное* число сепарабельных изображений. Это можно понять, создавая произвольные горизонтальные и вертикальные проекции и находя соответствующие им изображения. Например, на рис. 24.6 приводится иллюстрация этого для профилей, представляющих собой двустороннюю экспоненту. Изображение, соответствующее этим профилям, находится затем по уравнению (24.1). При демонстрации, изображение выглядит как экспоненциально затухающая до нуля, по мере увеличения расстояния от начала, бриллиантовая кривая.

В большинстве задач обработки идеальная ФРТ обладает *круговой симметрией* наподобие пилюлеобразной. Желательно модифицировать изображение во всех направлениях таким же образом, даже притом, что оцифрованные изображения обычно хранятся и обрабатываются в прямоугольном формате в виде строк и столбцов,. Это приводит к вопросу: существует ли ФРТ, обладающая круговой симметрией *и* являющаяся сепарабельной. Ответом будет - да, но только одна, и это *Гауссиан*. Как показано на рис. 24.7, двумерное Гауссово изображение имеет проекции, которые также являются

Гауссианами. Гауссианы изображения и проекций имеют *такое же* стандартное отклонение.

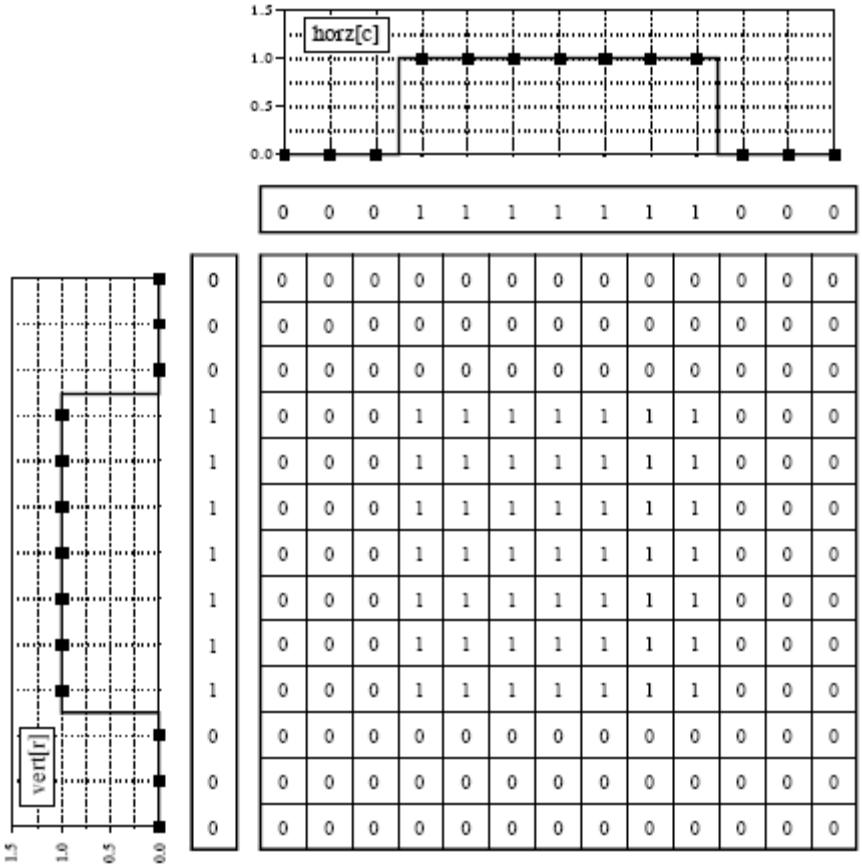


Рис. 24.5 Разделение прямоугольной ФРТ

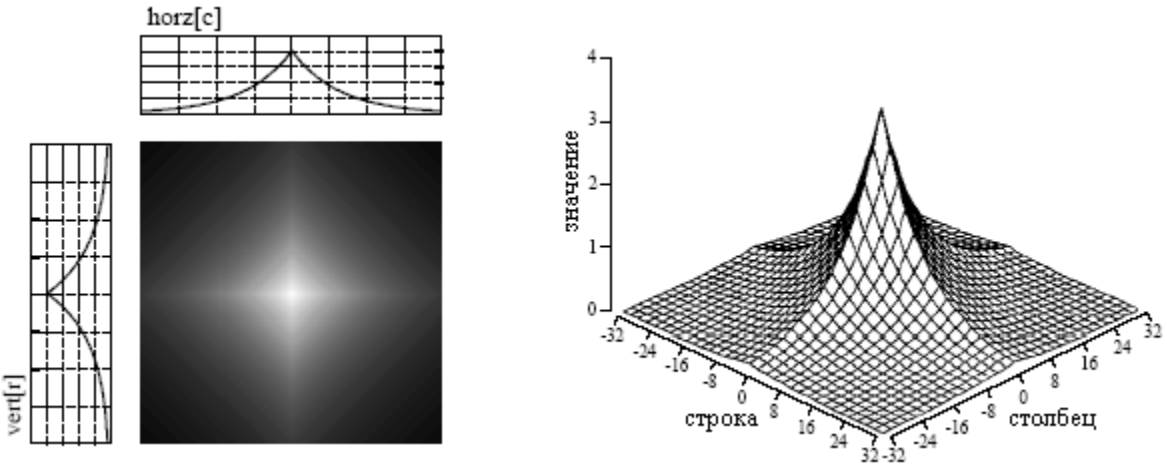
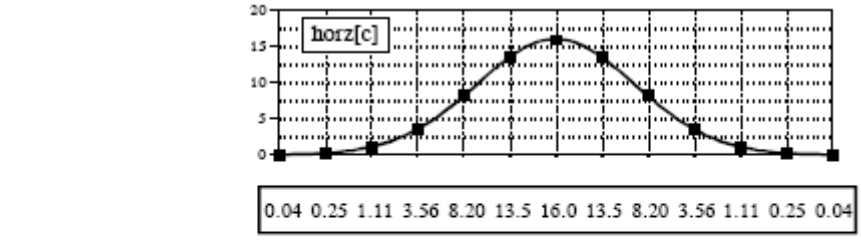


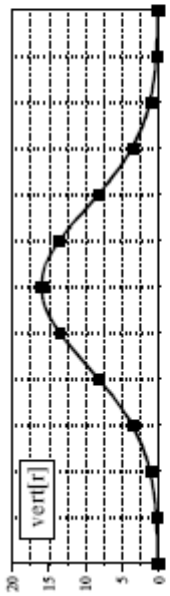
Рис. 24.6 Создание сепарабельной ФРТ

Для осуществления свертки изображения с сепарабельным ядром фильтра выполняется свертка каждой *строки* в изображении с *горизонтальной проекцией* и получается промежуточное изображение. Затем, выполняется свертка каждого *столбца* этого промежуточного изображения с *вертикальной проекцией* ФРТ. Полученное в результате изображение идентично непосредственной свертке исходного изображения с

ядром фильтра. Если Вам нравится, выполняйте сначала свертку столбцов, а затем строк результат будет таким же.



0.04	0.25	1.11	3.56	8.20	13.5	16.0	13.5	8.20	3.56	1.11	0.25	0.04



0.04	0	0	0	0	0	1	1	1	0	0	0	0	0
0.25	0	0	0	1	2	3	4	3	2	1	0	0	0
1.11	0	0	1	4	9	15	18	15	9	4	1	0	0
3.56	0	1	4	13	29	48	57	48	29	13	4	1	0
8.20	0	2	9	29	67	111	131	111	67	29	9	2	0
13.5	1	3	15	48	111	183	216	183	111	48	15	3	1
16.0	1	4	18	57	131	216	255	216	131	57	18	4	1
13.5	1	3	15	48	111	183	216	183	111	48	15	3	1
8.20	0	2	9	29	67	111	131	111	67	29	9	2	0
3.56	0	1	4	13	29	48	57	48	29	13	4	1	0
1.11	0	0	1	4	9	15	18	15	9	4	1	0	0
0.25	0	0	0	1	2	3	4	3	2	1	0	0	0
0.04	0	0	0	0	0	1	1	1	0	0	0	0	0

Рис. 24.7 Разделение Гауссиана

Свертка изображения размером $N \times N$ с ядром фильтра размером $M \times M$ потребует времени пропорционально N^2M^2. Другими словами, каждый пиксель в выходном изображении зависит от *всех* пикселей в ядре фильтра. Для сравнения, свертка разделением потребует времени пропорционально только N^2M. Для ядер фильтров шириной в сотни пикселей этот метод приведет к снижению времени исполнения в *сотни* раз.

Дело может быть даже лучше. Вычисления становятся еще более эффективными, если Вы пожелаете использовать *прямоугольную* ФРТ (рис. 24.5) или двусторонне экспоненциальную ФРТ (рис. 24.6). Это происходит потому, что одномерная свертка может быть реализована соответственно, как фильтр *скользящего среднего* (Глава15) и *двунаправленный однополюсный* фильтр (Глава 19). Оба этих одномерных фильтра могут быть быстро реализованы с помощью рекурсии. Это дает время свертки изображения, пропорциональное всего N^2, что полностью не зависит от размера ФРТ. Другими словами, с помощью всего нескольких целочисленных операций на пиксель можно осуществить свертку изображения с такой большой ФРТ, насколько потребуется. Например, свертка изображения 512×512 на персональном компьютере требует всего нескольких сотен миллисекунд. Это быстро! Не нравится форма ядер этих двух фильтров? Для аппроксимации Гауссовой ФРТ, выполните свертку изображения с одной из них *несколько раз* (гарантировано центральной предельной теоремой, Глава 7). Это великие алгоритмы, способные вырвать успех из челюстей неудачи. Их стоит хорошо запомнить.

Пример большой ФРТ: Сглаживание освещенности

Типичные приложения требуют больших ФРТ, улучшающих изображения с неравномерной освещенностью. Свертка разделением представляет собой идеальный алгоритм для выполнения такой обработки. За небольшим исключением видимые глазом изображения формируются из *отраженного* света. Это означает, что наблюдаемое изображение эквивалентно отражающей способности объектов, умноженной на окружающую освещенность. На рис. 24.8 показано, как это все действует. Рис. 24.8a представляет *отражающую* способность наблюдаемой сцены, в данном случае ряд светлых и темных полос. Рис. 28.4b иллюстрирует пример сигнала освещенности светового рисунка, падающего на рис. 28.4a. Как и в реальном мире, освещенность медленно изменяется вдоль наблюдаемой области. На рис. 24.8c приведено изображение того, что видит глаз, эквивалент изображения отражающей способности (рис. 28.4a), умноженного на изображение освещенности (рис. 24.8b). На рис. 24.8c области слабой освещенности трудно различимы по двум причинам: они слишком темны и их контрастность слишком низка (разница между пиками и седловинами).

Чтобы понять, как это связано с задачами повседневного зрения, представьте, что Вы рассматриваете двух одинаково одетых людей. Один из них стоит под ярким солнечным светом, в то время как другой стоит под деревом в тени. Процент отраженного от обоих мужчин падающего света один и тот же. Например, их лица могут отражать 80% падающего света, их серые рубашки - 40%, а их темные брюки - 5%. Проблема состоит в том, что освещение их обоих могло бы отличаться, скажем, раз в десять. Это делает изображение человека в тени в десять раз темнее, чем персоны на солнце, и в десять раз менее контрастным (между лицом, рубашкой и брюками).

Целью обработки изображения является *сглаживание* в полученном изображении составляющей освещенности. Другими словами, мы желаем, чтобы конечное изображение было представлено отражающей способностью объектов, а не условиями освещенности. В терминах рис. 24.8: дано рис. 24.8c, найти рис. 24.8a. Это задача нелинейной фильтрации, поскольку компоненты изображения были объединены умножением, а не сложением. Хотя такое разделение не может быть выполнено абсолютно корректно, улучшение может быть впечатляюще серьезным.

Для начала, выполним свертку изображения на рис. 24.8c с большой ФРТ размером в одну пятую от всего изображения. Целью является устранение резких переходов на рис. 24.8c, дающее аппроксимацию исходного сигнала освещенности на рис. 24.8b. Вот здесь и используется свертка разделением. Точная форма ФРТ не важна, единственно, что она должна быть значительно шире, чем переходы в отраженном изображении. На рис. 24.8d показан результат использования в качестве ядра фильтра Гауссиана.

Поскольку сглаживающий фильтр дает лишь оценку изображения освещенности, для нахождения отраженного изображения мы будем использовать фильтр, улучшающий края. То есть, выполним свертку изображения на рис. 24.8c с ядром фильтра, состоящим из дельта функции минус Гауссиан. Для снижения времени исполнения, это осуществляется с помощью вычитания сглаженного изображения на рис. 24.8d из исходного изображения на рис. 24.8c. Результат показан на рис. 24.8e. Это не работает! Хотя темные области были должным образом подсвечены, контрастность в этих областях остается ужасной.

Линейная фильтрация в таких приложениях работает плохо, поскольку отраженный сигнал и сигнал освещенности были изначально объединены умножением, а не сложением. Линейная фильтрация не может корректно разделить сигналы, объединенные с помощью нелинейной операции. Для разделения таких сигналов с ними следует выполнить операцию *обратную* умножению. Другими словами, исходное

изображение необходимо *делить* на сглаженное изображение, что и показано на рис. 24.8f. Это исправляет яркость и восстанавливает контрастность до надлежащего уровня.

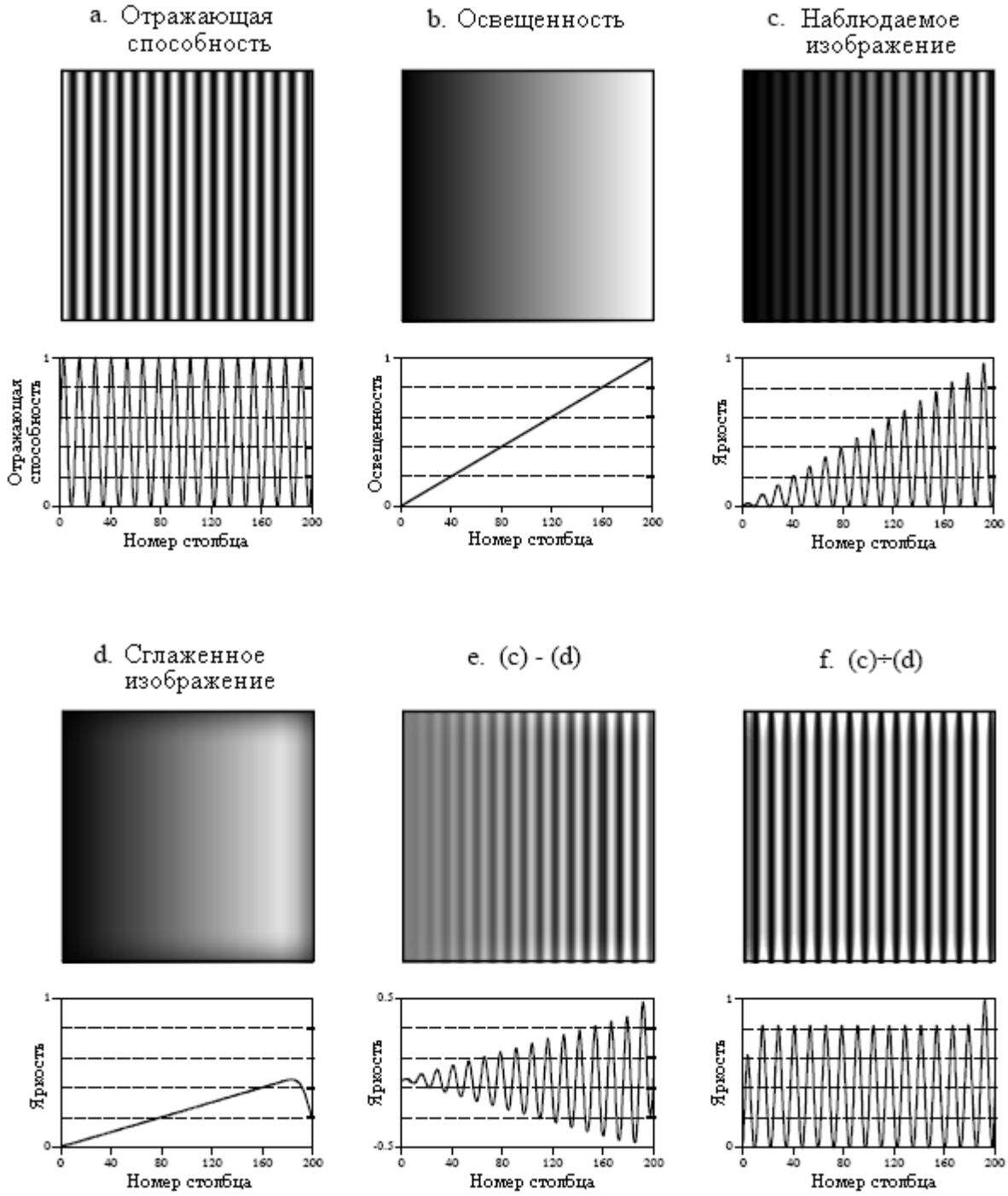


Рис. 24.8 Модель формирования изображения

Эта процедура деления изображений друг на друга тесно связана с **гомоморфной обработкой**, ранее описанной в Главе 22. Гомоморфная обработка представляет собой способ обработки сигналов, объединенных посредством нелинейных операций. Стратегия здесь заключается в том, чтобы при помощи соответствующих математических операций заменить нелинейную задачу линейной. Когда два сигнала объединяются посредством умножения, гомоморфная обработка начинается с *логарифмирования* полученного сигнала. Благодаря тождеству: $log(a \times b) = log(a) + log(b)$ задача разделения *перемноженных* сигналов превращается в задачу разделения *сложенных* сигналов.

Например, после логарифмирования изображения на рис. 24.8c для выделения отраженного изображения может быть использован линейный высокочастотный фильтр. Как и раньше наиболее быстрым способом реализации высокочастотного фильтра является вычитание сглаженной версии изображения. Затем, для того чтобы избавиться от логарифма, используется антилогарифм (возведение основания в соответствующую степень), дающий желаемую аппроксимацию отраженного изображения.

Что лучше, делить или идти по пути гомоморфного анализа? Они почти одинаковы, поскольку взятие логарифма с последующим вычитанием *эквивалентно* делению. Единственным отличием является используемая для освещенности изображения аппроксимация. Один метод использует сглаженную версию полученного изображения, тогда как другой метод использует сглаженную версию логарифма полученного изображения.

Такая техника выравнивания сигнала освещенности настолько полезна, что она включена в нейронную структуру глаза. Обработка в среднем слое сетчатки была описана ранее, как улучшение краев или высокочастотная фильтрация. Хотя это и правда, но это только половина правды. Первый слой в глазу является нелинейным, аппроксимированным на взятие *логарифма* от поступающего изображения. Это превращает глаз в гомоморфный процессор. Точно так же, как описывалось выше, за логарифмированием следует сглаживание составляющей освещенности при помощи улучшающей края линейной фильтрации, позволяющей глазу видеть в условиях плохой освещенности. Другое интересное использование гомоморфной обработки встречается в фотографии. Плотность (затемненность) негатива эквивалентна логарифму яркости конечной фотографии. Это означает, что любые манипуляции с негативом во время проявления представляют собой вид гомоморфной обработки.

Перед тем как закончить с этим примером, следует отметить еще один нюанс. Как обсуждалось в Главе 6, при осуществлении свертки N точечного сигнала с M точечным ядром фильтра результирующий сигнал становится $N+M-1$ точек длинной. Аналогично, когда осуществляется свертка изображения размером $N\times N$ с ядром фильтра размером $M\times M$, результатом является изображение $(N+M-1)\times(N+M-1)$. Проблема в том, что часто трудно управлять изменяющимся размером изображения. Например, должна измениться выделенная память, должен быть подрегулирован видеодисплей, может потребоваться изменение индексации массива и т.д. Самый простой способ обойти это – *игнорировать* это; если мы начали с изображения 512×512, мы желаем и закончить изображением 512×512. Пиксели, которые не вписываются в пределы первоначальных границ, отбрасываются.

Хотя это и сохраняет размеры изображения такими же, это не решает всей проблемы; здесь еще остаются *обстоятельства границ* для свертки. Например, представьте попытку вычислить пиксель в верхнем правом углу на рис. 24.8d. Это осуществляется расположением центра Гауссовой ФРТ в правом верхнем углу на рис. 24.8c. Затем, каждый пиксель на рис. 24.8c умножается на соответствующий пиксель в перекрывающейся части ФРТ, а результаты произведений складываются. Проблема в том, что три четверти ФРТ лежат за пределами заданного изображения. Самым легким подходом является приписывание значения нуля незаданным пикселям. Именно так был создан рис. 24.8d, очерчиванием темной полоски по периметру изображения. То есть, яркость плавно уменьшается до нулевого значения пикселя для множества внешних точек по отношению к заданному изображению.

К счастью, эта темная область, окаймляющая границы, может быть скорректирована (хотя в этом примере этого сделано не было). Это делается делением каждого пикселя на рис. 24.8d на корректирующий коэффициент. Корректирующий коэффициент является частью ФРТ, перекрывающейся с входным изображением на момент вычисления пикселя. То есть, для корректировки отдельного пикселя на рис. 24.8d, представьте, что ФРТ отцентрирована над соответствующим пикселем на рис.

24.8c. Например, самый правый верхний пиксель на рис. 24.8c дает всего 25% перекрытия ФРТ с входным изображением. Следовательно, скорректируйте этот пиксель на рис. 24.8d, разделив его на коэффициент 0,25. Это означает, что пиксели в центре рис. 24.8d не изменятся, но темные пиксели по периметру станут светлее. Для нахождения корректирующих коэффициентов представьте, свертку ядра фильтра с изображением, у которого все значения пикселей равны *единице*. Пиксели в результирующем изображении и есть корректирующие коэффициенты необходимые для устранения краевых эффектов.

Фурье анализ изображения

Анализ Фурье используется в обработке изображений почти таким же образом, как и в случае с одномерными сигналами. Однако изображения не содержат информации, закодированной в частотной области, что делает этот метод значительно менее полезным. Например, когда от *звукового* сигнала берется преобразование Фурье, то беспорядочная во временной области форма волны преобразуется в легкий для понимания частотный спектр. Для сравнения, взятие преобразования Фурье от изображения преобразует очевидную информацию пространственной области в мешанину частотной области. Короче, не ждите, что преобразование Фурье поможет Вам понять закодированную в изображении информацию.

Аналогичным образом, не обращайтесь к частотной области для проектирования фильтра. Основной характеристикой изображений являются края, линии - отделяющие один *объект* или *область* от другого *объекта* или *области*. Так как край состоит из широкого диапазона частотных составляющих, попытки модификации изображения, посредством манипулирования частотным спектром являются в основном непродуктивными. Обычно фильтры изображения проектируются в пространственной области, в которой информация закодирована в ее наиболее простейшей форме. Мыслите терминами операций *сглаживания* и *улучшения краев* (пространственная область), а не терминами *высокочастотных* и *низкочастотных* фильтров (частотная область).

Несмотря на это, Фурье анализ изображения обладает несколькими полезными свойствами. Например, *свертка* в пространственной области соответствует *умножению* в частотной области. Это важно, поскольку умножение представляет собой более простую математическую операцию, чем свертка. Как и в случае одномерных сигналов, это свойство позволяет реализовать БПФ свертку и различные методы обратной свертки. Другим полезным свойством частотной области является *теорема среза Фурье*, соотношение между изображением и его проекциями (изображение рассматривается с разных сторон). Это основы *компьютерной томографии*, рентгеновского метода отображения широко применяемого в медицине и промышленности.

Частотный спектр изображения может быть вычислен несколькими способами, но представленный здесь метод БПФ является единственным применяемым на практике методом. Исходное изображение должно состоять из *N* строк и *N* столбцов, где *N* кратно двум в соответствующей степени, т.е. 256, 512, 1024 и т.д. Если размер исходного изображения не кратен двойке в соответствующей степени, то для того, чтобы размер сделать правильным, добавляются пиксели с нулевым значением. Мы будем называть двумерный массив, содержащий изображение **действительным массивом**. Дополнительно потребуется еще один массив такого же размера, который мы будем называть **мнимым массивом**.

Рецепт вычисления преобразования Фурье изображения весьма прост: возьмите одномерное БПФ от каждой строки, после чего возьмите одномерное БПФ от каждого из столбцов. Начните взятие БПФ с *N* значений пикселей в строке *0* действительного массива. Действительная часть с выхода БПФ помещается назад в строку *0* действительного массива, тогда как мнимая часть с выхода БПФ помещается в строку *0*

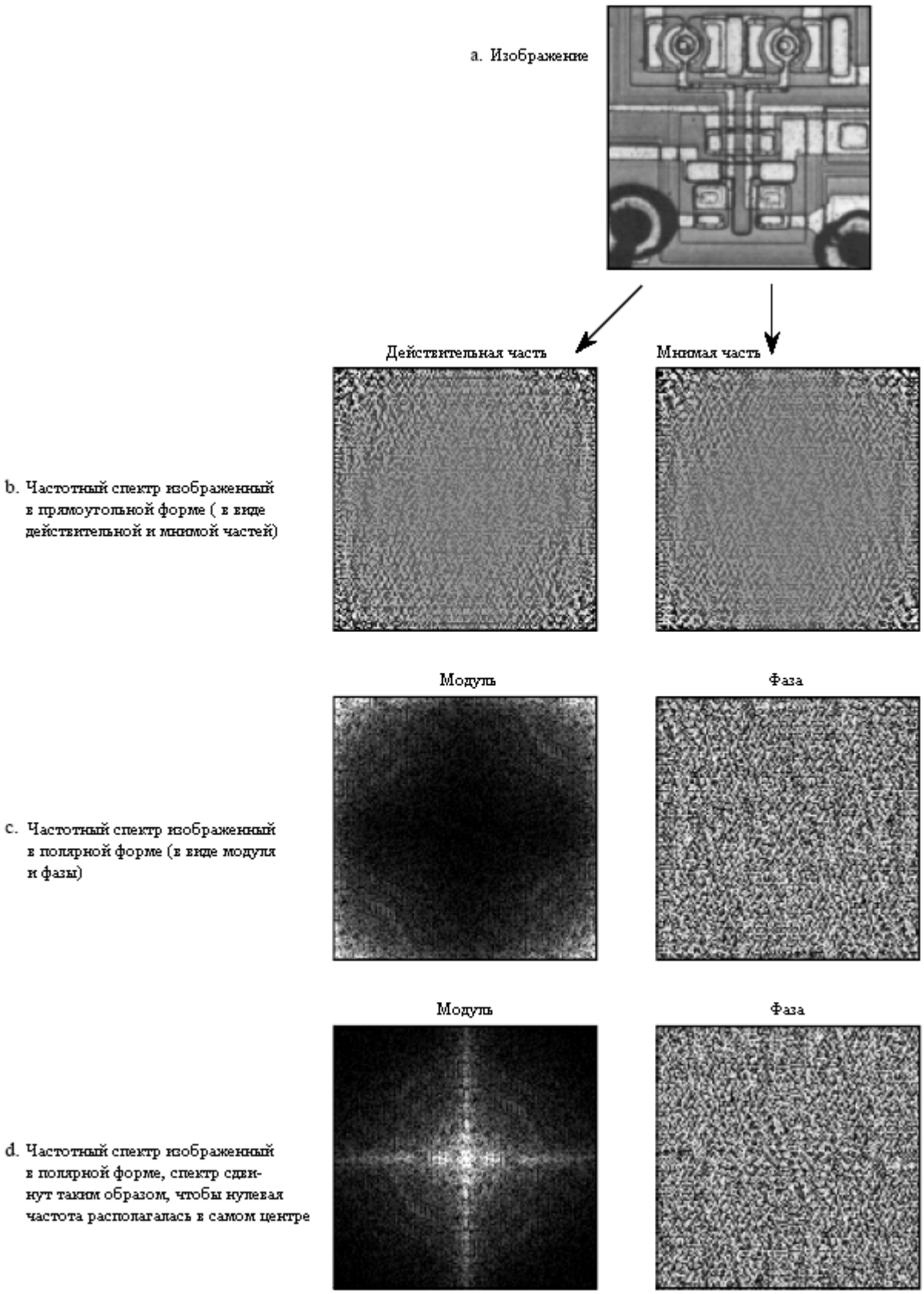


Рис. 24.9 Частотный спектр изображения

мнимого массива. После повторения этой процедуры со строками от *1* до *N-1*, оба действительный и мнимый массивы содержат промежуточное изображение. Затем, процедура повторяется с каждым *столбцом* промежуточных данных. Возьмите значения

N пикселей в столбце *0* действительного массива *и* значения *N* пикселей в столбце *0* мнимого массива и вычислите БПФ. Действительная часть с выхода БПФ помещается назад в столбец *0* действительного массива, тогда как мнимая часть с выхода БПФ помещается в столбец *0* мнимого массива. После повторения этой процедуры со столбцами от *1* до *N-1*, оба массива заполнены частотным спектром изображения.

Поскольку в изображении вертикальное и горизонтальное направления одинаковы, этот алгоритм может также быть выполнен сначала преобразованием столбцов, а затем преобразованием строк. Вне зависимости от используемого порядка, результат будет один и тот же. В соответствии с порядком обработки данных, которого придерживается БПФ, амплитуды низкочастотных составляющих располагаются по углам двумерного спектра, тогда как высокие частоты помещаются в центре. Обратное преобразование Фурье от изображения вычисляется взятием обратного БПФ от каждой строки, за чем следует обратное БПФ от каждого столбца (или наоборот).

На рис. 24.9 показан пример преобразования Фурье от изображения. Рис. 24.9a представляет исходное изображение, вид под микроскопом входного каскада интегральной схемы операционного усилителя μA741 (отечественным аналогом является микросхема КР140УД7 – прим. перев.). На рис. 24.9b показаны действительная и мнимая части частотного спектра этого изображения. Так как частотная область может содержать отрицательные значения пикселей, значения шкалы серого для этого изображения смещены таким образом, что отрицательные значения представляются темным, нулевые серым, а положительные значения светлым. Обычно низкочастотные составляющие изображения по амплитуде значительно больше высокочастотных составляющих. Это объясняет самые яркие и самые темные пиксели в четырех углах на рис. 24.9b. Кроме этого, спектры *типичных* изображений не имеют никакого видимого порядка, выглядя хаотично. Конечно же, можно *придумать* изображения, имеющие любой спектр, который Вы пожелаете.

Как показано на рис. 24.9c, полярная форма спектра изображения лишь чуть-чуть проще для понимания. Более низкие частоты по модулю имеют большие положительные значения (белые углы), тогда как высокие частоты имеют маленькие положительные значения (черный центр). На низких и на высоких частотах фаза выглядит одинаково, кажется случайно меняющейся между $-\pi$ и π радиан.

На рис. 24.9d показан альтернативный способ демонстрации изображения спектра. Так как пространственная область содержит *дискретные* сигналы, частотная область является *периодической*. Другими словами, массивы частотной области бесконечное число раз дублируются влево, вправо, вверх и вниз. Например, представьте стену, покрытую кафельной плиткой, где каждая плитка представляет собой $N \times N$ модулей показанных на рис. 24.9c. Рис. 24.9d также представляет собой часть этой стены размером $N \times N$, покрытой кафельной плиткой, но он располагается на четырех кафельных плитках с центром изображения, находящемся в точке соприкосновения этих плиток. Другими словами, рис. 24.9c представляет собой то же самое изображение, что и рис. 24.9d, за исключением того, что в периодическом частотном спектре он был сдвинут на *N/2* пикселей по горизонтали (либо влево, либо вправо) и на *N/2* пикселей по вертикали (либо вверх, либо вниз). Это собирает яркие пиксели с четырех углов на рис. 24.9c вместе, в центре на рис. 24.9d.

Рис. 24.10 иллюстрирует то, как организована частотная область (низкие частоты расположены по углам). Строка с индексом *N/2* и столбец с индексом *N/2* разбивают частотный спектр на четыре квадранта. Для действительной части и модуля верхний правый квадрант является зеркальным отражением левого нижнего, тогда как верхний левый является зеркальным отражением правого нижнего. Такой же симметрией обладают так же мнимая часть и фаза, за исключением того, что значения зеркальных пикселей противоположны по знаку. Другими словами, каждая точка в частотном спектре имеет соответствующую ей точку, симметрично расположенную на противоположной стороне

от центра изображения (строка *N/2* и столбец *N/2*).Одни из точек соответствуют *положительным* частотам, тогда как другие соответствуют *отрицательным* частотам. В форме уравнения такая симметрия выражается как:

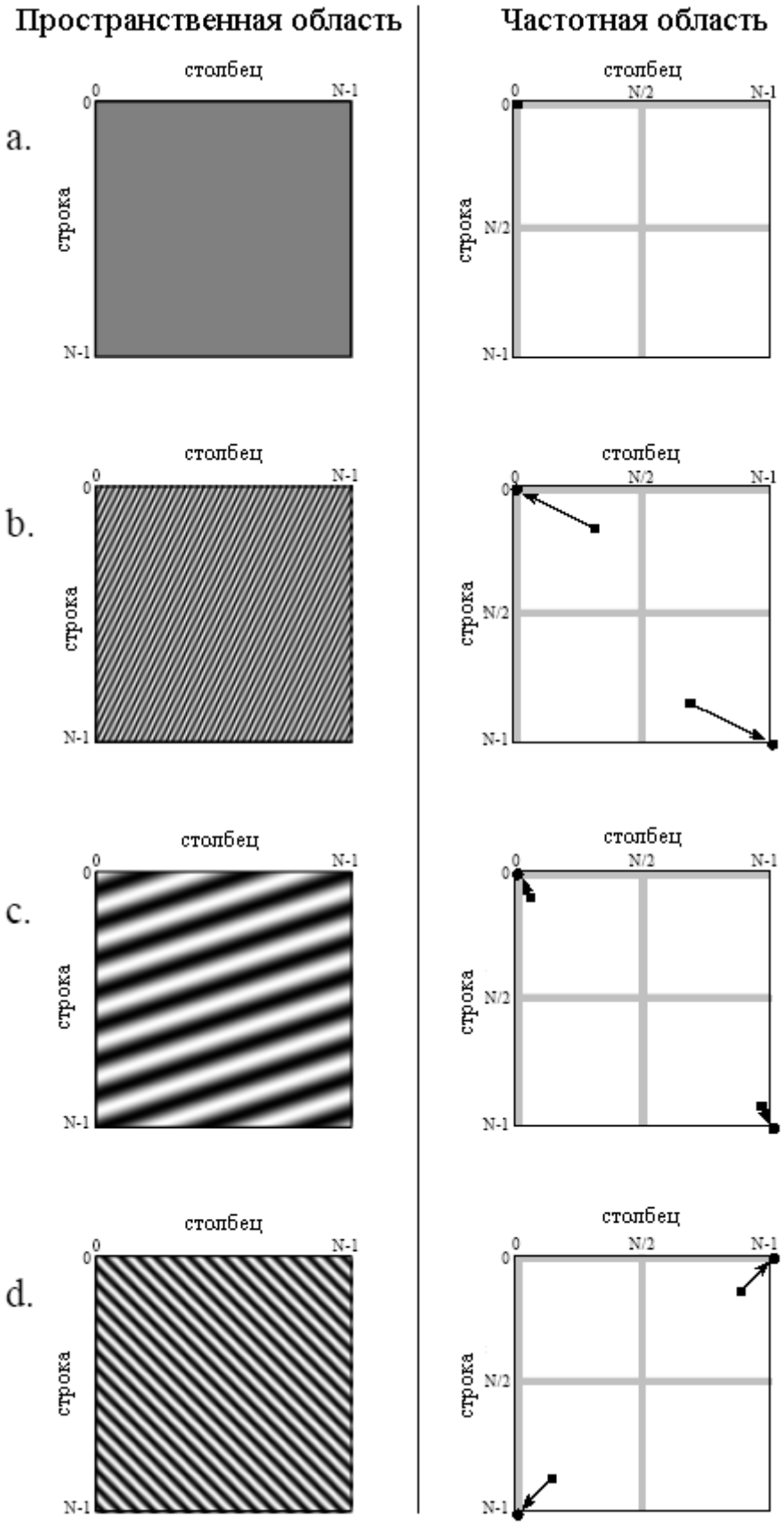


Рис. 24.10 Двумерные синусоиды

$$\begin{aligned}\operatorname{Re} X[r,c] &= \operatorname{Re} X[N-r,N-c],\\ \operatorname{Im} X[r,c] &= \operatorname{Im} X[N-r,N-c].\end{aligned} \tag{24.2}$$

Это уравнение учитывает, что частотный спектр является периодическим с изменяющимися от *0* до *N-1* индексами и повторяющимся каждые *N* отсчетов. Другими словами, *X[r,N]* следует интерпретировать как *X[r,0]*, а *X[N,c]* интерпретировать как *X[0,c]* и *X[N,N]* интерпретировать как *X[0,0]*. Такая симметрия создает в спектре четыре точки соответствующих *самим себе*. Эти точки имеют следующее местоположение: *[0,0]*, *[0,N/2]*, *[N/2,0*] и *[N/2,N/2]*.

Каждая пара точек в частотной области соответствует синусоиде в пространственной области. Как показано на рис. 24.10a, значение при *[0,0]* соответствует синусоиде в пространственной области с *нулевой частотой*, т.е. постоянной составляющей изображения. На этом рисунке показана только одна точка, поскольку это одна из точек, которая соответствует сама себе. Как показано на рис. 24.10b, рис. 24.10c и рис. 24.10d, другие пары точек соответствуют двумерным синусоидам, которые выглядят как волны в океане. У одномерных синусоид есть *частота*, *фаза* и *амплитуда*. Двумерные синусоиды обладают еще и *направлением*.

Частота и направление каждой из синусоид определяются местоположением пары точек в частотной области. Проведите, как показано, линию из каждой точки в сторону местоположения с *нулевой частотой* к внешнему углу квадранта, в котором находится точка, т.е. *[0,0]*, *[0,N-1]*, *[N-1,0]* или *[N-1,N-1]* (на этом рисунке показаны кружочками). Направление этой линии определяет направление пространственной синусоиды, тогда как ее длина пропорциональна частоте волны. В результате низкие частоты располагаются вблизи углов, а высокие частоты вблизи центра.

Когда спектр изображается с нулевой частотой расположенной в самом центре (рис. 24.9d), линия из каждой пары точек проводится в сторону значения постоянной составляющей к *центру* изображения, т.е. *[N/2,N/2]*. Такая организация проще для понимания и работы, поскольку все линии проводятся в одну и туже точку. Другое преимущество размещения нуля в центре состоит в том, что это соответствует частотным спектрам *непрерывных* изображений. Когда пространственная область непрерывна частотная область *апериодическая*. При этом нулевая частота находится в центре при нарастании частоты до бесконечности во всех направлениях от центра. Вообще, всякий раз, когда данные будут просматривать люди, в учебниках, в журнальных статьях и алгоритмической документации, частотные спектры дискретных изображений отображаются с нулевой частотой в центре. Однако большинство вычислений производятся с компьютерными массивами, хранящими данные в других форматах (низкие частоты располагаются по углам). Это связано с тем, что у БПФ именно такой формат.

Даже при использовании БПФ время, требуемое для вычисления преобразования Фурье чрезвычайно узкое место в обработке изображений. Например, преобразование Фурье от изображения размером 512×512 на персональном компьютере требует нескольких минут. Грубо, это в 10000 раз медленнее, чем необходимо для обработки изображения в режиме реального времени - 30 кадров в секунду. Такое большое время исполнения - следствие огромного количества информации содержащейся в изображении. Для сравнения, в обычном изображении приблизительно такое же число *пикселей*, как и слов в этой книге. По мере того как компьютеры будут становиться быстрее, обработка изображений через частотную область будет становиться более популярной. Это технология двадцать первого века; следите за ее появлением!

БПФ свертка

Даже притом, что преобразование Фурье медленно, оно остается самым быстрым способом свертки изображения с большим ядром фильтра. Например, по сравнению с традиционной сверткой, свертка изображения размером 512×512 с ФРТ размером 50×50 при использовании БПФ приблизительно в 20 раз быстрее. Как работает БПФ свертка для одномерных сигналов, обсуждается в Главе 18. Двумерная версия является простым расширением этого.

Рис. 24.11 Пример обнаружения цели

Продемонстрируем БПФ свертку на примере алгоритма отыскивающего в изображении предварительно заданный образец. Предположим, что мы строим систему для проверки купюр достоинством в один доллар, наподобие тех, которые могут быть использованы при проверке качества печати, обнаружения подделок или проверки оплаты в торговых автоматах. Как показано на рис. 24.11, с купюры получено изображение размером 100×100 пикселей с портретом Джорджа Вашингтона в центре. Целью исследования этого изображения является поиск известного образца, в нашем примере это изображение лица размером 29×29 пикселей. Проблема заключается в следующем: даны полученное изображение и известный образец, какой способ является наиболее эффективным для определения того, где находится (и находится ли вообще) образец на изображении? Если Вы уделили внимание Главе 7, то Вы знаете, что путем к решению этой проблемы является *корреляция* (согласованная фильтрация) и, что она может быть выполнена с помощью *свертки*.

Рис. 24.12 Развитие ядра корреляционного фильтра

Перед выполнением собственно свертки для преобразования изображения цели в ФРТ, необходимо выполнить две модификации. Они показаны на рис. 24.12. На рис. 24.12a показан сигнал цели, образец который мы пытаемся обнаружить. На рис. 24.12b изображение было повернуто на 180^0, это то же самое, что и переворот слева направо, а затем сверху вниз. Как обсуждалось в Главе 7, при выполнении *корреляции* с помощью *свертки*, сигнал цели должен быть перевернут для того, чтобы компенсировать переворот, происходящий при выполнении операции свертки. Вскоре мы вернемся к этой проблеме.

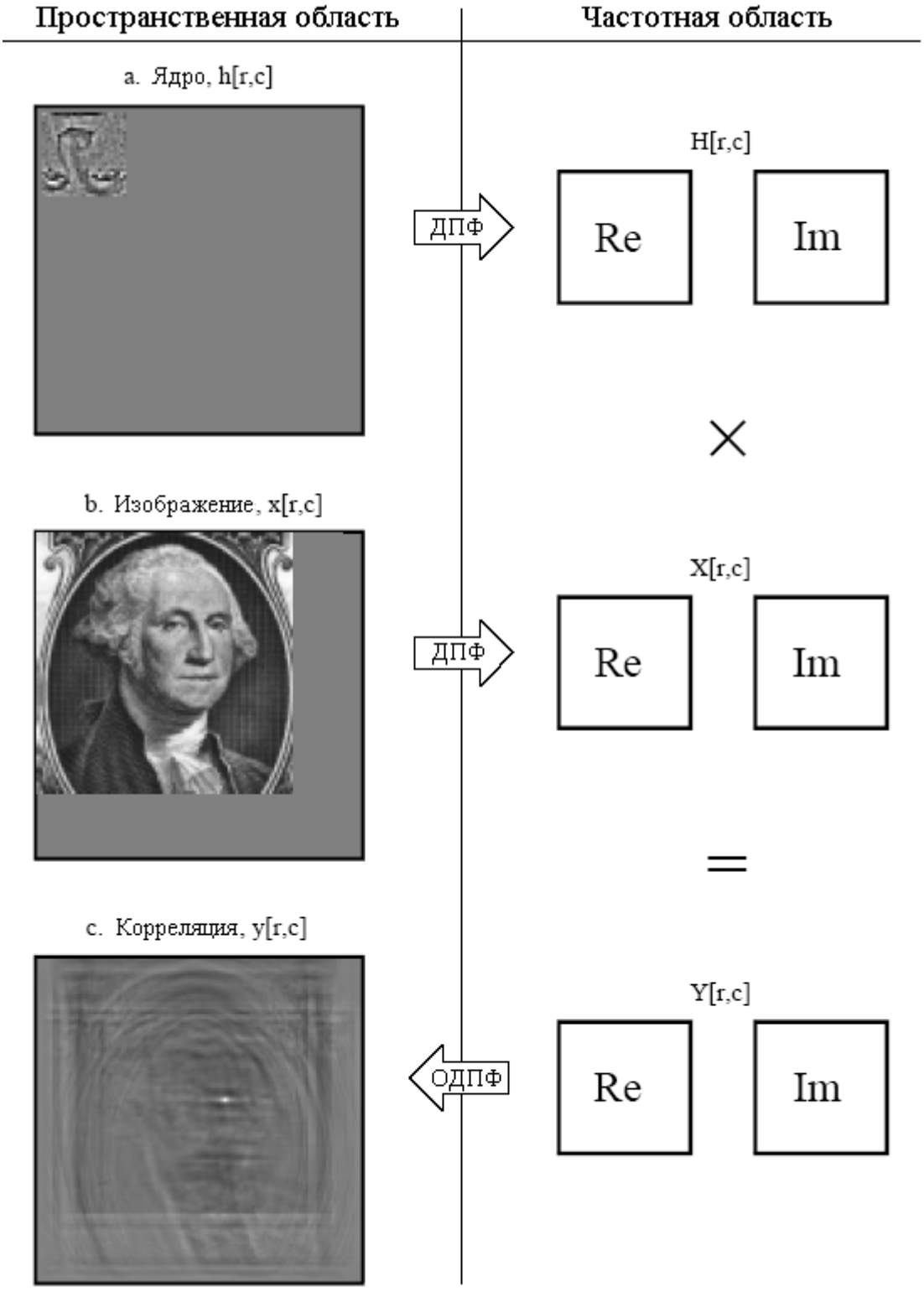


Рис. 24.13 Структурная схема БПФ свертки изображения

Вторая модификация – это остроумный прием для улучшения эффективности алгоритма. Вместо того, чтобы пытаться обнаружить лицо в исходном изображении, значительно более эффективно обнаруживать *края лица* в *краях исходного изображения* (*контуры лица* в *контурах исходного изображения* – прим. перев.). Это связано с тем, что края острее (резче – прим. перев.) чем исходные черты, что делает корреляционный пик острее. Этот шаг необязателен, но он значительно улучшает результат. В наиболее простой форме, до выполнения корреляции исходное изображение и сигнал цели пропускаются через обнаруживающий край фильтр размером 3×3. В соответствии со свойством ассоциативности свертки это равносильно пропусканию сигнала цели через обнаруживающий край фильтр *дважды*, не трогая при этом исходного изображения. В действительной практике использование обнаруживающего край ядра размером 3×3 всего один раз вообще достаточно. Именно так из рис. 24.12b был получен рис. 24.12c. Это сделало изображение на рис. 24.12c функцией размыва точки, используемой для выполнения свертки.

Рис. 24.13 иллюстрирует детали БПФ свертки. В этом примере мы осуществим свертку изображения на рис. 24.13a с изображением на рис. 24.13b, для того, чтобы получить изображение на рис. 24.13c. Тот факт, что эти изображения были отобраны, и для того, чтобы выполнить *корреляцию*, предварительно обработаны - не имеет значения; это – структурная схема *свертки*. Первым шагом является дополнение обоих сигналов, свертка которых будет осуществляться, нулями, для того чтобы раза в два увеличить их размер и сделать их достаточно большими для размещения конечного изображения. Например, при осуществлении свертки двух изображений размерами 100×100 и 29×29 пикселей результирующее изображение будет иметь размер 128×128 пикселей. Следовательно, к изображениям на рис. 23.13a и рис. 23.13b должно быть добавлено такое количество нулей, чтобы сделать их размером 128×128 пикселей каждое. Если этого не сделать, будет иметь место круговая свертка и конечное изображение будет искажено. Если у Вас есть ощущение непонимания этой концепции, вернитесь назад и посмотрите Главу 18, где более детально обсуждается одномерный случай.

Для преобразования изображений на рис. 23.13a и рис. 23.13b в частотную область используется алгоритм БПФ. Это дает *четыре* массива размером 128×128 - действительные и мнимые части изображений, свертка которых будет осуществляться. Перемножение действительной и мнимой частей изображения на рис. 24.13a с действительной и мнимой частями изображения на рис. 24.13b дает действительную и мнимую части изображения на рис. 24.13c. (Если Вам нужно вспомнить, как это делается, посмотрите уравнение (9.1).) Взятие обратного БПФ завершает алгоритм, давая конечное свернутое изображение.

Значение каждого пикселя в изображении корреляции является мерой того, насколько хорошо *в данной точке* изображение цели соответствует исследуемому изображению. В этом конкретном примере изображение корреляции на рис. 24.13c состоит из шума плюс яркого одиночного пика, показывающего хорошее соответствие сигналу цели. На обнаруженные координаты лица указывает просто расположение в этом изображении самого яркого пикселя. Если бы мы не использовали модификацию обнаружения края в сигнале цели, пик продолжал бы присутствовать, но был бы, намного менее различим.

Хотя корреляция является мощным средством обработки изображения, ей присущи значительные ограничения: изображение цели должно быть точно такого же *размера* и так же *повернуто*, как и соответствующая область в исследуемом изображении. Шумы и другие изменения амплитуды каждого пикселя относительно неважны, но точное пространственное соответствие является критичным. Последнее делает этот метод почти бесполезным, например, для обнаружения вражеских танков на военных разведывательных фотографиях, опухолей на медицинских снимках и пистолетов при

просмотре багажа в аэропортах. Единственный подход заключается в том, чтобы осуществлять *многократную* корреляцию исследуемого изображения с изображением цели разнообразных размеров и с разнообразными поворотами. В принципе это работает, но время исполнения очень быстро приведет Вас к потере всякого интереса к этому.

Более близкий взгляд на свертку изображения

Давайте используем этот предыдущий пример для более детального исследования двумерной свертки. Так же, как и в случае одномерных сигналов свертка изображения может рассматриваться как с *входной стороны*, так и с *выходной стороны*. Как вы помните из Главы 6, взгляд на свертку с точки зрения входного сигнала представляет собой лучшее описание того, как работает свертка, тогда как взгляд на свертку с точки зрения выходного сигнала представляет собой то, как написаны большая часть математики и алгоритмов. Вам следует чувствовать себя комфортно с обоими этими взглядами на операцию.

На рис. 24.14 приведено описание свертки изображения с точки зрения входного сигнала. Каждый пиксель во входном изображении дает смасштабированную и сдвинутую ФРТ, добавляемую к выходному изображению. Тогда выходное изображение вычисляется как сумма вкладов от всех ФРТ. Этот рисунок показывает вклад в выходное изображение от точки с местоположением [*r*,*c*] во входном изображении. ФРТ сдвигается таким образом, что пиксель [*0*,*0*] в ФРТ совмещается с пикселем [*r*,*c*] в выходном изображении. Если ФРТ, как в данном примере, определена только положительными индексами, сдвинутая ФРТ будет полностью располагаться ниже и правее точки [*r*,*c*]. Не запутайтесь, увидев появившееся на этом рисунке перевернутое сверху вниз лицо; это перевернутое сверху вниз лицо *и есть* ФРТ, которую мы используем в данном примере (рис. 24.13а). Во взгляде на свертку с точки зрения входного сигнала поворот ФРТ отсутствует, она просто сдвигается.

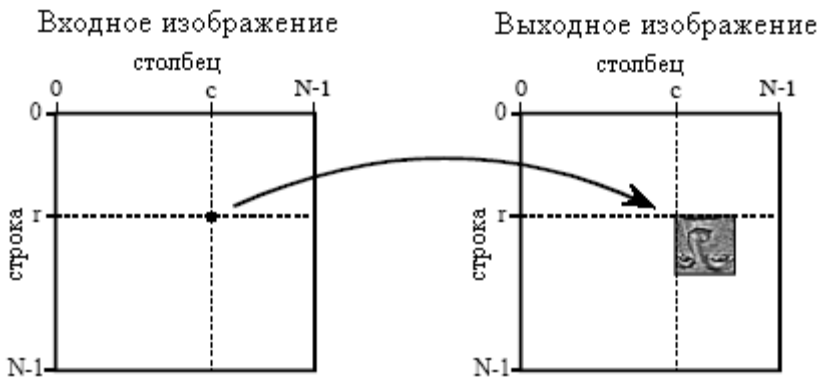


Рис. 24.14 Свертка изображения рассматриваемая со стороны входа

Взгляд на свертку изображения с точки зрения выходного сигнала иллюстрируется на рис. 24.15. Каждый пиксель в выходном изображении получает вклад от множества пикселей во входном изображении так, как это показано на рисунке для отсчета [*r*,*c*]. ФРТ поворачивается на 180^0 вокруг пикселя [*0*,*0*], а затем сдвигается таким образом, чтобы пиксель [*0*,*0*] в ФРТ совместился с пикселем [*r*,*c*] во входном изображении. Если ФРТ использует только положительные индексы, она будет располагаться выше и левее пикселя [*r*,*c*] во входном изображении. Значение пикселя при [*r*,*c*] в выходном изображении находится умножением пикселей перевернутой ФРТ на соответствующие пиксели во входном изображении и суммированием результатов произведений. Эта процедура показана уравнением (24.3) и программой в таблице 24.1.

$$y[r,c] = \sum_{k=0}^{M-1} \sum_{j=0}^{M-1} h[k,j]x[r-k,c-j] \qquad (24.3)$$

Обратите внимание, что поворот ФРТ, являющийся результатом операции свертки, компенсировал поворот, сделанный при формировании ФРТ. Последнее позволило расположить лицо на рис. 24.15 нормально, сохраняя такую же ориентацию, как и у обнаруживаемого образца во входном изображении. Таким образом, мы успешно использовали *свертку* для выполнения *корреляции*. Остается по примеру рис. 24.13 убедиться, что яркое пятнышко в изображении корреляции, которое должно появиться на рис. 24.15, показывает, что цель была достигнута.

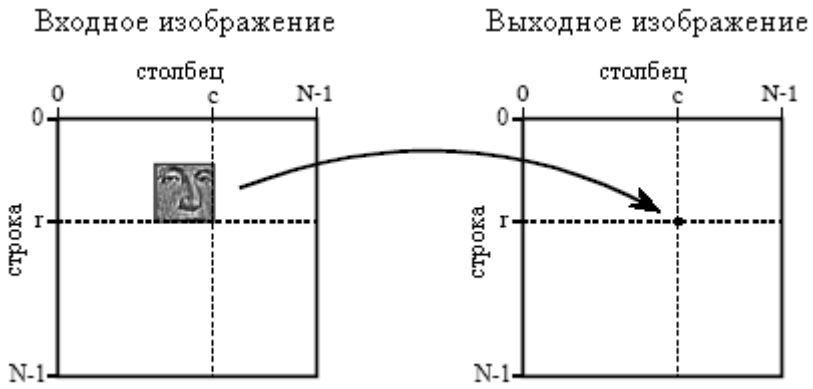


Рис. 24.15 Свертка изображения рассматриваемая со стороны выхода

БПФ свертка дает такой же результат, что и программа традиционной свертки в таблице 24.1. Действительно ли сокращенное время исполнения, обеспечиваемое БПФ сверткой, стоит дополнительного усложнения программы? Давайте разберемся по подробнее. На рис. 24.6 приведено сравнение времени исполнения между традиционной сверткой, использующей *плавающую запятую* (отмечено как ПЗ), традиционной сверткой, использующей *целые* (отмечено как ЦЕЛ) и БПФ сверткой, использующей плавающую запятую (отмечено как БПФ). Данные приведены для двух различных изображений с размерами 512×512 и 128×128.

Во-первых, заметим, что время исполнения, требуемое для БПФ свертки, не зависит от размера ядра, что дает пологую линию на этом графике. На персональном компьютере Pentium с тактовой частотой 100 МГц БПФ свертка изображения размером 128×128 пикселей может быть выполнена примерно за 15 секунд, тогда как изображение размером 512×512 потребует более чем четырех минут. Увеличение числа вычислений показывает, что для БПФ свертки время исполнения для изображения размером $N \times N$ пропорционально $N^2 log_2(N)$. То есть, изображение размером 512×512 требует времени примерно в 20 раз больше, чем изображение размером 128×128.

Время исполнения традиционной свертки для свертки изображения размером $N \times N$ с ядром размером $M \times M$ пропорционально N^2M^2. Это можно понять, исследуя программу в табл. 24.1. Другими словами, время исполнения для традиционной свертки очень *сильно зависит* от размера используемого ядра. Как показано на рисунке, БПФ свертка быстрее традиционной свертки, использующей плавающую запятую, если ядро больше чем приблизительно 10×10 пикселей. В большинстве случаев для традиционной свертки могут быть использованы целые, увеличивающие точку пересечения графиков до, примерно, 30×30 пикселей. Как показано на рисунке, точки пересечения графиков слегка зависят от размера свертываемого изображения. Концептуально следует запомнить, что БПФ свертка полезна только для *больших* ядер фильтров.

Таблица 24.1.

```
100 CONVENTIONAL IMAGE CONVOLUTION
110 '
120 DIM X[99,99]                'holds the input image, 100×100 pixels
130 DIM H[28,28]                'holds the filter kernel, 29×29 pixels
140 DIM Y[127,127]              'holds the output image, 128×128 pixels
150 '
160 FOR R% = 0 TO 127           'loop through each row and column in the output
170 FOR C% = 0 TO 127           'image calculating the pixel value via Eq. 24.3
180 '
190 Y[R%,C%] = 0                'zero the pixel so it can be used as an accumulator
200 '
210 FOR J% = 0 TO 28            'multiply each pixel in the kernel by the corresponding
220 FOR K% = 0 TO 28            'pixel in the input image, and add to the accumulator
230 Y[R%,C%] = Y[R%,C%] + H[J%,K%] * X[R%-J%,C%-J%]
240 NEXT K%
250 NEXT J%
260 '
270 NEXT C%
280 NEXT R%
290 '
300 END
```

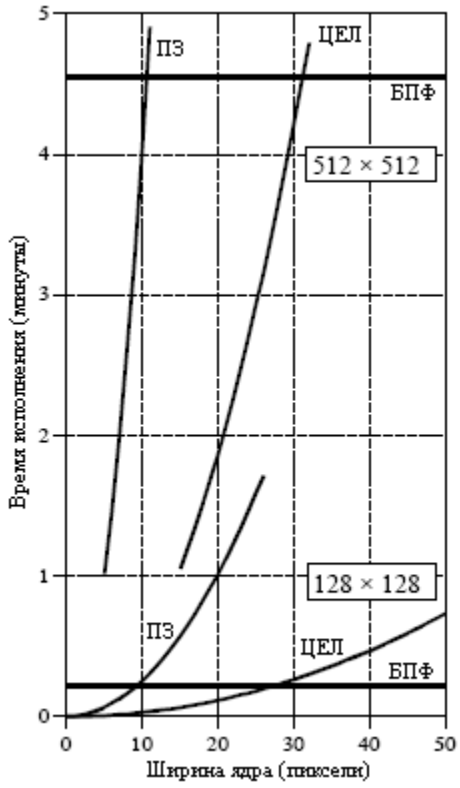


Рис. 24.16 Время исполнения для свертки изображения

ГЛАВА
25

СПЕЦИАЛЬНЫЕ МЕТОДЫ ОБРАБОТКИ ИЗОБРАЖЕНИЯ

Эта глава представляет четыре специфических аспекта обработки изображения. Во-первых, обсуждаются способы, характеризующие *пространственное разрешение*. Оно описывает минимальный размер, который должен иметь объект, чтобы его можно было заметить в изображении. Во-вторых, исследуется *отношение сигнал-шум*, объясняется, каким образом слабо светящийся объект, может быть и будет обнаружен. В-третьих, знакомятся с *морфологическими* методами. Это нелинейные операции, используемые для манипулирования двоичными изображениями (в которых каждый пиксель является либо черным, либо белым). В-четвертых, описывается замечательная техника *компьютерной томографии*. Она произвела революцию в медицинской диагностике, дав детальные изображения внутренних органов тела человека.

Пространственное разрешение

Предположим, что мы хотим сравнить две системы формирования изображения с целью определения, какая из них имеет лучшую пространственную разрешающую способность. Другими словами, мы хотим узнать, какая система может обнаружить наиболее мелкий объект. Чтобы упростить сравнение, мы бы желали, чтобы ответ для каждой из систем представлял собой *некоторое число*. Это позволяет сделать непосредственное сравнение при принятии решения, какую из систем взять за основу разрабатываемого проекта. К сожалению, одного параметра не всегда бывает достаточно для того, чтобы охарактеризовать все тонкие аспекты получения изображения. Это осложняется еще и тем фактом, что пространственное разрешение ограничивается двумя разными, но взаимосвязанными эффектами: *интервалом между отсчетами* и *размером апертуры дискретизации*. Данный раздел охватывает две главные темы: (1) как наилучшим образом можно использовать один параметр для того, чтобы охарактеризовать пространственное разрешение, и (2) взаимосвязь интервала между отсчетами с размером апертуры дискретизации.

На рис. 25.1 показаны профили трех обладающих круговой симметрией ФРТ: пилюлеобразной, Гауссиана и экспоненциальной. Они представляют обычно встречающиеся в системах получения изображений ФРТ. Как описывалось в предыдущей главе, пилюлеобразная ФРТ является следствием неправильно сфокусированного объектива. Аналогичным образом, Гауссиан является результатом объединения случайных ошибок, подобных ошибкам при наблюдении звезд сквозь турбулентности в атмосфере. Экспоненциальная ФРТ появляется тогда, когда энергия электронов или рентгеновских лучей, падающих на люминесцентный слой, преобразуется в свет. Это явление используется в детекторах излучения, усилителях света приборов ночного видения и в дисплеях на электроннолучевых трубках (ЭЛТ) (это явление используется также и в плазменных экранах, где, как и в лампах дневного света, на слой люминофора падает ультрафиолетовое излучение – прим. перев.). Точная форма этих трех ФРТ для данной дискуссии не важна, единственно, что следует заметить, что они широко представляют ФРТ наблюдаемые в приложениях реального мира.

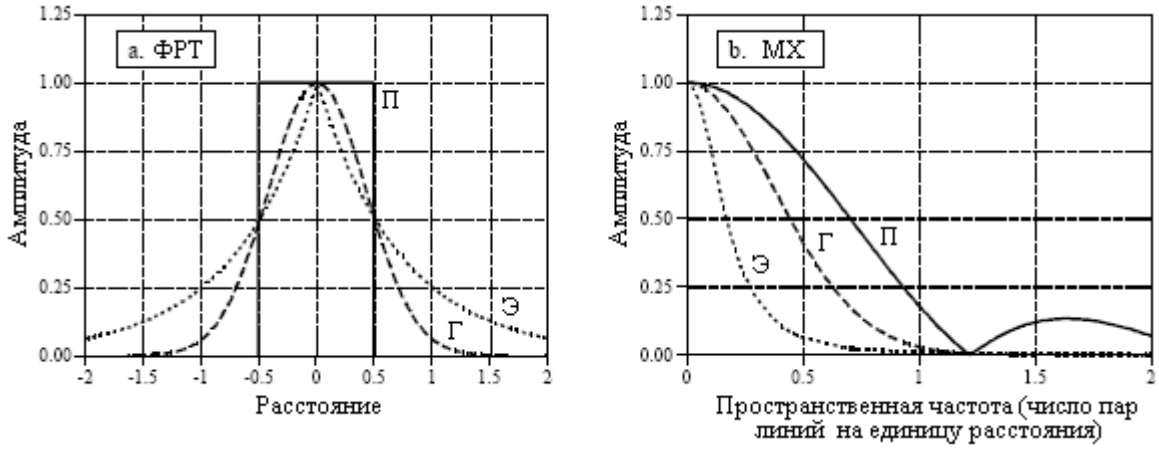


Рис. 25.1 ПШПМ против ФРТ

ФРТ содержит полную информацию о пространственном разрешении. Чтобы выразить пространственное разрешение одним числом, мы можем игнорировать *форму* ФРТ и просто измерить ее *ширину*. Наиболее обычным способом определения ширины является нахождение значения полной ширины ФРТ при половине ее максимума (ПШПМ). Например, все ФРТ на рис. 25.1a обладают ПШПМ равной 1 единице (*0,5*-(*-0,5*) = *1* – прим. перев.).

К сожалению, этот метод имеет два существенных недостатка. Во-первых, он не соответствует другим мерам пространственного разрешения, включая субъективные суждения рассматривающих изображения наблюдателей. Во-вторых, обычно очень трудно непосредственно измерить ФРТ. Только представьте, ввести дельта импульс в систему изображения; то есть, сфотографировать изображение очень маленькой белой точки на черном фоне. По определению, полученное изображение и будет ФРТ системы. Проблема заключается в том, что измеренная ФРТ будет содержать всего несколько пикселей, и ее контрастность будет низкой. Особенно если Вы еще и не очень аккуратны, то случайный шум поглотит измерение. Например, представьте, что изображением дельта импульса является массив размером 512×512, состоящий из всех нулей за исключением одного единственного пикселя, имеющего значение 255. Теперь сравните это с нормальным изображением, в котором все 512×512 пикселей имеют среднее значение около 128. Оперируя приблизительными величинами, сигнал изображения дельта импульса примерно в 100000 раз слабее, чем нормальное изображение. Неудивительно, что отношение сигнал–шум будет плохим; едва ли здесь будет наблюдаться какой-нибудь сигнал!

Основной мыслью, пронизывающей эту книгу является то, что, прежде всего, необходимо понять, в какой области закодирована информация в сигнале. Например, звуковые сигналы нужно обрабатывать в частотной области, тогда как сигналы изображения следует обрабатывать в пространственной области. И, несмотря на это, одним из способов измерения разрешающей способности изображения является исследование *частотной характеристики*. Это идет в разрез с основной философией этой книги; однако, это общий метод и Вам следует с ним познакомиться.

Взятие двумерного преобразования Фурье от ФРТ дает двумерную частотную характеристику. Если ФРТ обладает круговой симметрией, ее частотная характеристика тоже будет обладать круговой симметрией. В этом случае полная информация о частотной характеристике содержится в ее профиле. То есть, после вычисления частотной области посредством метода БПФ, все, что нужно, так это столбцы с *0* по *N/2* в строке *0*. На жаргоне обработки изображений, такое отображение частотной характеристики называется **модуляционной характеристикой** (**МХ**). На рис. 25.1b показаны МХ для трех

ФРТ, приведенных на рис. 25.1a. В тех случаях, когда ФРТ не обладает круговой симметрией, информация содержится во всей двумерной частотной характеристике. Однако обычно достаточно знать кривые МХ в вертикальном и горизонтальном направлениях (т.е. столбцы с *0* по *N/2* в строке *0*, и строки с *0* по *N/2* в столбце *0*). Возьмите на заметку: процедура извлечения строки или столбца из двумерного частотного спектра *не* является эквивалентной взятию одномерной БПФ от профилей, показанных на рис. 25.1a. Вскоре мы вернемся к этой проблеме. Как показано на рис. 25.1, значения ПШПМ не соответствуют таковым для кривым МХ.

На рис. 25.2 показан **калибратор пар линий**, инструмент, используемый для измерения разрешающей способности изображения посредством МХ. Калибраторы пар линий поставляются в различных формах в зависимости от конкретных приложений. Например, черно-белый узор, показанный на этом рисунке, может непосредственно использоваться для тестирования видеокамер. Для систем рентгеновских изображений, черные линии могут быть сделаны из свинца с прозрачным для рентгеновских лучей материалом между ними. Ключевой характеристикой здесь является то, что черные и белые линии сближаются к одному из концов. При снятии изображения калибратора пар линий, линии со стороны их сближения будут сливаться вместе, хотя с другого конца они будут отчетливо видны, как отдельные линии. Где-то в средине линии будут едва-едва отделимы друг от друга. Наблюдатель, смотрящий на изображение, отмечает это место и считывает соответствующее разрешение по отградуированной в соответствующем масштабе шкале.

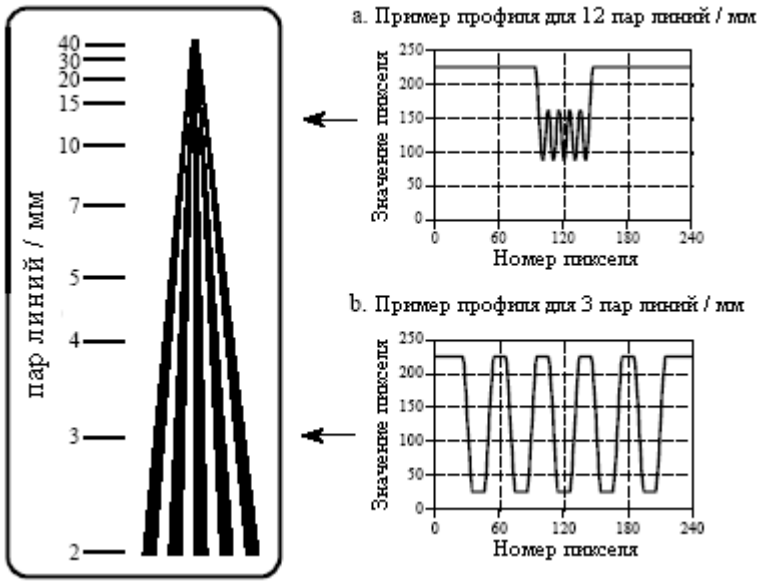


Рис. 25.2 Калибратор пар линий

То, каким образом черные линии сливаются вместе, является очень важным для понимания ограничений, связанных с такими измерениями. Представьте полученное изображение калибратора пар линий с рис. 25.2. На рис. 25.2a и рис. 25.2b показаны примеры профилей для низкой и высокой пространственных частот. Для низкой частоты, показанной на рис. 25.2b, кривая имеет плоские вершину и основание, и растянутые фронты. Для высокой пространственной частоты, рис. 25.2a, уменьшилась *амплитуда* модуляции. Это точно то, что описывает кривая МХ, показанная на рис. 25.1b: более высокие частоты уменьшаются по амплитуде. Отдельные черные линии будут различимы в изображении до тех пор, пока их амплитуды больше примерно 3% - 10% их первоначальной высоты. Это связано со способностью глаза различать низко контрастную разницу между пиками и впадинами в присутствии шума в изображении.

Сильным преимуществом измерений при помощи калибратора пар линий является то, что они являются простыми и быстрыми. Сильным недостатком является то, что они полагаются на глаз человека, и поэтому в них есть некоторая субъективная составляющая. Если же измерена полная кривая MX, то наиболее общий способ выразить разрешающую способность системы состоит в том, чтобы указать частоту, при которой MX снижается либо до 3%, либо до 5%, либо до 10%. К сожалению, Вам не всегда скажут, какое из этих значений амплитуд используется в данный момент; технические описания часто пользуются неопределенными терминами, наподобие "предел разрешающей способности". Поскольку производители любят, чтобы их спецификации были как можно лучше (вне зависимости от того, на что действительно способен их прибор) будьте осторожны и интерпретируйте эти амбициозные термины как: до 3% по кривой MX.

Следует отметить тонкий момент, что MX определяется в терминах *синусоидальных* волн, тогда как калибратор пар линий использует *прямоугольные* волны. То есть, черные линии представляют собой однородные темные области, разделенные однородными светлыми областями. Это сделано с точки зрения удобства производства; очень трудно выполнить линии синусоидально изменяющейся темноты. Каковы последствия использования прямоугольных волн для измерения MX? Все частотные составляющие в высокочастотной пространственной области, кроме основной частоты прямоугольной волны, удаляются. Это приводит к тому, что появляющаяся модуляция выглядит синусоидальной, такой как показано на рис. 25.2a. На низких частотах, таких как показано на рис. 25.2b, волны выглядят прямоугольными. Содержащаяся в прямоугольной волне синусоидальная волна основной частоты имеет амплитуду равную $4/\pi = 1{,}27$ от амплитуды прямоугольного импульса (см. рис. 13.10). Как результат: калибратор пар линий дает небольшую переоценку истинной разрешающей способности системы, начиная с действующей амплитуды от более чем для чисто черного до чисто белого. Это интересно, но почти всегда не учитывается.

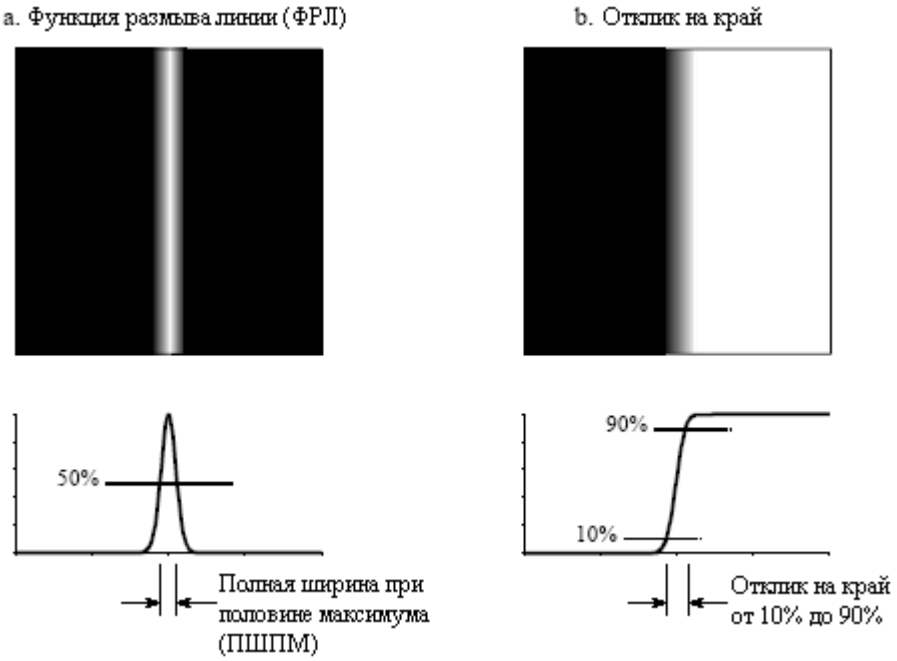


Рис. 25.3 Функция размыва линии и отклик на край

То, что для измерения MX прямоугольные волны и синусоидальные волны используются попеременно, привело к возникновению специальной терминологии. Вместо слова "период" в обработке изображений используется термин **пара линий** (темная линия, за которой следует светлая линия). Например, для пространственной

частоты вместо *25 периодов на миллиметр* будет употребляться *25 пар линий на миллиметр*.

Ширина ФРТ трудно измерима и плохо отслеживается человеческим восприятием. Методы МХ находятся в трудной области для понимания того, какое воздействие оказывает на закодированную информацию разрешающая способность. Существует ли более благоприятная альтернатива? Ответом будет - да, **функция размыва линии (ФРЛ)** и **отклик на край**. Как показано на рис. 25.3, функция размыва линии представляет собой отклик системы на тонкую линию поперек изображения. Точно так же откликом на край является реакция системы на резкий немедленный переход (край). Поскольку линия является производной (или первой разностью) края, ФРЛ является производной (или первой разностью) отклика на край. Единственным измеряемым параметром, используемым здесь, является расстояние, требуемое для того, чтобы отклик на край вырос от уровня в 10% до уровня в 90%.

Существует большое число преимуществ использования отклика на край для измерения разрешающей способности. Во-первых, измерение проводится в той же форме, в какой закодирована информация в изображении. Действительно, единственной причиной желания узнать разрешающую способность системы является желание понять, насколько *смазываются* края в изображении. Вторым преимуществом является то, что отклик на край просто измерить, поскольку край легко создается в изображениях. Если это необходимо, то можно легко найти ФРЛ, взяв первую разность от отклика на край.

Третьим преимуществом является то, что все типичные отклики на край обладают подобной формой даже при том, что они могут происходить от решительно различных ФРТ. Это показано на рис. 25.4a, где представлены отклики на край пилюлеобразной ФРТ, Гауссиана и экспоненциальной ФРТ. Так как кривые подобны, то расстояние в переделах уровня от 10% до 90% является отличным единственным измеряемым параметром разрешения. Четвертым преимуществом является то, что МХ может быть непосредственно найдена взятием одномерного БПФ от ФРЛ (в отличие от вычисления МХ из ФРТ, где должно использоваться двумерное преобразование Фурье). На рис. 25.4b показаны МХ соответствующие откликам на край, приведенным на рис. 25.4a. Другими словами, кривые на рис. 25.4a преобразуются в кривые на рис. 25.4b взятием первой разности (чтобы найти ФРЛ), а затем взятием БПФ.

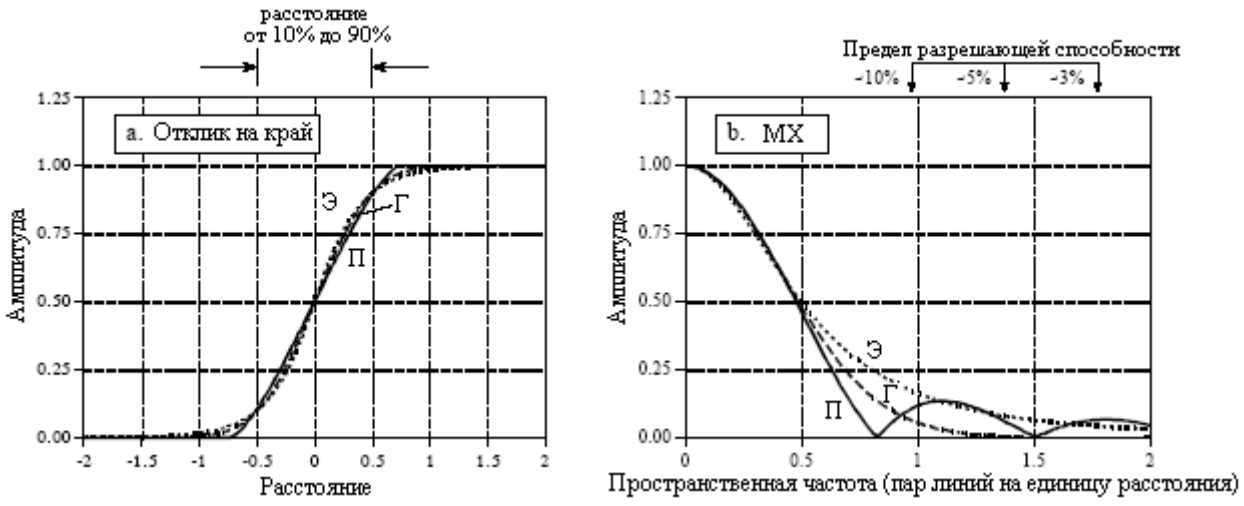


Рис. 25.4 Отклик на край и МХ

Пятым преимуществом, как показано на рис. 25.4a и рис. 25.4b, является то, что подобные отклики на край имеют подобные кривые МХ. Это позволяет нам легко преобразовывать одно измерение в другое. В частности система, имеющая при отклике на край в пределах уровня от 10% до 90% расстояние x, имеет предел разрешающей

способности (контрастность 10%) около 1 пары линий на расстояние равное *x*. Единицы "расстояния" будут зависеть от типа системы, с которой имеют дело. Например, рассмотрим три различных системы обработки изображения имеющих при отклике на край в пределах уровня от 10% до 90% расстояния: 0,05 мм, 0,2 миллирадиана и 3,3 пикселя. 10% уровень контрастности на соответствующей кривой MX будет примерно: 20 пар линий на миллиметр, 5 пар линий на миллирадиан и 0,33 пары линий на пиксель, соответственно.

На рис. 25. 5 иллюстрируется математическое соотношение между ФРТ и ФРЛ. На рис. 25.5a изображена пилюлеобразная ФРТ с показанной белым круглой областью со значением 1, охваченной областью с нулевыми значениями повсюду, которая показана серым. Профиль ФРТ (т.е., значения пикселей вдоль линии, проведенной через центр изображения) будет прямоугольным импульсом. На рис. 25.5b показаны соответствующие ФРЛ. Как показано, ФРЛ математически равна *интегрированному профилю* ФРТ. Он находится путем прохода через изображение в некотором направлении, как показано лучами (стрелками). Каждое значение в интегрированном профиле является *суммой* значений пикселей вдоль соответствующего луча.

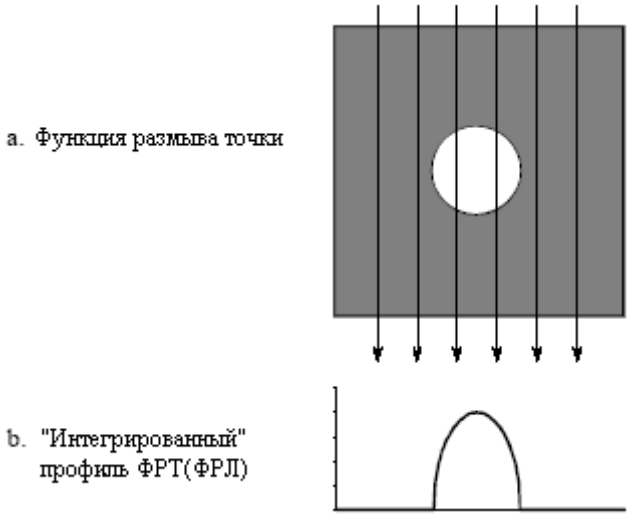


Рис. 25.5 Связь между ФРТ и ФРЛ

В этом примере, где лучи вертикальны, каждая точка в интегрированном профиле находится суммированием значений всех пикселей в каждом столбце. Это соответствует ФРЛ *вертикальной* линии в изображении. ФРЛ *горизонтальной* линии в изображении находится суммированием значений всех пикселей в каждой строке. Для непрерывных изображений эта концепция остается такой же, но суммирование заменяется интегралом.

Как показано в этом примере ФРЛ может быть непосредственно вычислена из ФРТ. Однако ФРТ не всегда может быть вычислена из ФРЛ. Это связано с тем, что ФРТ содержит информацию о пространственном разрешении по *всем направлениям*, тогда как ФРЛ ограничена всего одним конкретным направлением. Система обладает только одной ФРТ, но бесконечным числом ФРЛ, по одному для каждого угла. Например, представьте систему имеющую продолговатую ФРТ. Это делает пространственное разрешение разным в вертикальном и горизонтальном направлениях, что выражается в отличных друг от друга ФРЛ по этим направлениям. Измерение ФРЛ для одного угла не обеспечивает достаточной информацией для вычисления полной ФРТ, за исключением специальных случаев, когда ФРТ обладает круговой симметрией. Множественные измерения ФРЛ для разных углов делают возможным вычисление ФРТ не обладающих круговой симметрией; однако тогда очень сильно привлекается математика, и, обычно, это не стоит затраченных усилий. Фактически, проблема вычисления ФРТ по множеству измерений ФРЛ точно

такая же проблема, с которой сталкиваются в *компьютерной томографии*, обсуждаемой в этой главе позже.

С практической точки зрения для большинства систем обработки изображений между ФРЛ и ФРТ значительной разницы нет, и очень типично видеть, как одну из них используют в виде аппроксимации другой. Это даже более чем оправдано, учитывая, что существуют два общих случая, где они идентичны: прямоугольная ФРТ обладает прямоугольной ФРЛ (с точно такой же шириной) и Гауссова ФРТ имеет Гауссову ФРЛ (с точно таким же стандартным отклонением).

Данные концепции могут быть обобщены в два приема: как *оценить* представленную Вам характеристику разрешения и как *измерить* Вашу собственную характеристику разрешения. Предположим, что Вы натолкнулись на рекламу, утверждающую: "Эта система обладает разрешающей способностью 40 пар линий на миллиметр". Вам следует интерпретировать это следующим образом: "Синусоида 40 пар линий на миллиметр будет иметь амплитуду, уменьшенную до 3%-10% от своего истинного значения и будет только едва заметна в изображении". Вам также следует посчитать в уме, что 40 пар линий на миллиметр при 10% контрастности, эквивалентной росту уровня отклика на край в пределах от 10% до 90%, это - *1/(40* пар линий на миллиметр) = 0,025 миллиметра. Если спецификация МХ приводится для уровня контрастности в 3%, то отклик на край будет приблизительно от 1,5 до 2 раз шире.

Когда Вы измеряете пространственную разрешающую способность системы обработки изображения, действия выполняются в обратном порядке. Поместите на изображение резкий край и измерьте получившийся в результате отклик на край. Расстояние в пределах уровня от 10% до 90% этой кривой представляет собой лучшее отдельное измерение параметра разрешающей способности системы. Чтобы осчастливить Вашего босса и продавцов, для нахождения ФРЛ возьмите первую разность от отклика на край, а затем для нахождения МХ используйте БПФ.

Интервал между отсчетами и апертура дискретизации

На рис. 25.6 показаны два экстремальных примера дискретизации, которые мы будем называть **идеальный датчик** и **искажающий датчик**. Представьте, что на рис. 25.6а показана поверхность датчика изображения наподобие прибора с зарядовой связью. Свет, попадающий куда-нибудь внутрь на один из квадратных пикселей, вносит вклад в значение *только* этого пикселя и никакого другого. Это показано на рисунке черной апертурой дискретизации, точно заполняющей один из квадратных пикселей. Это оптимальная ситуация для датчика изображений, потому что обнаружен *весь* свет, и между смежными пикселями *отсутствуют* перекрытия или перекрестная связь. Другими словами, апертура дискретизации точно равна интервалу между отсчетами.

На рис. 25.6е воспроизведен альтернативный пример. Апертура дискретизации значительно больше, чем интервал между отсчетами, и она соответствует Гауссову распределению. Другими словами, каждый пиксель датчика в области, *окружающей* пиксель, получает вклад от падающего на датчик света. Это должно звучать знакомо, потому что это – взгляд на свертку со стороны выхода. С соответствующей точки зрения на свертку со стороны входа, узкий луч света, попадающий на датчик, внесет вклад в значение нескольких соседних пикселей, тоже согласно Гауссову распределению.

А сейчас обратите свое внимание на отклик на край для этих двух примеров. Маркеры на каждом графике показывают действительное значение пикселя, которое Вы бы нашли в изображении, тогда как соединительные линии показывают лежащую в *основе кривую*, которая подвергается дискретизации. Важной концепцией является то, что форма этой лежащей в основе кривой определяется *только апертурой дискретизации*. Это означает, что разрешающая способность конечного изображения может ограничиваться

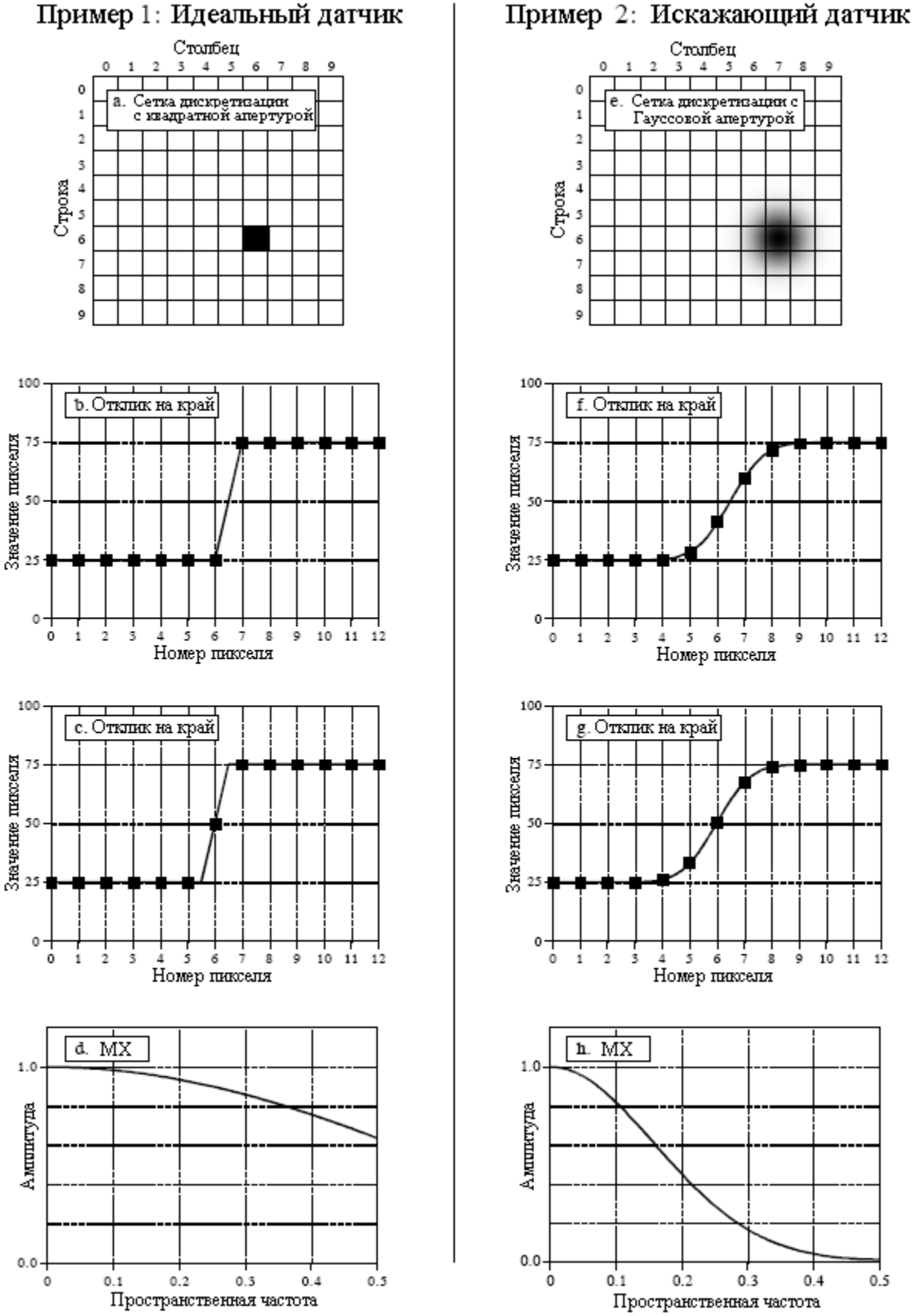


Рис. 25.6 Идеальный и искажающий датчики

двумя способами. Во-первых, лежащая в основе кривая может иметь плохое разрешение, как следствие слишком большой апертуры дискретизации. Во-вторых, может быть слишком большим интервал между отсчетами, в результате чего между отсчетами

теряются маленькие детали. Представленные для каждого примера две кривые отклика на край иллюстрируют, что действительные отсчеты могут попадать в любое место на лежащей в основе кривой. Другими словами, отображаемый край может сидеть точно на пикселе или захватывать два пикселя. Заметим, что идеальный датчик имеет *ноль* отсчетов или *один* отсчет на нарастающем участке отклика на край. Аналогично, искажающий датчик имеет от *трех* до *четырех* отсчетов на нарастающем участке отклика на край.

Что ограничивает разрешение этих двух систем? Ответ дается *теоремой о дискретизации*. Как обсуждалось в Главе 3, дискретизация захватывает составляющие всех частот, лежащих ниже половины частоты дискретизации, тогда как более высокие частоты из-за совмещения имен теряются. Теперь посмотрим на кривую MX на рис. 25.6h. Апертура дискретизации искажающего датчика удаляет все частоты, лежащие выше половины частоты дискретизации, следовательно, во время дискретизации *ничего* не теряется. Это означает, что разрешающая способность данной системы целиком ограничивается апертурой дискретизации, а не интервалом между отсчетами. Говоря по-другому, апертура дискретизации действует как фильтр антисовмещения, что позволяет осуществлять дискретизацию без потерь.

Для сравнения, кривая MX на рис. 25.6d показывает, что разрешающую способность этой системы ограничивают *оба* процесса. Высокочастотный завал кривой MX представляет информационные потери из-за *апертуры дискретизации*. Поскольку кривая MX до частоты 0,5 не падает до нуля, здесь также присутствуют потери информации во время дискретизации, результат конечного *интервала между отсчетами*. Что ограничивает разрешающую способность больше? На этот вопрос трудно дать численный ответ, поскольку эти процессы по-разному приводят к ухудшению изображения. Достаточно сказать, что разрешающая способность идеального датчика (пример 1) ограничивается интервалом между отсчетами в большей степени.

Хотя эти концепции могут показаться трудными, они сводятся к очень простому правилу для практического применения. Рассмотрим систему с некоторым расстоянием, например 1 мм, при росте отклика на край от 10% до 90%. Если интервал между отсчетами больше, чем 1 мм (менее одного отсчета на нарастающем участке отклика на край) система будет ограничиваться *интервалом между отсчетами*. Если интервал между отсчетами менее чем, 0,33 мм (более трех отсчетов на нарастающем участке отклика на край) разрешающая способность будет ограничиваться *апертурой дискретизации*. Когда система имеет 1-3 отсчета на нарастающем участке отклика на край, она будет ограничиваться обоими факторами.

Отношение сигнал–шум

Объект видим на изображении потому, что он обладает отличной от его окружения яркостью. Другими словами, контрастность объекта (т.е. сигнал) должна превышать шум изображения. Последнее можно разделить на два класса: ограничения, связанные с *глазом* и ограничения, связанные с *данными*.

На рис. 25.7 иллюстрируется эксперимент по измерению способности глаза обнаруживать слабые сигналы. В зависимости от условий наблюдения глаз человека может обнаруживать минимальную контрастность от 0,5% до 5%. Другими словами, люди могут различать приблизительно от 20 до 200 оттенков серого между наичернейшим черным и наибелейшим белым. Точное число зависит от различных факторов, таких как яркость окружающего освещения, расстояния между двумя сравниваемыми областями и того, каким образом формируется изображение шкалы серого (видеомонитор, фотография, полутона и т.д.).

Для усиления контрастности пикселей выбранного диапазона значений может быть использовано преобразование шкалы серого из Главы 23, дающее ценный инструмент по преодолению ограничений глаза человека. Контрастность одного уровня яркости

увеличивается ценой снижения контрастности другого уровня яркости. Однако это работает только тогда, когда контрастность объекта не теряется в случайном шуме изображения. Это более серьезная ситуация; вне зависимости от работы глаза, *сигнал* не содержит информации достаточной для обнаружения объекта.

Рис. 25.7 Определение контрастности

На рис. 25.8 показано изображение с тремя квадратами, имеющими контрастность 5%, 10% и 20%. Фон содержит нормально распределенный случайный шум со стандартным отклонением около 10% контрастности. Отношение сигнал–шум определяется как контрастность, деленная на стандартное отклонение шума, что дет в результате для трех квадратов отношения сигнал–шум, равные *0,5*, *1,0* и *2,0*. Вообще неприятности начинаются тогда, когда отношение сигнал–шум падает примерно ниже *1,0*.

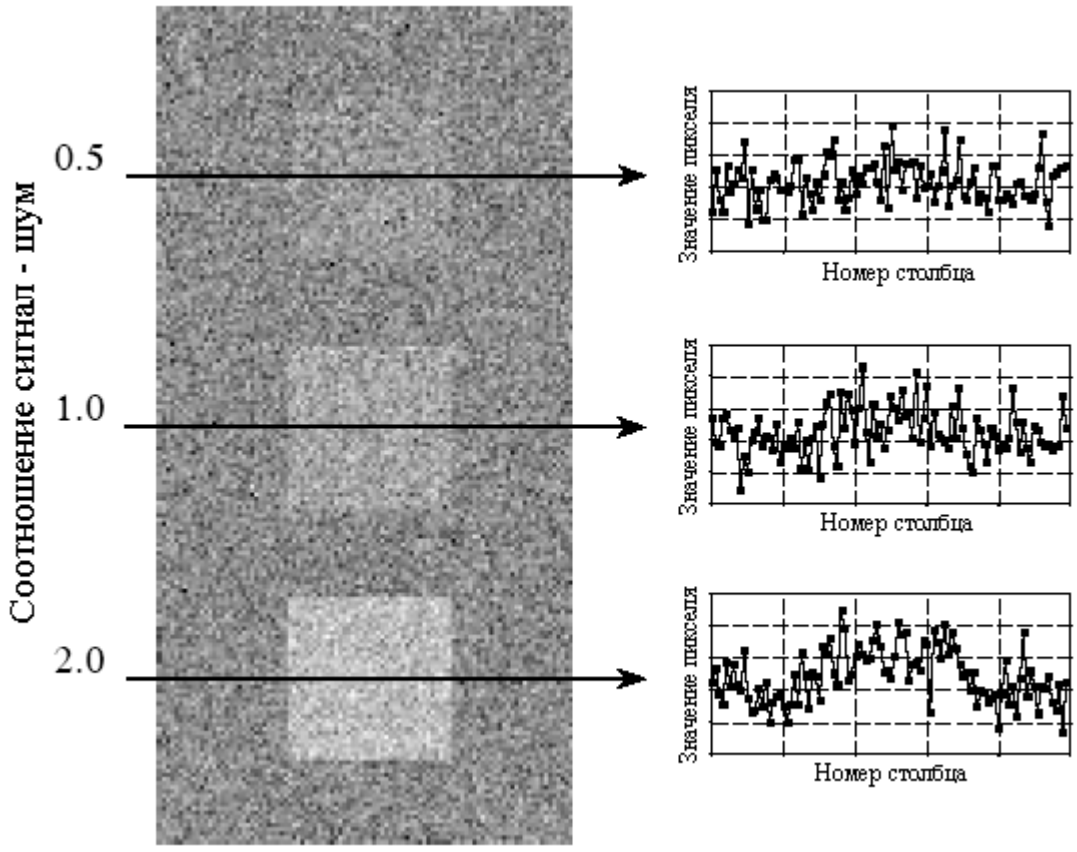


Рис. 25.8 Минимально обнаруживаемое отношение сигнал–шум

Точное значение минимально обнаруживаемого отношения сигнал–шум зависит от *размера* объекта; чем больше объект, тем легче его обнаружить. Для того чтобы это

понять, представьте процесс сглаживания изображения на рис. 25.8 при помощи квадратного ядра фильтра размером 3×3. Контрастность остается такой же, но шум снижается раза в три (т.е. в корень квадратный от числа пикселей в ядре). Поскольку отношение сигнал–шум утроилось, слабо контрастные объекты можно увидеть. Чтобы увидеть едва заметные объекты, ядро фильтра должно быть сделано еще больше. Например, ядро размером 5×5 улучшает отношение сигнал–шум в $\sqrt{25} = 5$ раз. Такая стратегия может быть продолжена до тех пор, пока ядро фильтра не станет равным размеру обнаруживаемого объекта. Это означает, что способность обнаруживать объекты пропорциональна *корню квадратному* от его *площади*. Если диаметр объекта удваивается, то он может быть обнаружен в шуме вдвое большей величины.

Почти таким же способом происходит визуальная обработка в мозгу, сглаживая наблюдаемое изображение с ядрами фильтров различного размера в попытке распознать слабоконтрастные объекты. Три профиля на рис. 25.8 как раз демонстрируют, как хорошо удается людям обнаруживать объекты среди помех. Даже если профили объектов едва различимы, они все равно видны на изображении. Для того чтобы действительно оценить способности зрительной системы человека попробуйте написать алгоритмы, работающие в среде с таким низким отношением сигнал–шум. Вы будете унижены тем, что могут делать Ваши мозги, а Ваши программы - нет!

Случайный шум изображения возникает в двух обычных формах. Первый вид, показанный на рис. 25.9a, имеет постоянную амплитуду. Другими словами, темные и светлые области в изображении одинаково зашумлены. Для сравнения, рис. 25.9b иллюстрирует шум, *увеличивающийся с увеличением уровня сигнала*, что приводит к тому, что яркие области в изображении зашумлены больше чем черные. В большинстве изображений присутствуют оба источника шума, но обычно один или другой доминирует. Например, типичным для шума является то, что при уменьшении уровня сигнала уменьшается и шум до тех пор, пока не будет достигнуто плато шума постоянной амплитуды.

Типичным источником шума постоянной амплитуды является *предварительный видео усилитель*. Все схемы аналоговой электроники производят шумы. Однако, наибольший вред наносится там, где усиливаемый сигнал имеет наиболее маленькую величину, как в ПЗС или других сенсорах изображения. Шум предварительного усилителя возникает в результате хаотичного движения электронов в транзисторах. Это делает уровень шума зависимым от того, каким образом сконструирована электроника, а не от уровня усиливаемого сигнала. Например, типичная камера с ПЗС будет иметь отношение сигнал–шум приблизительно от 300 до 1000 (от 40 dB до 60 dB), определяемое как: уровень сигнала полной шкалы, деленный на стандартное отклонение шума постоянной амплитуды.

Шум, увеличивающийся с ростом уровня сигнала (дробовой шум – прим. перев.), возникает тогда, когда изображение представлено небольшим количеством отдельных частиц. Например, это может быть *рентгеновское излучение*, проходящее через пациента, попадающие в камеру *фотоны света* или *электроны* в ямах в ПЗС. Математика, управляющая этими вариациями, называется **статистикой отсчетов** или **распределением Пуассона**. Предположим, что экран ПЗС равномерно освещен так, что в каждой яме в среднем генерируется 10000 электронов. Совершенно случайным образом в некоторых ямах будет находиться большее число электронов, а в некоторых меньшее. Чтобы быть более точными число электронов будет иметь нормальное распределение со средним значением равным 10000 и некоторым стандартным отклонением, описывающим величину изменений от ямы к яме. Ключевой характеристикой распределения Пуассона является то, что стандартное отклонение здесь равно корню квадратному от числа отдельных частиц. То есть, если в каждом пикселе есть N частиц, среднее значение равно

N, а стандартное отклонение равно $\sqrt{N}$. Это делает отношение сигнал–шум равным $\frac{N}{\sqrt{N}}$ или просто $\sqrt{N}$. В форме уравнений это выглядит как:

$$\mu = N;$$

$$\sigma = \sqrt{N}; \qquad (25.1)$$

$$\textit{Отношение сигнал-шум} = \sqrt{N}.$$

В примере с ПЗС стандартное отклонение равно $\sqrt{10000} = 100$. Аналогично, отношение сигнал–шум также равно $\sqrt{10000} = 100$. Если среднее число электронов на яму увеличивается до одного миллиона, оба и стандартное отклонение и отношение сигнал–шум увеличиваются до *1000*. То есть шум становится больше, по мере того как сигнал становится больше, как и показано на рис. 25.9b. Однако сигнал нарастает *быстрее*, чем шум, приводя к общему улучшению отношения сигнал–шум. Не заблуждайтесь, думая, что более слабый сигнал дает меньший шум и следовательно, больше информации. Помните, Ваша цель *не* снизить шум, а извлечь *из* шума сигнал. Это делает отношение сигнал–шум ключевым параметром.

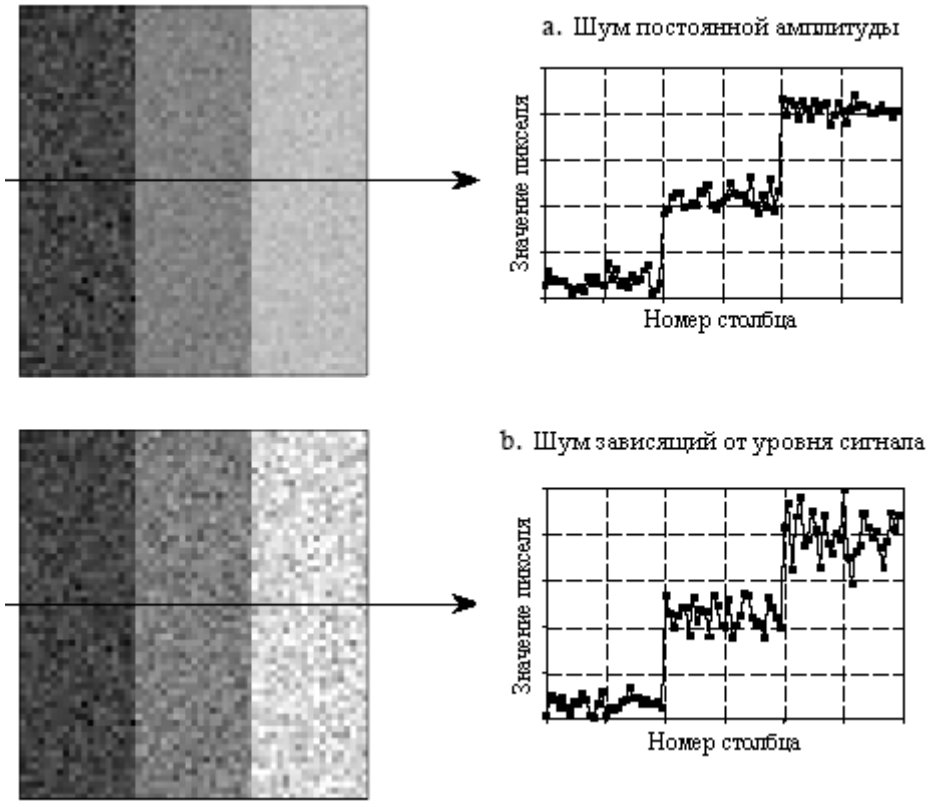


Рис. 25.9 Шум изображения

Большое число систем получения изображения работает посредством преобразования частиц одного типа в частицы другого типа. Например, рассмотрим, что происходит в медицинских рентгеновских системах получения изображения. Внутри

рентгеновской трубки, *электроны* падают на металлическую цель, вырабатывая *рентгеновские лучи*. После прохождения сквозь пациента рентгеновские лучи попадают в датчик на вакуумной трубке известный как усилитель изображения. Здесь рентгеновские лучи последовательно преобразуются в *фотоны света*, затем *электроны*, а затем обратно в *фотоны света*. Эти фотоны света попадают в камеру, где они преобразуются в *электроны* в ямах в ПЗС. В каждой из этих промежуточных форм изображение представляется конечным числом частиц дающих в соответствии с уравнением (25.1) добавочный шум. Конечное отношение сигнал–шум отражает объединенный шум *всех* стадий преобразования; однако, одна стадия обычно доминирует. Это - стадия с *наихудшим* отношением сигнал-шум, поскольку она обладает *наименьшим* количеством частиц. Эта ограничивающая стадия называется **квантовым стоком**.

В системах ночного видения квантовым стоком является количество фотонов света, которое может быть захвачено камерой. Чем темнее ночь, тем больше шума на конечном изображении. Подобным примером являются медицинские рентгеновские изображения; квантовым стоком является количество рентгеновских лучей попадающих на датчик. Большие уровни радиации дают менее зашумленные изображения ценой большего облучения пациента.

Когда шум Пуассона является основным шумом в изображении? Он доминирует всякий раз, когда шум квантового стока больше шума других источников системы, таких как электроника. Например, рассмотрим типичную камеру на ПЗС с отношением сигнал–шум порядка 300. То есть, шум от предварительного усилителя ПЗС составляет 1/300 полной шкалы сигнала. Эквивалентный шум будет возникать, если квантовый сток системы содержит 90000 частиц на пиксель. Если квантовый сток имеет меньшее число частиц, в системе будет доминировать шум Пуассона. Если квантовый сток имеет большее число частиц, будет доминировать шум предварительного усилителя. Соответственно, большинство ПЗС разрабатываются на полную емкость ямы от *100000* до *1000000* электронов, сводя шум Пуассона к минимуму.

Морфологическая обработка изображения

Идентификация объектов в пределах изображения может быть очень трудной задачей. Одним из способов упрощения проблемы является преобразование шкалы серого изображения в *двоичное* **изображение**, в котором каждый пиксель ограничивается значениями либо *0*, либо *1*. Методы, используемые в этих двоичных изображениях, идут под такими названиями как: **капельный анализ**, **анализ связанности** и **морфологическая обработка изображения** (от греческого слова *morphe*, означающего очертание или форма). Основы морфологической обработки лежат в области математически строгой *теории множеств*; однако, такой уровень сложности требуется редко. Большинство морфологических алгоритмов представляют собой простые логические операции и в основном *специально подбираются только к данному конкретному случаю*. Другими словами, каждое приложение требует сделанного по заказу решения, полученного методом проб и ошибок. Обычно это больше искусство, чем наука. Вместо стандартных алгоритмов и формальных математических свойств используется мешок фокусов. Вот несколько примеров.

На рис. 25.10а показан пример двоичного изображения. Это могло бы представлять вражеский танк в инфракрасном изображении, астероид на космической фотографии или подозрительную опухоль на медицинском рентгеновском снимке. Каждый пиксель фона отображается как белый, тогда как каждый пиксель объекта отображается как черный. Часто двоичные изображения формируются при помощи задания порога шкалы серого изображения; пиксели, значения которых выше порога, устанавливаются в *1*, тогда как пиксели со значениями ниже порога устанавливаются в *0*. Обычно, до задания порога изображения в шкале серого обрабатываются линейными методами. Например,

сглаживание освещенности (описано в Главе 24) часто может улучшить качество первоначального двоичного изображения.

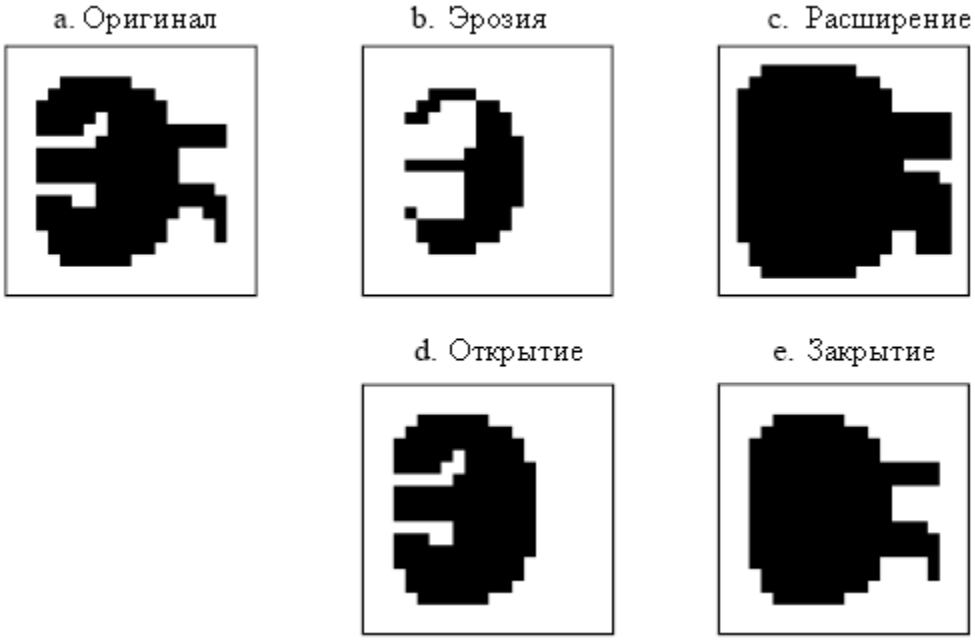


Рис. 25.10 Морфологические операции

На рис. 25.10b и рис. 25.10c показано, как изменяется изображение с помощью двух наиболее частых морфологических операций **эрозии** и **расширения**. В эрозии, каждый пиксель объекта, соприкасающийся с пикселем фона, заменяется на пиксель фона. В расширении, каждый пиксель фона, соприкасающийся с пикселем объекта, заменяется на пиксель объекта. Эрозия делает объект меньше и может разъединить единый объект на множество объектов. Расширение делает объект больше и может соединить множественные объекты в один.

Как показано на рис. 25.10d, **открытие** определяется как эрозия, сопровождаемая расширением. На рис. 25.10e показана противоположная операция - **закрытие**, определяемая как расширение, сопровождаемое эрозией. Как иллюстрируется данными примерами, открытие удаляет маленькие островки и тонкие нити *пикселей объекта*. Аналогично, закрытие удаляет островки и тонкие нити *пикселей фона*. Эти методы полезны для обработки зашумленных изображений, где некоторые пиксели имеют неверное двоичное значение. Например, может быть известным заранее, что объект не может содержать "дыру" или, что границы объекта должны быть гладкими.

На рис. 25.11 показан пример морфологической обработки. На рис. 25.11a представлено двоичное изображение отпечатка пальца. Алгоритмы, разработанные для анализа этого рисунка, позволяют поставить в соответствие индивидуальный отпечаток пальца с отпечатками, находящимися в базе данных. Типичным шагом таких алгоритмов, показанным на рис. 25.11b, является операция, называемая **скелетизацией**. Она упрощает изображение, *удаляя* избыточные пиксели; то есть, изменяет соответствующие пиксели от черных к белым. В результате каждый гребень превращается в тонкую линию шириной всего в один пиксель.

В таблице 25.1 и таблице 25.2 приведена программа скелетизации. Даже притом, что изображение отпечатка пальца двоичное, оно хранится в массиве, в котором каждый пиксель может изменяться от 0 до 255. Черному пикселю присваивается значение *0*, тогда как белому пикселю присваивается значение *255*. Как показано в таблице 25.1, алгоритм состоит из 6 итераций, которые постепенно разрушают гребни до тонкой линии. Число

итераций находится методом проб и ошибок. Альтернативой может быть прекращение работы в тот момент, когда очередная итерация не приводит ни к каким изменениям.

Рис. 25.11 Двоичная скелетизация

Во время итерации каждый пиксель в изображении оценивается на предмет возможности его *удаления*, для того, чтобы изменить его от черного до белого, пиксель должен отвечать ряду критериев. Цикл в строках с 200 по 240 предназначен для обхода каждого пикселя в изображении, тогда как программа в таблице 25.2 предназначена для оценки пикселя. Если рассматриваемый пиксель не удаляем, подпрограмма ничего не делает. Если пиксель удаляем, подпрограмма изменяет его значение от *0* к *1*. Это указывает, что пиксель все еще черный, но в конце итерации будет изменен до белого. После того как все пиксели будут оценены, в строках с 260 по 300 происходит изменение значений помеченных пикселей от *1* до *255*. В результате этого двух шагового процесса, толстые гребни равномерно эродируют по всем направлениям быстрее, чем у рисунка, основывающегося на сканировании строк и столбцов.

Таблица 25.1

```
100 'SKELETONIZATION PROGRAM
110 'Object pixels have a value of 0 (displayed as black)
120 'Background pixels have a value of 255 (displayed as white)
130 '
140 DIM X%[149,149]                 'X%[ , ] holds the image being processed
150 '
160 GOSUB XXXX                      'Mythical subroutine to load X%[ , ]
170 '
180 FOR ITER% = 0 TO 5              'Run through six iteration loops
190 '
200 FOR R% = 1 TO 148               'Loop through each pixel in the image.
210 FOR C% = 1 TO 148               'Subroutine 5000 (Table 25.2) indicates which
220 GOSUB 5000                      'pixels can be changed from black to white,
230 NEXT C%                         'by marking the pixels with a value of 1.
240 NEXT R%
250 '
260 FOR R% = 0 TO 149               'Loop through each pixel in the image changing
270 FOR C% = 0 TO 149               'the marked pixels from black to white.
280 IF X%(R%,C%) = 1 THEN X%(R%,C%) = 255
```

```
290 NEXT C%
300 NEXT R%
310 '
320 NEXT ITER%
330 '
340 END
```

Решение удалить пиксель основывается на четырех правилах, содержащихся в подпрограмме, показанной в таблице 25.2. Чтобы изменить пиксель от черного к белому должны быть удовлетворены *все* эти правила. Первых три правила довольно просты, тогда как четвертое - весьма сложное. Как показано на рис. 25.12a, пиксель с местоположением [R,C] имеет восемь соседей. Четырех соседей в горизонтальном и вертикальном направлениях (отмечены 2, 4, 6, 8) часто называют **ближними соседями**. Диагональные пиксели (отмечены 1, 3, 5, 7), соответственно, называют **дальними соседями**. Четыре правила следующие:

Правило первое: Рассматриваемый пиксель в данный момент должен быть черным. Если пиксель уже белый, никакое действие не должно предприниматься.

Правило второе: По крайней мере, один из ближних соседей пикселя, должен быть белым. Это обеспечивает то, что эрозия толстых гребней будет происходить с внешней стороны. Другими словами, если пиксель черный и его окружают черные пиксели, то в этой итерации его следует оставить в покое. Почему используются только *ближние соседи*, а не *все* соседи? Ответ простой: прогон алгоритма обоими способами показывает, что этот лучше. Запомните, это очень типично для морфологической обработки изображений; для того чтобы определить работает ли один метод лучше, чем другой, используется метод проб и ошибок.

Правило третье: пиксель должен иметь более одного черного соседа. Если он у него только один, то это должен быть конец строки, и следовательно, пиксель не должен быть удален.

Правило четвертое: Пиксель не может быть удален, если это приводит к *разъединению* его соседей. Это потому, что каждый гребень должен превратиться в непрерывную линию, а не в группу разорванных сегментов. Как показано в примере на рис. 25.12, *связанные* означает, что все черные соседи соприкасаются друг с другом. Аналогично, *несвязанные* означает, что черные соседи образуют две или более групп.

Алгоритм для определения, связаны или не связаны соседи, основан на подсчете черно-белых переходов между примыкающими соседними пикселями в направлении *часовой стрелки*. Например, если пиксель 1 - черный, а пиксель 2 – белый, это рассматривается как черно-белый переход. Аналогично, если пиксель 2 - черный, а оба пикселя 3 и 4 - белые, то это также черно-белый переход. В целом существует восемь мест, где может происходить черно-белый переход. Для дальнейшей иллюстрации этого определения в примерах на рис. 25.12b и на рис. 25.12c возле каждого черно-белого перехода помещена звездочка. Ключом к этому алгоритму является то, что если соседи *связаны*, будет ноль или один черно-белый переход. Наличие более чем одного такого перехода показывает, что соседи *не связаны*.

В качестве дополнительного примера обработки двоичного изображения рассмотрим виды алгоритмов, которые могут оказаться полезными после скелетизации отпечатка пальца. Недостатком этого конкретного алгоритма скелетизации является то, что он оставляет значительное количество *пушинок*, торчащих в сторону от протяженных сегментов коротких ответвлений. Существует несколько различных подходов для устранения этих художеств. Например, программа может обойти изображение и удалить каждый пиксель на всех концах каждой линии. Эти пиксели распознаются по одному единственному черному соседу. Проделайте это несколько раз, и пушинки удалены ценой того, что каждая правильная линия стала короче. Более лучшим методом был бы проход

по изображению с распознаванием *ответвляющихся пикселей* (пикселей у которых более, чем два соседа). Начиная от каждого ответвляющегося пикселя, подсчитывается количество пикселей в каждом ответвлении. Если число пикселей в каждом ответвлении меньше, чем некоторое значение (скажем, 5), считается, что это пушинка и пиксели в ответвлении изменяются от черного к белому.

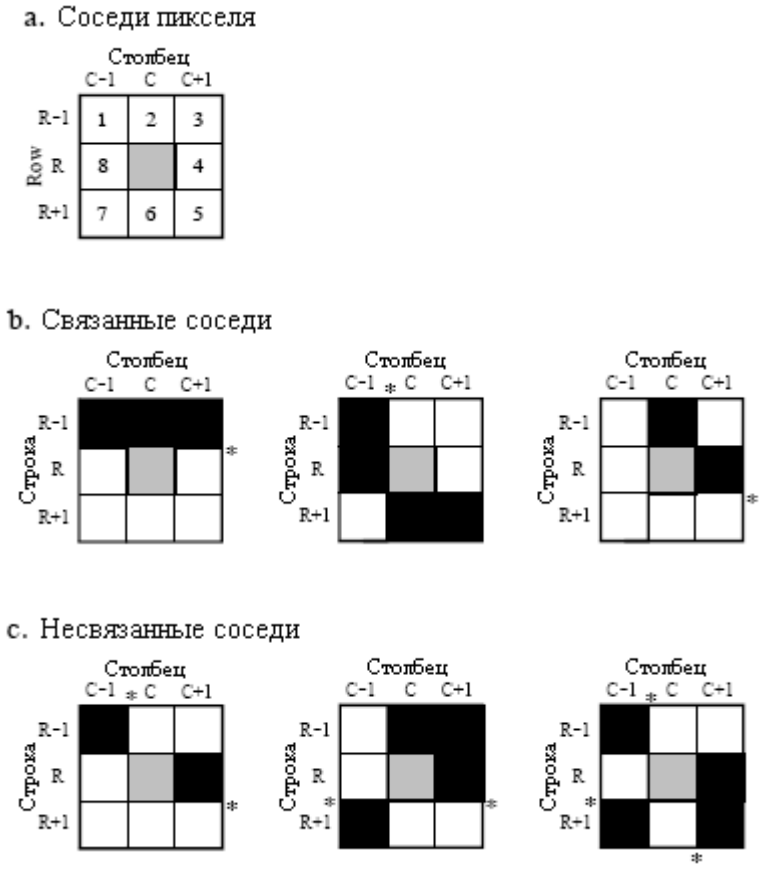


Рис. 25.12 Соседние пиксели

Таблица 25.2

```
5000 ' Subroutine to determine if the pixel at X%[R%,C%] can be removed.
5010 ' If all four of the rules are satisfied, then X%(R%,C%], is set to a value of 1,
5020 ' indicating it should be removed at the end of the iteration.
5030 '
5040 'RULE #1: Do nothing if the pixel already white
5050 IF X%(R%,C%) = 255 THEN RETURN
5060 '
5070 '
5080 'RULE #2: Do nothing if all of the close neighbors are black
5090 IF X%[R% -1,C% ] <> 255 AND X%[R% ,C%+1] <> 255 AND
          X%[R%+1,C% ] <> 255 AND X%[R% ,C% -1] <> 255 THEN RETURN
5100 '
5110 '
5120 'RULE #3: Do nothing if only a single neighbor pixel is black
5130 COUNT% = 0
5140 IF X%[R% -1,C% -1] = 0 THEN COUNT% = COUNT% + 1
5150 IF X%[R% -1,C%     ] = 0 THEN COUNT% = COUNT% + 1
5160 IF X%[R% -1,C%+1] = 0 THEN COUNT% = COUNT% + 1
```

```
5170 IF X%[R%    ,C%+1] = 0 THEN COUNT% = COUNT% + 1
5180 IF X%[R%+1,C%+1] = 0 THEN COUNT% = COUNT% + 1
5190 IF X%[R%+1,C%    ] = 0 THEN COUNT% = COUNT% + 1
5200 IF X%[R%+1,C% -1] = 0 THEN COUNT% = COUNT% + 1
5210 IF X%[R%    ,C% -1] = 0 THEN COUNT% = COUNT% + 1
5220 IF COUNT% = 1 THEN RETURN
5230 '
5240 '
5250 'RULE 4: Do nothing if the neighbors are unconnected.
5260 'Determine this by counting the black-to-white transitions
5270 'while moving clockwise through the 8 neighboring pixels.
5280 COUNT% = 0
5290 IF X%[R% -1,C% -1] = 0 AND X%[R% -1,C%    ] > 0 THEN COUNT% = COUNT% + 1
5300 IF X%[R% -1,C%    ] = 0 AND X%[R% -1,C%+1] > 0 AND X%[R%  ,C%+1] > 0
        THEN COUNT% = COUNT% + 1
5310 IF X%[R% -1,C%+1] = 0 AND X%[R%    ,C%+1] > 0 THEN COUNT% = COUNT% + 1
5320 IF X%[R%    ,C%+1] = 0 AND X%[R%+1,C%+1] > 0 AND X%[R%+1,C%  ] > 0
        THEN COUNT% = COUNT% + 1
5330 IF X%[R%+1,C%+1] = 0 AND X%[R%+1,C%    ] > 0 THEN COUNT% = COUNT% + 1
5340 IF X%[R%+1,C%    ] = 0 AND X%[R%+1,C% -1] > 0 AND X%[R%   ,C%-1] > 0
        THEN COUNT% = COUNT% + 1
5350 IF X%[R%+1,C% -1] = 0 AND X%[R%    ,C% -1] > 0 THEN COUNT% = COUNT% + 1
5360 IF X%[R%    ,C% -1] = 0 AND X%[R% -1,C% -1] > 0 AND X%[R%-1,C%   ] > 0
        THEN COUNT% = COUNT% + 1
5370 IF COUNT% > 1 THEN RETURN
5380 '
5390 '
5400 'If all rules are satisfied, mark the pixel to be set to white at the end of the iteration
5410 X%(R%,C%) = 1
5420 '
5430 RETURN
```

Следующий алгоритм мог бы преобразовать данные из формата *карты битов* в формат *карты векторов*. Это включает создание списка содержащихся в изображении гребней и содержащихся в каждом гребне пикселей. В форме карты векторов, в противоположность изображению, состоящему из множества не связанных пикселей, каждый гребень в отпечатке пальца обладает индивидуальной особенностью. Преобразование можно осуществить проходом по изображению в поисках конечных точек каждой линии, пикселей, имеющих только одного черного соседа. Каждая линия прослеживается от пикселя к связанному пикселю, начиная с конца. После того, как достигнут противоположный конец линии, все отслеженные пиксели считаются единым *объектом*, и, соответственно, рассматриваются в последующих алгоритмах.

Компьютерная томография

Основной проблемой в получении рентгеновских (или другой проникающей радиации) изображений является то, что от *трехмерного объекта* получается *двумерное изображение*. Это означает, что в конечном изображении могут происходить *наложения* структур, даже если в объекте они полностью отделены. Это особенно опасно в медицинской диагностике, где присутствует множество анатомических структур, которые могут загораживать то, что врач пытается разглядеть. В период 1930-ых годов, эту проблему штурмовали, скоординированным перемещением источника и датчика рентгеновского излучения во время формирования изображения. В соответствии с геометрией этих перемещений одна *плоскость* внутри пациента остается в фокусе, тогда как структуры, находящиеся за пределами этой плоскости, становятся не резкими. Это

аналогично тому, когда камера сфокусирована на объекте, находящемся в 5 футах, при этом объекты на расстоянии 1 и 50 футов выглядят нерезкими. Эти взаимосвязанные методы, основанные на нерезкости вследствие движения, теперь называются всеми **классической томографией**. Слово *томография* означает "*картина плоскости*".

Несмотря на то, что за более чем 50 лет она была хорошо развита, классическая томография применяется редко. Это связано с тем, что у нее существуют значительные ограничения: препятствующие объекты не *удаляются* из изображения, они становятся только смазанными. Качество получаемого в результате изображения обычно очень плохое для того, чтобы иметь практическое применение. Решением, которое долго искали, была система, которая может создавать изображение двумерного (2D) среза, проходящего через трехмерный (3D) объект, без каких-либо наложений других структур в трехмерном объекте.

Эта задача была решена в начале 1970-х с появлением техники, называемой **компьютерная томография (КТ)**. Компьютерная томография произвела революцию в области рентгеновской медицины с ее беспрецедентной способностью визуализации анатомической структуры тела. На рис. 25.13 показано типичное медицинское изображение КТ. Первоначально компьютерная томография вышла на рынок под названием *компьютерная аксиальная томография* или *КАТ сканер*. Сейчас в области медицины на этот термин смотрят неодобрительно, хотя Вы часто слышите, как он используется широкой публикой.

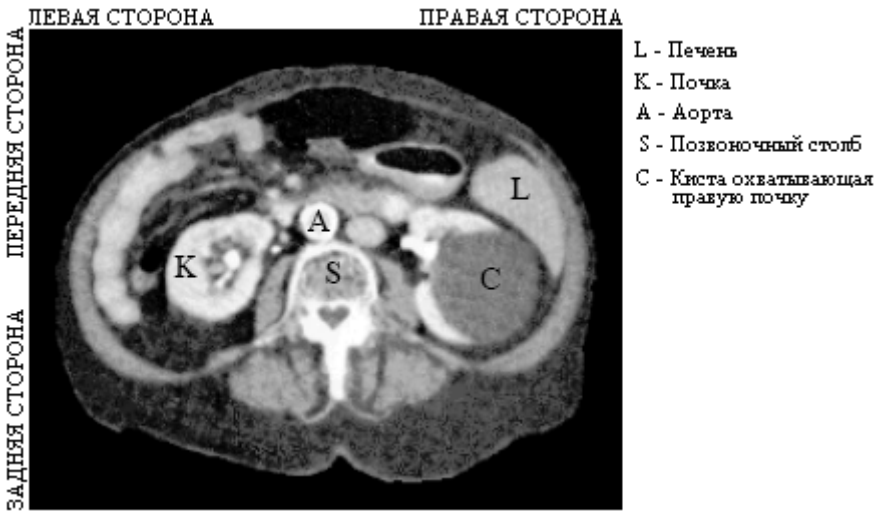


Рис. 25.13 Изображение компьютерной томографии

На рис. 25.14 иллюстрируется простая геометрия для получения среза КТ, проходящего через центр головы. Узкий, с карандаш, луч рентгеновского излучения проходит от рентгеновского источника к рентгеновскому датчику. Это значит, что измеренное на датчике значение зависит от общего количества помещенного где-нибудь на пути луча материала. Материалы наподобие костей и зубов поглощают большее количество рентгеновских лучей, давая в результате более слабый сигнал по сравнению с мягкими тканями и жиром. Как показано на иллюстрации, показания узла источник – датчик передается для получения **вида** (жаргон КТ) под этим конкретным углом. Хотя на этом рисунке показано, как получается всего один единственный вид, полное КТ сканирование требует от 300 до 1000 видов сделанных с шагом угла поворота от $0{,}3^{o}$ до $1{,}0^{o}$. Это осуществляется за счет установки источника и датчика рентгеновского излучения на вращающейся раме, окружающей пациента. Ключевой особенностью КТ системы сбора данных является то, что рентгеновские лучи проходят *только* через исследуемый срез тела. Это отличается от классической томографии, где рентгеновские

лучи проходят через структуры, которые Вы пытаетесь подавить в конечном изображении. Компьютерная томография не дает даже попасть в собираемые данные информации из других мест.

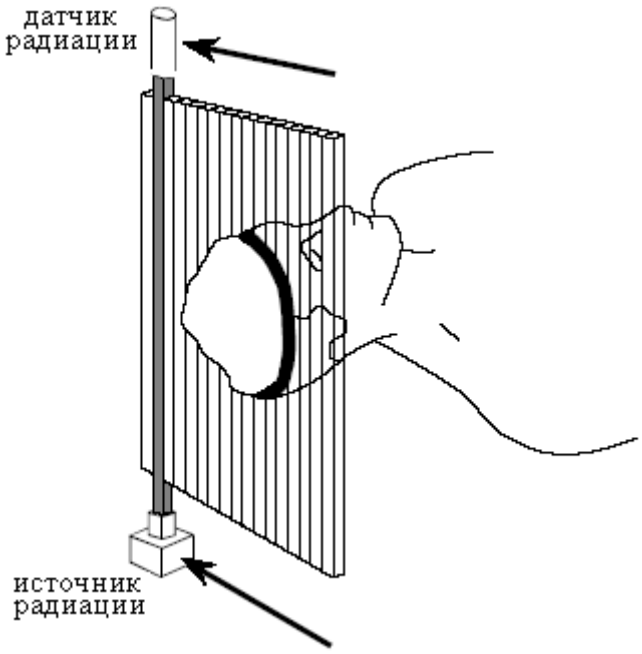


Рис. 25.14 Сбор данных КТ

Перед осуществлением реконструкции изображения обычно требуется несколько этапов предварительной обработки. Например, от каждого измерения рентгеновских лучей необходимо взять логарифм. Это связано с тем, что рентгеновские лучи экспоненциально затухают при прохождении через материал. Взятие логарифма обеспечивает линейную зависимость сигнала от характеристик измеряемого материала. Другим, используемым этапом предварительной обработки, является компенсация *полихроматичности* (наличия квантов с более чем одной энергией) рентгеновских лучей и *многоэлементности* датчика (в противоположность одноэлементному датчику, показанному на рис. 25.14). Хотя эти этапы, если взять метод целиком, и являются ключевыми, мы не будем обсуждать их в дальнейшем, поскольку они не относятся к алгоритмам реконструкции изображений.

На рис. 25.15 показана связь между измеренными видами и соответствующим изображением. Каждый снятый в КТ системе отсчет представляет собой сумму значений точек, расположенных вдоль луча, указывающего на этот отсчет. Например, вид 1 находится суммированием значений всех пикселей в каждой строке. Аналогично вид 3 находится суммированием значений всех пикселей в каждом столбце. Другие виды, такие как вид 2, представляют собой сумму значений пикселей вдоль лучей расположенных под углом.

Существует четыре основных подхода для вычисления изображения среза по заданному набору его видов. Они называются **КТ алгоритмы реконструкции**. Первый метод совершенно неосуществим, но приводит к лучшему пониманию проблемы. Он основан на **совместном** решении множества **линейных уравнений**. Для каждого измерения может быть записано одно уравнение. То есть, конкретный отсчет в конкретном профиле равен сумме значений конкретной группы пикселей в изображении. Для того чтобы вычислить N^2 неизвестных переменных (т.е. значений пикселей изображения размером $N \times N$), должно быть N^2 независимых уравнений, и следовательно, N^2 измерений. Большинство КТ сканеров снимают отсчетов примерно на 50% больше, чем

строго необходимо для этого анализа. Например, для реконструкции изображения размером 512×512 система должна отснять 700 видов при 600 отсчетах на каждый вид. *Излишне определяя* задачу в такой манере, в конечном изображении уменьшаются шум и блики. Проблемным в этом первом методе КТ реконструкции изображения является время вычисления. Совместное решение нескольких сотен тысяч линейных уравнений – пугающая задача.

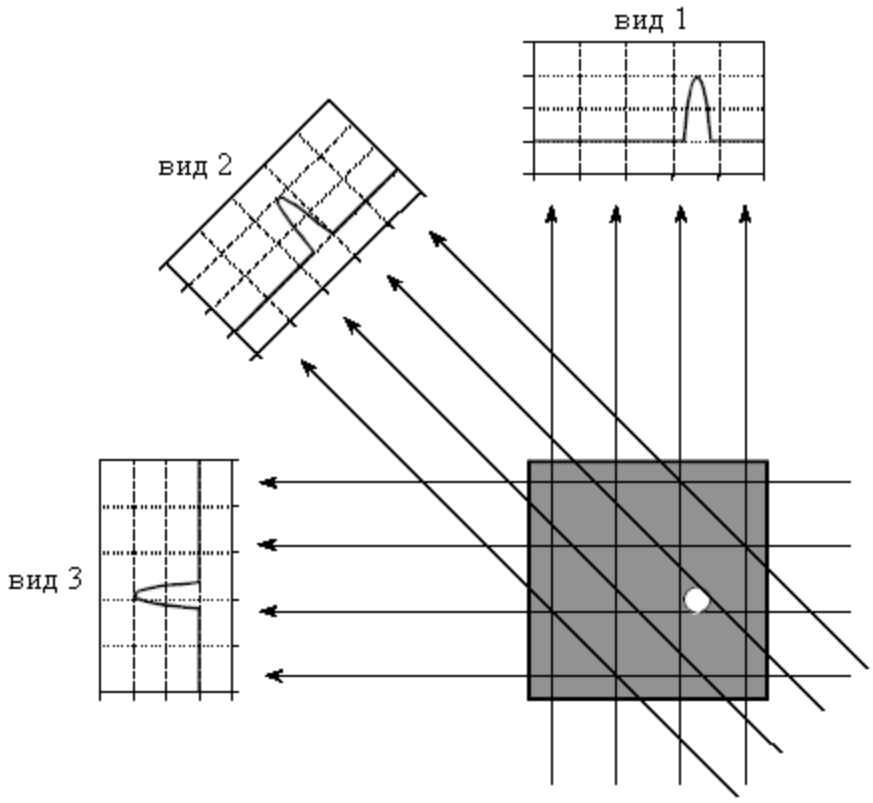


Рис. 25.15 КТ виды

Второй метод КТ реконструкции использует **итеративную** технику вычисления конечного изображения небольшими приращениями. Существует несколько разновидностей этого метода: техника алгебраической реконструкции, техника реконструкции совместными итерациями и итеративная техника наименьших квадратов. Разница между этими методами заключается в том, каким образом осуществляются последовательные коррекции: луч за лучом, пиксель за пикселем или, соответственно, совместная корректировка всего набора данных. В качестве примера этих методов рассмотрим метод алгебраической реконструкции.

Перед началом алгоритма алгебраической реконструкции все пиксели в массиве изображения устанавливаются в некоторое произвольное значение. Затем, для постепенного изменения массива изображения до соответствия профилю используется итеративная процедура. Цикл итерации состоит из прохода через каждую точку измеренных данных. Для каждого измеренного значения задается следующий вопрос: *каким образом могут быть изменены значения пикселей в массиве*, *чтобы сделать их согласующимися с этим конкретным измерением*? Другими словами, измеренный отсчет сравнивается с суммой значений пикселей в изображении вдоль луча, указывающего на отсчет. Если сумма вдоль луча меньше, чем измеренный отсчет, значение всех пикселей вдоль луча увеличивается. Аналогично, если сумма вдоль луча больше, чем измеренный отсчет, значение всех пикселей вдоль луча уменьшается. После полного завершения первого цикла итераций, между измеренными значениями и суммами значений всех

пикселей вдоль луча все еще будет оставаться ошибка. Это связано с тем, что изменения, сделанные для некоторого отдельного измерения разрушают все предыдущие коррекции. Идея состоит в том, что с повторением итераций ошибки становятся меньше, пока изображение не сойдется к правильному решению.

Итеративные методы обычно медленные, но когда более лучшие алгоритмы недоступны, они бывают очень полезны. Фактически метод алгебраической реконструкции был использован в первом коммерческом медицинском КТ сканнере EMI Mark I, выпущенном в 1972 году. Мы еще вернемся к итеративным методам в следующей главе по нейронным сетям. Развитие третьего и четвертого методов почти полностью вытеснило итеративные методы из коммерческих продуктов КТ.

Два последних алгоритма реконструкции изображений основаны на формальном математическом решении проблемы. Они являются элегантными примерами ЦОС. Третий метод называется методом **отфильтрованной обратной проекции**. Это модификация старого метода, называемого методом **обратной проекции** или **простой обратной проекции**. На рис. 25.16 показано, что метод простой обратной проекции представляет собой подход здравого смысла и очень бесхитростен. Индивидуальный отсчет проецируется обратно, посредством установки значений всех пикселей изображения вдоль луча, указывающего на этот отсчет, равными значению этого отсчета. Говоря не техническими терминами, обратная проекция формируется путем *размазывания* каждого вида по изображению в направлении противоположном тому, в котором он был первоначально получен. Конечное изображение является результатом суммирования обратной проекции всех видов.

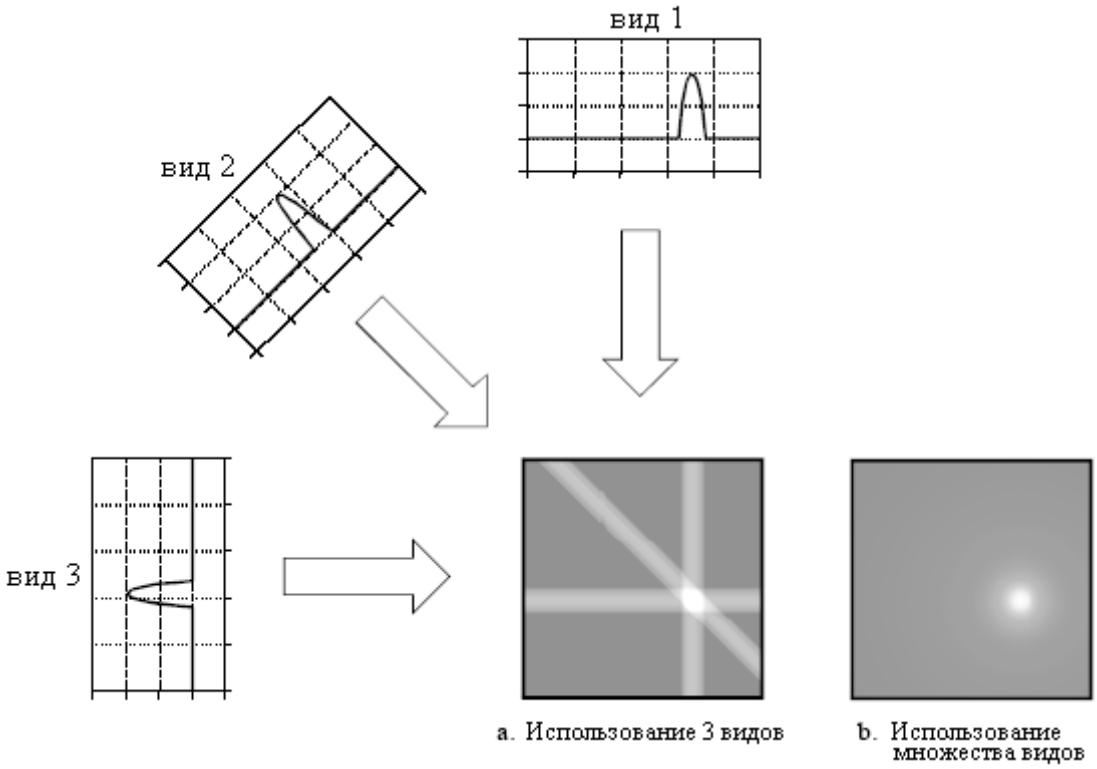


Рис. 25.16 Обратная проекция

Хотя обратное проецирование и является концептуально простым, оно не дает точного решения задачи. Как показано на рис. 25.16b, изображение, полученное методом обратной проекции, очень *размыто*. Одна единственная точка *исходного* изображения реконструировалась в виде окружности с уменьшающейся интенсивностью от центра.

Говоря более формальным языком, функция размыва точки для обратной проекции обладает круговой симметрией и уменьшается обратно пропорционально своему радиусу.

Метод отфильтрованной обратной проекции - это метод корректирующий размытость, с которой сталкиваются в простой обратной проекции. Как показано на рис. 25.17, перед тем как осуществить обратную проекцию, производится *фильтрация* каждого вида для того, чтобы противодействовать размытию ФРТ. То есть с целью создания набора *отфильтрованных видов* производится свертка каждого из одномерных видов с одномерным ядром фильтра. Затем, для получения реконструированного изображения, близкого приближения к "правильному" изображению, осуществляется обратная проекция этих отфильтрованных видов. Фактически, изображение, полученное с помощью отфильтрованной обратной проекции, *идентично* "правильному" изображению, когда существует *бесконечное* число видов с *бесконечным* числом точек на вид.

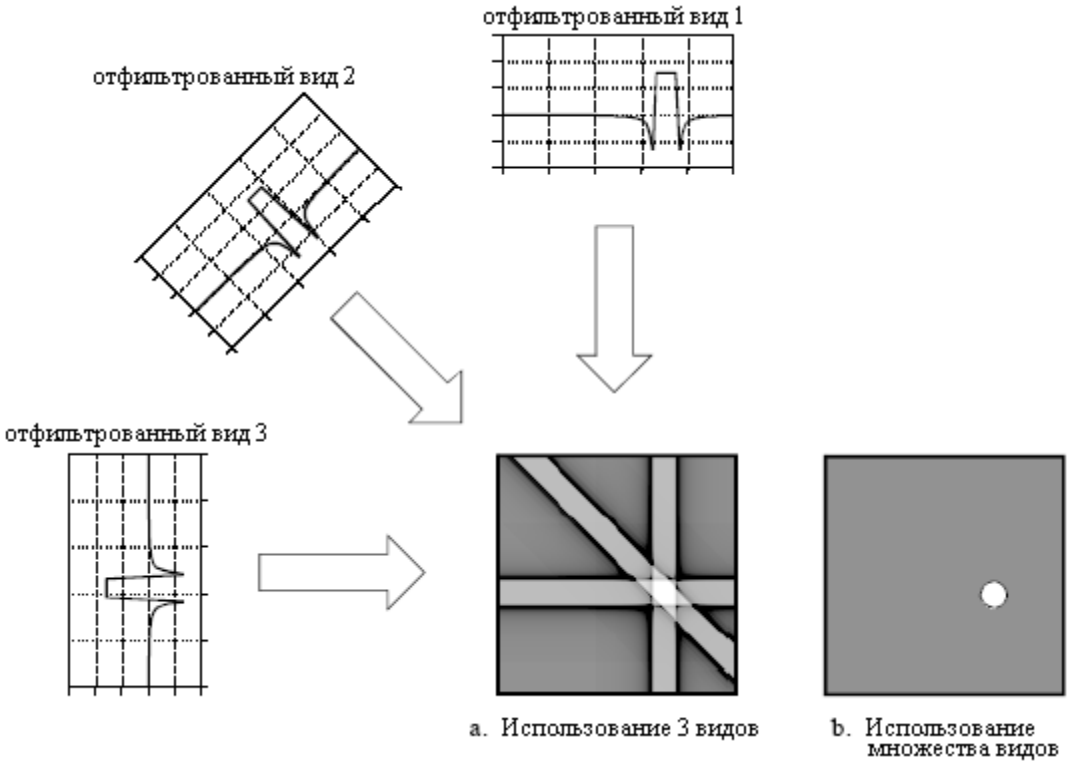


Рис. 25.17 Отфильтрованная обратная проекция

Кратко обсудим используемое в этом методе ядро фильтра. Для начала, обратите внимание, как после фильтрации изменились профили. В этом примере изображение представляет собой однородную белую окружность, охваченную черным фоном (пилюлеобразная форма). Каждый из полученных видов представляет собой пологое основание с закругленным возвышением над ним, представляющим белую окружность. Фильтрация приводит к изменению вида двумя важными способами. Во-первых, вершина возвышения делается плоской, что приводит в конечной обратной проекции к созданию *не изменяющегося* уровня сигнала внутри окружности. Во вторых, слева и справа от возвышения введены отрицательные выбросы. При обратной проекции эти отрицательные области противодействуют размытию.

Четвертый метод называется **Фурье реконструкцией**. КТ реконструкция заключается в установлении отношения в пространственной области между двумерным изображением и набором его одномерных видов. Взятием двумерного преобразования Фурье от изображения и одномерного преобразования Фурье от каждого из видов проблема может быть перенесена в частотную область. Как оказывается, отношение

между изображением и его видами в частотной области гораздо проще, чем в пространственной. Анализ этой проблемы в частотной области является краеугольным камнем технологии КТ, называемой **теоремой среза Фурье**.

На рис. 25.18 показано, как выглядит проблема в обеих пространственной и частотной областях. В пространственной области каждый вид находится интегрированием изображения вдоль лучей под конкретным углом. В частотной области спектр изображения представлен на этой иллюстрации двумерной сеткой. Спектр каждого вида (одномерный сигнал) представлен наложенными на сетку жирными линиями. Как показано расположением линий на сетке, теорема среза Фурье утверждает, что спектр вида идентичен значениям вдоль линии (среза) спектра изображения. Например, спектр вида 1 - это то же самое, что и центральный столбец спектра изображения, а спектр вида 3 - это то же самое, что и центральная строка спектра изображения. Обратите внимание, что спектр каждого вида располагается на сетке под тем же углом, под каким был первоначально получен вид. Все эти частотные спектры содержат отрицательные частоты и отображаются с расположенной в центре нулевой частотой.

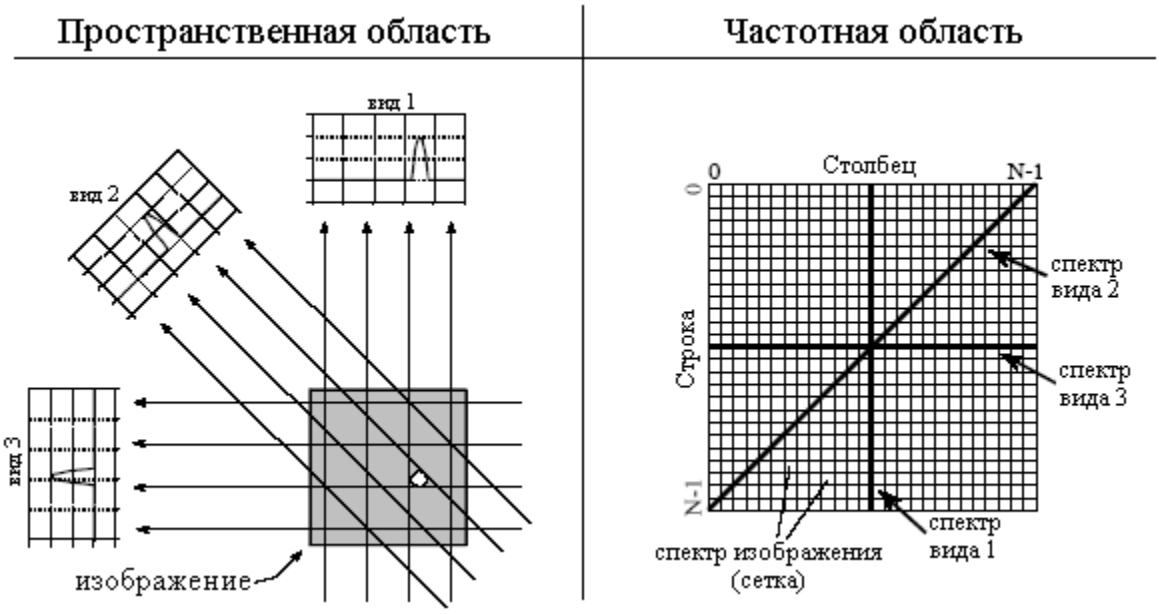


Рис. 25.18 Теорема среза Фурье

Фурье реконструкция изображения КТ требует трех этапов. Во-первых, от каждого вида берется одномерное преобразование Фурье. Во-вторых, в соответствии с теоремой среза Фурье эти спектры видов используются для вычислений двумерного частотного спектра изображения. Поскольку спектры видов организованы *радиально*, а спектр правильного изображения организован *прямоугольно*, потребуется интерполяционная подпрограмма, осуществляющая преобразование. В-третьих, с целью получения реконструированного изображения, от полученного спектра берется обратное преобразование Фурье.

Отмеченное "радиально – прямоугольное" преобразование также является ключом к пониманию отфильтрованной обратной проекции. Спектром обратной проекции изображения являются радиусы, тогда как спектром правильного изображения является прямоугольная сетка. Если мы сравним одну маленькую область радиального спектра с соответствующей областью прямоугольной сетки, то мы обнаружим, что значения отсчетов идентичны. Однако у этих областей будет разная *плотность отсчетов*. Правильный спектр повсюду имеет равномерно расположенные точки, что показано четным числом интервалов в прямоугольной сетке. Для сравнения, из-за своей радиальной

организации спектр обратной проекции обладает более высокой плотностью отсчетов в центре. Другими словами, возле оси спицы колеса становятся ближе друг к другу. Этот вопрос не оказывает влияния на Фурье реконструкцию, поскольку интерполяция зависит от величины *значений* ближних соседей, а не от их *плотности*.

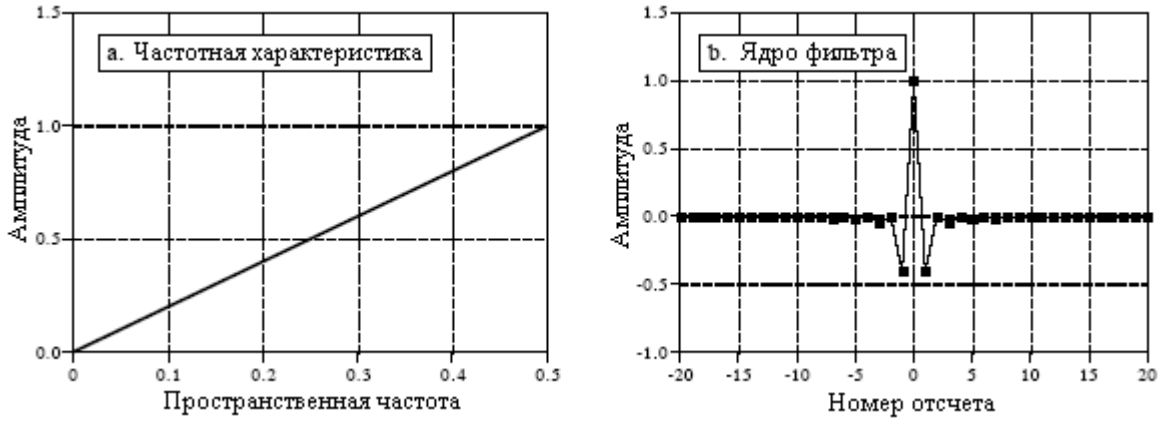


Рис. 25.19 Фильтр обратной проекции

В отфильтрованной обратной проекции фильтрация устраняет такую плотность с неравномерным расположением отсчетов. В частности, частотная характеристика фильтра должна быть *инверсной* плотности отсчетов. Поскольку плотность спектра обратной проекции *1/f*, частотная характеристика соответствующего фильтра *H[f] = f*. Эта частотная характеристика показана на рис. 25.19a. Затем, взятием от нее обратного преобразования Фурье находится ядро фильтра, показанное на рис. 25.19b. Математически ядро фильтра задается как:

$$h[0] = 1;$$

$$h[k] = 0, \text{ для четных значений } k; \qquad (25.2)$$

$$h[k] = \frac{-4/\pi^2}{k^2}, \text{ для нечетных значений } k.$$

Прежде чем покинуть тему компьютерной томографии следует упомянуть, что в области медицины существует несколько подобных технологий получения изображения. Все они обширно используют ЦОС. **Позитронная эмиссионная томография (ПЭТ)** использует введение пациенту слабо радиоактивного вещества, которое испускает *позитроны*. Сразу же после эмиссии позитрон аннигилирует с электроном, создавая два гамма луча, покидающих тело в строго противоположных направлениях. Датчики радиации расположенные вокруг пациента находят эти два взаимообратных гамма луча, определяя положение *линии* вдоль которой путешествуют гамма кванты. Так как точка, в которой были созданы гамма лучи должна находиться где-то на этой линии, то здесь могут быть использованы алгоритмы подобные алгоритмам КТ. Это дает изображение, которое выглядит похожим на КТ за исключением того, что его *яркость* связана с количеством находящегося в каждом месте радиоактивного вещества. Уникальным преимуществом ПЭТ является то, что радиоактивные вещества могут быть добавлены в различные субстанции, например глюкозу, которые некоторым образом используются телом. В этом случае реконструированное изображение будет связано с концентрацией

этой биологической субстанции. Это позволяет получать изображения *физиологии* тела, а не просто его *анатомии*. Например, могут быть получены изображения, показывающие, какие участки мозга человека вовлечены в различные мыслительные процессы.

Более прямым конкурентом компьютерной томографии является **магниторезонансная интроскопия (МРИ)**, которая теперь есть в большинстве главных клиник. Этот метод первоначально был разработан под названием **ядерный магнитный резонанс (ЯМР)**. Изменение названия было сделано для широкой публики, когда местные власти стали протестовать против использования в своих владениях всего *ядерного*. Часто это было невозможной задачей объяснить общественности, что термин *ядерный* просто относится к тому факту, что все атомы содержат *ядра*. МРИ сканирование проводится посредством помещения пациента в центр мощного магнита. Радиоволны совместно с магнитным полем заставляют резонировать определенные ядра в теле, что приводит к эмиссии вторичных радиоволн. Эти вторичные радиоволны оцифровываются, и формируется набор данных, используемый в алгоритмах реконструкции МРИ. Результатом является набор изображений, которые создаются почти так же, как и в компьютерной томографии. Преимущества МРИ многочисленны: хорошее разделение мягких тканей, гибкий выбор срезов и полное отсутствие потенциально опасного рентгеновского излучения. К отрицательным сторонам МРИ следует отнести то, что этот метод является более дорогим, чем КТ и плох для отображения костей и других твердых тканей. КТ и МРИ будут столпами получения медицинских изображений на многие годы вперед.

ГЛАВА

26

НЕЙРОННЫЕ СЕТИ (И БОЛЕЕ!)

Традиционная ЦОС базируется на *алгоритмах*, преобразующих данные из одной формы в другую с помощью шаг за шагом выполняемых процедур. Большинству из этих методов для функционирования необходимы также *параметры*. Например, рекурсивные фильтры используют рекурсивные *коэффициенты*, обнаружение характерных особенностей может быть выполнено с помощью корреляции и *порогов*, демонстрация изображения зависит от установок яркости и контрастности и т.д. Алгоритмы описывают то, что должно быть сделано, в то время как параметры обеспечивают эталонными нормами для того, чтобы оценивать данные. Надлежащий выбор параметров часто более важен, чем непосредственно алгоритм. Нейронные сети доводят эту идею до крайности, используя очень простые алгоритмы, но зато множество высоко оптимизированных параметров. Это революционный отход от традиционных столпов науки и инженерного дела: за математической логикой и теоретизированием следуют эксперименты. Нейронные сети заменяют стратегии решения проблем методом проб и ошибок, прагматическими решениями и методом “это работает лучше, чем то”. Эта глава представляет разнообразие проблем относительно выбора параметра, как в нейронных сетях, так и в алгоритмах более традиционной ЦОС.

Обнаружение цели

Ученым и инженерам часто необходимо знать присутствует ли конкретный объект или параметр. Например, геофизики исследуют землю в поисках нефти, врачи обследуют пациентов в поисках болезни, астрономы ищут во вселенной внеземные цивилизации и т.д. Эти проблемы обычно включают процесс сравнения полученных данных с порогом. Если порог превышен (объект или условие найдено), то полагают, что **цель** присутствует.

Например, предположим, что Вы изобрели прибор для обнаружения рака у человека. Аппарат вращается вокруг пациента, и на видео экране выскакивает число между 0 и 30. Низкие числа соответствуют здоровым субъектам, в то время как высокие числа указывают на присутствие злокачественной ткани. Вы находите, что прибор работает весьма хорошо, но не точно и иногда делает ошибку. Вопрос состоит в том, как Вам использовать эту систему на благо обследуемого пациента?

Рисунок 26.1 иллюстрирует систематический путь анализа этой ситуации. Предположим, прибор тестируется на двух группах: несколько сотен добровольцев, о которых известно, что они здоровы (не целевые), и несколько сотен добровольцев, о которых известно, что у них рак (целевые). Рис. 26.1a и 26.1b показывают результаты этого теста, демонстрируемые в виде гистограмм. Вообще здоровые субъекты дают более низкие числа, чем те, у которых есть рак (это хорошо), но между этими двумя распределениями есть некоторое перекрытие (это плохо).

Как обсуждалось в Главе 2, гистограмма может быть использована для оценки **функции распределения вероятности**, как это показано на рис. 26.1c. Представьте, например, что прибор используется на случайно выбранном здоровом субъекте. Из рис. 26.1c видно, что существует примерно 8%-й шанс, что результат теста будет 3, примерно

1%-й шанс, что результат теста будет 18 и т.д. (Этот пример не определяет, является ли выходная величина действительным числом, требующим *функцию распределения вероятности*, или целым, требующим *функцию массы вероятности*. Не переживайте об этом, здесь это не важно.).

Теперь подумайте о том, что случится, когда прибор используется на пациенте с неизвестным состоянием здоровья. Какое заключение мы можем сделать, если персона, которую мы никогда раньше не видели, получает значение 15? Есть у нее рак или нет? Мы знаем, что вероятность здоровой персоны, генерирующей 15 составляет 2,1%. Аналогично существует 0,7%-й шанс, что 15 выдаст персона, пораженная раком. Если никакая другая информация не доступна, мы сделали бы заключение, что у этого субъекта вероятность отсутствия рака в три раза больше, чем его наличие. То есть, результат теста 15 подразумевает 25% вероятность того, что этот субъект из целевой группы. Этот метод может быть обобщен для формирования кривой на рис.26.1d, причем вероятность того, что у субъекта есть рак, основана только на числе, выданном прибором [математически, *функция распределения вероятности*$|_t$ / (*функция распределения вероятности*$|_t$ + *функция распределения вероятности*$|_{nt}$)].

Если бы мы остановили анализ на этом месте, то мы бы сделали одну из наиболее обычных (и серьезных) ошибок в определении цели. Чтобы сделать кривую на рис.26.1d значимой, обычно должен быть принят во внимание другой источник информации. А именно, относительное число в тестируемой совокупности целевых пациентов по сравнению с не целевыми. Например, мы можем обнаружить, что только один из тысячи человек болен раком, который мы пытаемся выявить. Для включения этого в анализ амплитуда не целевой функции распределения вероятности на рис. 26.1c выверяется таким образом, чтобы площадь под кривой была 0,999. Аналогично, амплитуда целевой функции распределения вероятности выверяется так, чтобы сделать площадь под кривой равной 0,001. Затем рассчитывается кривая, показанная на рис. 26.1d, для того чтобы дать, также как и раньше, вероятность того, что у пациента рак.

Пренебрежение этой информацией является серьезной ошибкой, поскольку это очень сильно влияет на то, как интерпретируются результаты теста. Другими словами, кривая на рис. 26.1d меняется коренным образом, когда в рассмотрение включается доминирующая информация. Например, если часть популяции, имеющей рак, составляет 0,001, результат теста 15 соответствует вероятности всего в 0,025 %, что у этого пациента рак. Это сильно отличается от вероятности в 25%, найденной, полагаясь только на выходные данные машины.

Этот метод, преобразования выходных величин в вероятности, может быть полезен для понимания проблемы, но это не основной путь, которым заканчивается обнаружение цели. Большинство приложений, относительно присутствия цели, требуют принятия решения типа да/нет, поскольку *да*, будет выражаться в одном действии, а *нет,* будет выражаться в другом. Принятие решения реализуется путем сравнения выходной величины испытания с некоторым **порогом**. Если значение выходной величины выше порога, говорят что испытание **положительно**, показывая, что цель присутствует. Если значение выходной величины ниже порога, говорят что испытание **отрицательно**, показывая, что цель отсутствует. В нашем примере с болезнью раком отрицательный результат теста означает, что пациенту говорят, что он здоров и отправляют домой. Когда результат теста положителен, будут проделаны дополнительные тесты (анализы – прим. перев.) такие, как получение образца ткани с помощью введения иглы для биопсии.

Поскольку целевые и не целевые распределения перекрываются, некоторые результаты тестов не будут корректными. То есть некоторые пациенты, отправленные домой, будут действительно иметь рак, а некоторые пациенты, отправленные на дополнительное обследование, будут здоровы. На жаргоне обнаружения цели *корректная* классификация называется **истинной**, в то время как *некорректная* классификация называется **ложной**. Например, если у пациента есть рак, и тест правильно определяет его

состояние, говорят что он **истинно-положительный**. Аналогично, если у пациента нет рака, и тест показывает, что рак не присутствует, говорят что он **истинно-отрицательный**. **Ложноположительный** случай встречается тогда, когда у пациента нет рака, но тест ошибочно указывает, что рак есть. Это приводит в результате к беспричинному беспокойству к душевной боли и к затратам на дополнительные обследования. Встречаются и даже худшие сценарии, в **ложноотрицательных** случаях, где рак присутствует, а тест показывает, что пациент здоров. Как мы все знаем, рак оставленный без лечения может вызвать массу проблем со здоровьем, включая и преждевременную смерть.

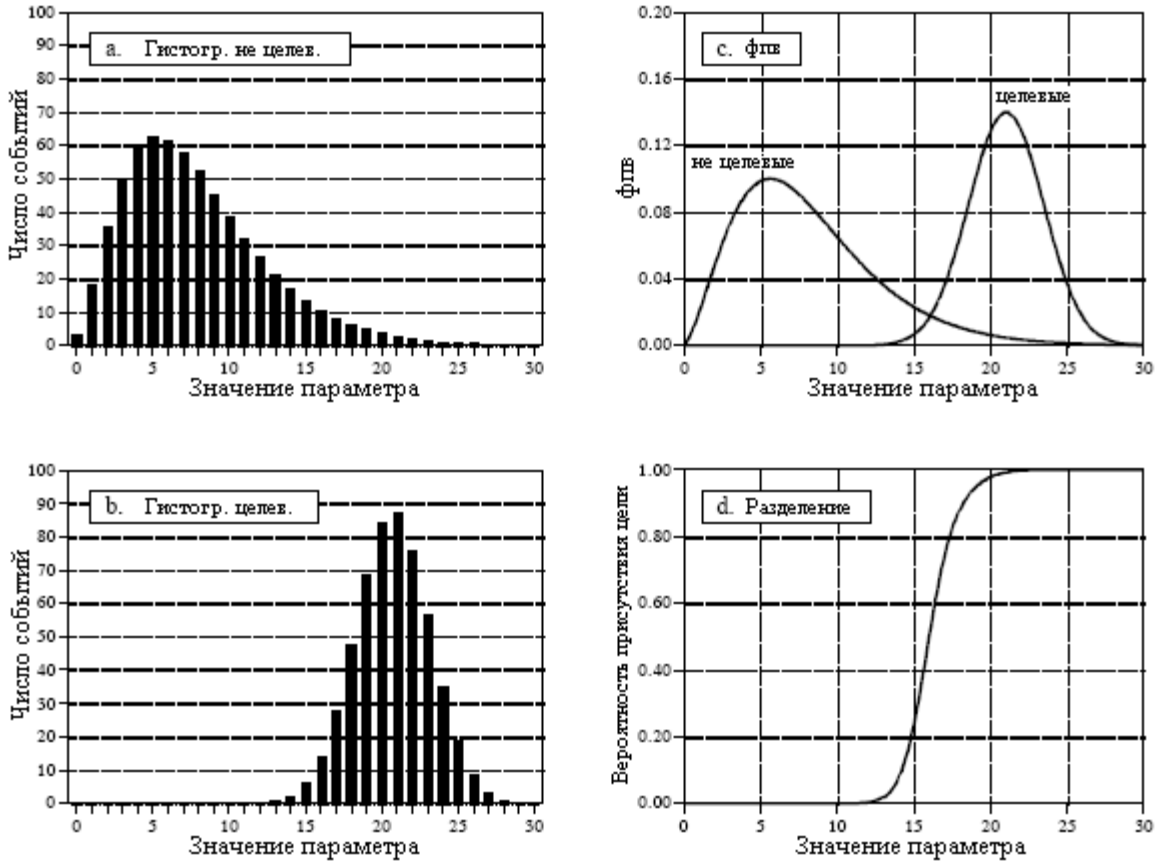


Рис. 26.1 Вероятность обнаружения цели

Страдания человека, вытекающие из этих двух типов ошибок, делают выбор при помощи порога деликатным балансирующим актом. Какое же число ложноположительных случаев можно допустить, чтобы снизить число ложноотрицательных? Рис.26.2 показывает графический путь оценки этой проблемы, кривую **рабочей характеристики приемника** (РХП). График кривой рабочей характеристики приемника показывает процент целевых сигналов, представленных как положительные (чем выше, тем лучше), по сравнению с процентом не целевых сигналов, ошибочно представленных, как положительные (чем ниже, тем лучше), для различных значений устанавливаемого порога. Другими словами каждая точка на этой кривой представляет одно возможное соотношение истинно-положительного и ложноположительного действий.

Рисунки от 26.2a до 26.2d показывают четыре разных установленных пороговых уровня для нашего примера обнаружения рака. Например, посмотрите на рис. 26.2b, где порог установлен на 17. Запомните, каждый тест, который выдает выходную величину *больше* установленного порога, рассматривается как *положительный* результат.

Приблизительно 13% площади не целевого распределения превышает порог (т.е. находится *справа* от порога). Из всех пациентов, не болеющих раком, только 87% будут

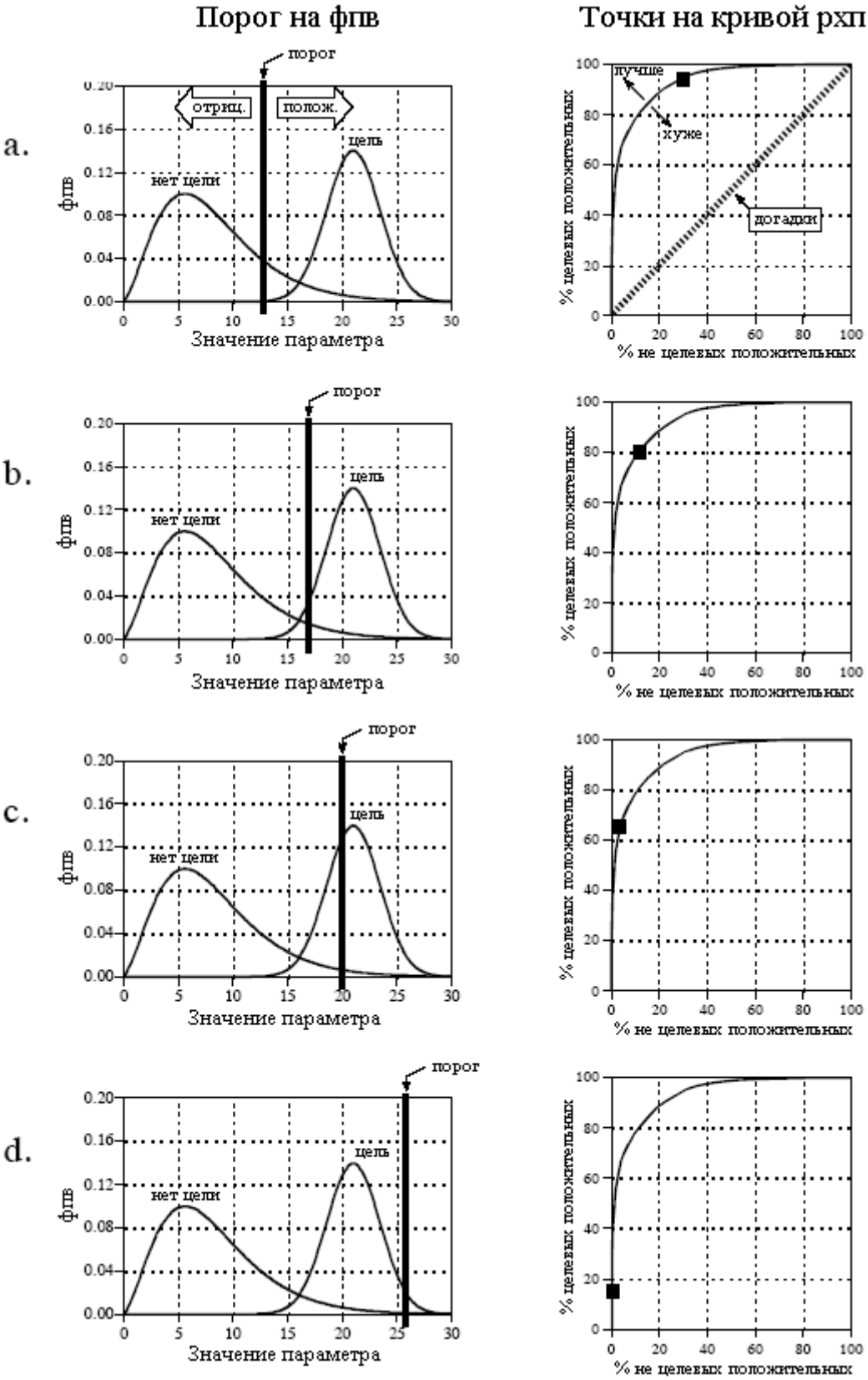


Рис. 26.2 Связь между кривой РХП и ФПВ

Рассматриваться, как персоны с отрицательным результатом (т.е. истинно-отрицательным), тогда как 13% будут рассматриваться как с положительным результатом (т.е. ложноположительным). Для сравнения, примерно 80% площади целевого распределения превышает порог. Это означает, что 80% больных раком дадут положительный результат тестирования (т.е. истинно-положительный). Другие 20% у кого рак есть, будут ошибочно рассматриваться как персоны с отрицательным результатом (т.е. ложноотрицательные). Как показано на кривой рабочей характеристики приемника на рис. 26.2b, этот порог выражается точкой на кривой при: *% не целевых положительный = 13%*, *% целевых положительный = 80%*.

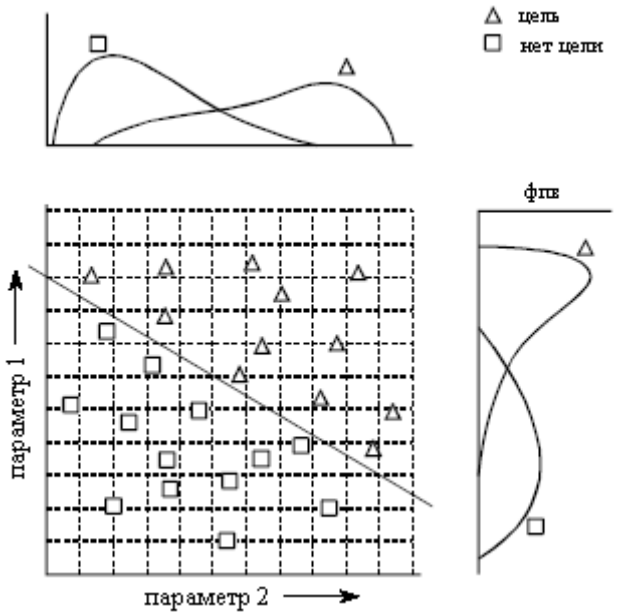


Рис. 26.3 Пример двухпараметрического пространства

Чем более эффективен процесс обнаружения, тем сильнее кривая рабочей характеристики приемника будет выгибаться к левому верхнему углу графика. Результаты чистых догадок располагаются на прямой линии, проходящей под углом 45^{o} по диагонали. Если установить относительно низкий порог, как показано на рис 26.2a, это приведет к тому, что почти все целевые сигналы будут обнаруживаться. Это дается ценой большого числа ложных тревог (ложноположительных). Как показано на рис. 26.2d, установка относительно высокого порога дает обратную ситуацию: всего несколько ложных тревог, но и много пропущенных целей.

Эти методы анализа полезны для понимания последствий выбора порога, но окончательное решение основано на том, что выберет конкретный *человек*. Предположим, Вы первоначально установите порог аппарата, обнаруживающего рак на некотором уровне, который, как Вам чувствуется, является подходящим. После того как большое количество пациентов было просвечено системой, Вы разговариваете с дюжиной, или около того, пациентов, которые были отнесены к ложноположительной группе. Услышанное, о том, как *Ваша* система без необходимости разрушила жизни этих людей, глубоко подействовало на Вас, мотивировав Вас к увеличению порога. В конечном счете, Вы сталкиваетесь с ситуацией, которая заставляет Вас испытывать еще более плохие чувства: Вы разговариваете с пациентом неизлечимо больным раком, который *Ваша* система не сумела обнаружить. Вы реагируете на этот трудный жизненный опыт сильным снижением порога. По мере того как идет время, и эти события многократно повторяются, порог постепенно движется к значению *равновесия*. То есть ложноположительный уровень, умноженный на фактор значимости (снижение порога), уравновешивается

ложноотрицательным уровнем, умноженным на другой коэффициент значимости (увеличение порога).

Этот анализ может быть распространен и на приборы, которые выдают больше, чем одно выходное значение. Например, предположим, что система обнаружения рака работает при помощи создания рентгеновского снимка субъекта, сопровождающегося автоматическим анализом с помощью алгоритмов идентификации опухолей. Алгоритмы определяют подозрительные области, а затем, в целях оценки, измеряют ключевые характеристики. Предположим, например, что мы измеряем *диаметр* подозрительной области (параметр 1) и его *яркость* на изображении (параметр 2). Далее, предположим, что наше исследование указывает, что опухоли значительно больше и ярче, чем нормальная ткань. В качестве первой попытки, мы могли бы пойти путем ранее представленного анализа с помощью кривой рабочей характеристики приемника и для каждого параметра найти приемлемый порог. Затем мы могли бы классифицировать тест, как положительный только в том случае, если бы мы встретили оба критерия: параметр 1 больше, чем некоторый порог *и* параметр 2 больше, чем некоторый другой порог.

Этот метод соотнесения параметров раздельно с привлечением в последствии логических функций (И, ИЛИ и т.д.) является наиболее типичным. Тем не менее, он очень неэффективен, и существуют методы гораздо лучше. Рис. 26.3 показывает, почему это так. На этом рисунке каждый треугольник представляет единичный случай наличия цели (пациент болен раком), вычерченный в месте, соответствующем значению его двух параметров. Аналогично, каждый квадрат представляет единичный случай отсутствия цели (у больного нет рака). Как показано на графике функции распределения вероятности, на стороне каждой оси оба параметра имеют большое перекрытие между целевым и не целевым распределениями. Другими словами, каждый параметр, взятый индивидуально является плохим предсказателем рака. Объединение двух параметров с помощью простых логических функций будет обеспечивать только небольшое улучшение. Это особенно интересно, поскольку оба параметра содержат информацию для *идеального* отделения целей от не целей. Такое отделение делается так, как показано на рисунке, вычерчиванием диагональной линии между двумя группами.

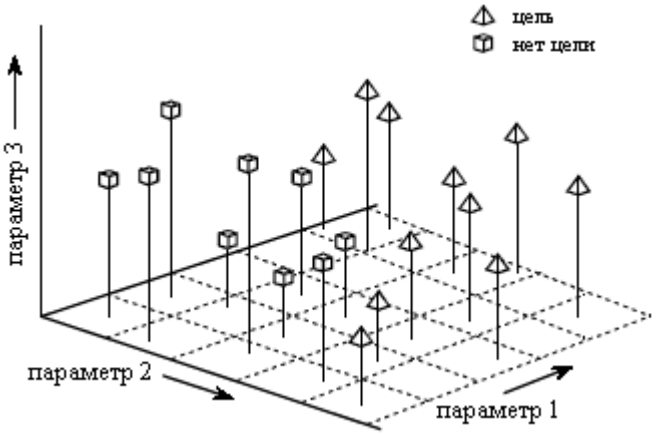


Рис. 26.4 Пример трехпараметрического пространства

На жаргоне этой области такой тип системы координат называется **параметрическим пространством**. Например, двумерная плоскость в нашем примере могла бы быть названа пространством диаметр-яркость. Идея состоит в том, что цели будут занимать одну область параметрического пространства, в то время как не цели будут занимать другую. Раздел между этими двумя областями может быть настолько прост, как, например, прямая линия, или столь же сложен, как, например, замкнутые области с нерегулярными границами. Рисунок 26.4 показывает следующий уровень

сложности, трехмерное пространство, представляемое осями *x*, *y* и *z*. Например, это могло бы соответствовать системе обнаружения рака, которая измеряет *диаметр*, *яркость* и некоторый третий параметр, скажем, *резкость границ* изображения. Так же, как и в двумерном случае, главная мысль здесь состоит в том, что члены целевых и не целевых групп будут (надеемся) занимать разные области пространства, позволяя разделить эти две группы. В трех измерениях, области разделяются плоскостями и кривыми поверхностями. Чтобы описать пространство параметра с больше чем тремя измерениями, часто используется термин **гиперпространство** (сверх, выше, или за пределами нормального пространства). Математически, гиперпространства не отличаются от одно, двух или трехмерных пространств; однако, с ними связана одна практическая проблема, их нельзя показать в графической форме в нашей трехмерной вселенной.

Порог, выбранный для проблемы с одним параметром, не может (обычно) классифицироваться как правильный или неправильный. Это в силу того, что каждое значение порога дает уникальную комбинацию ложноположительных и ложноотрицательных, то есть некоторую точку на кривой рабочей характеристики приемника. Это подменяет одну цель другой, и не дает абсолютно правильного ответа. С другой стороны, параметрические пространства с двумя или более параметрами могут определенно иметь деление между областями неправильной формы. Например, представьте увеличение числа точек данных на рис. 26.3, обнаруживающих небольшое перекрытие между целевыми и не целевыми группами. Было бы возможно переместить линию порога между группами, чтобы соотнести число ложноположительных по сравнению с числом ложноотрицательных. То есть диагональная линия была бы перемещена вверх вправо, или вниз влево. Однако было бы неправильно смещать линию, потому что это увеличило бы *оба* типа ошибок.

Как следует из этих примеров, традиционный подход к обнаружению цели (иногда называемый **распознаванием образов**) является двух шаговым процессом. Первый шаг называется **выделением характеристик**. Он использует алгоритмы, чтобы сократить сырые данные до нескольких параметров, таких как диаметр, яркость, резкость края изображения и т.д. Эти параметры часто называются **характеристиками** или **классификаторами**. Выделение характеристик необходимо для уменьшения количества данных. Например, медицинское рентгеновское изображение может содержать больше чем миллион пикселей. Цель выделения характеристик состоит в том, чтобы преобразовать информацию в более сконцентрированную и управляемую форму. Этот этап разработки алгоритма представляет собой больше искусство, чем науку. Требуется огромный опыт и навык, чтобы посмотреть на проблему и сказать: "*Вот классификаторы, которые лучше всего представляют информацию*". Метод проб и ошибок играет здесь значительную роль.

На втором шаге, проводится оценка классификаторов, чтобы определить, присутствует цель или нет. Другими словами, используется некоторый метод, для разделения пространства параметра на область, которая соответствует целям, и область, которая соответствует не целям. Это вполне выполнимо для одно или двухмерного пространства; на график наносятся известные точки данных (так, как на рис. 26.3), а затем на глазок разделяются области. После чего разделение записывается в компьютерную программу в виде уравнения или каким-нибудь другим образом, отделяющим одну область от другой. В принципе, тот же самый метод может быть применен и к трехмерному параметрическому пространству. Проблема в том, что трехмерные графики очень трудны для понимания человеком и визуального восприятия (как на рис. 26.4). Предостережение: не пытайтесь проделывать это с гиперпространством; Ваши мозги взорвутся!

Короче говоря, нам нужна машина, которая может выполнить деление многопараметрического пространства на примерах целевых и не целевых сигналов. Эта

идеальная система обнаружения цели особенно близка основной теме этой главы, *нейронной сети*.

Архитектура нейронной сети

Люди и другие животные обрабатывают информацию при помощи *нейронных сетей*. Они формируются из *миллиардов* **нейронов** (нервных клеток), обменивающихся короткими электрическими импульсами, называемыми **потенциалами действия**. Алгоритмы компьютера, которые подражают этим биологическим структурам, чтобы отличать их от требухи внутри животных, формально называются **искусственными нейронными сетями**. Однако большинство ученых и инженеров не такие формалисты и используют термин нейронная сеть для обозначения и биологических и небиологических систем.

Исследование нейронной сети мотивируется двумя желаниями: получить лучшее понимание мозга человека, и создать компьютеры, которые могут иметь дело с решением абстрактных и неопределенных задач. Обычные компьютеры, например, испытывают трудности в понимании речи и распознавании человеческих лиц. Для сравнения, люди чрезвычайно хорошо справляются с этими задачами.

На сегодняшний день было испытано множество структур различных нейронных сетей, некоторые основаны на подражании тому, что биологи видят под микроскопом, некоторые основаны на большом математическом анализе. На рис. 26.5 показана наиболее часто используемая структура. Эта нейронная сеть формируется из трех слоев, называемых **входной слой**, **скрытый слой**, и **выходной слой**. Каждый слой состоит из одного или более **узлов**, представленных на этой схеме небольшими кружочками. Линии между узлами показывают поток информации от одного узла к следующему. В этом конкретном типе нейронной сети информация передается только от входа к выходу (т.е. слева направо). Другие типы нейронных сетей имеют более сложные соединения, такие как соединения обратной связи.

Узлы входного слоя **пассивны**, это означает, что они не модифицируют данные. Они получают одно значение на вход и копируют его на свои многочисленные выходы. Для сравнения, узлы скрытого и выходного слоев являются **активными**. Это означает, что они модифицируют данные, как показано на рис. 26.6. Переменные $X1_1$, $X1_2$ … $X1_{15}$ содержат требующие оценки данные (см. рис 26.5). Это могут быть, например, значения пикселя изображения, отсчета звукового сигнала, рыночная цена акции за несколько дней подряд и т.д. Они также могут быть выходными данными некоторого другого алгоритма, типа классификаторов в нашем примере с обнаружением рака: диаметра, яркости, резкости края изображения и т.д.

Каждое значение от входного слоя копируется и посылается *всем* узлам скрытого слоя. Это называется полностью **взаимосвязанной структурой**. Как показано на рис. 26.6, значения, входящие в скрытый узел, умножаются на **веса**, набор предварительно определенных чисел, загруженных в программу. Затем взвешенные входы суммируются, для получения единственного числа. Это показано на схеме знаком ∑. Прежде чем покинуть узел, это число проходит через нелинейную математическую функцию, называемую *сигмоидой*. Это “S” образная кривая, ограничивающая выходной сигнал узла. То есть на вход сигмоиды может быть подано значение от -∞ до +∞, в то время как выходное значение может быть только между нулем и единицей.

Выходные сигналы от скрытого слоя представлены на структурной схеме (рис. 26.5) переменными $X2_1$, $X2_2$, $X2_3$ и $X2_4$. Так же, как и прежде, каждое из этих значений копируется и направляется к следующему слою. Активные узлы выходного слоя объединяют и изменяют данные для выдачи двух выходных значений этой сети $X3_1$ и $X3_2$.

Нейронные сети могут иметь любое количество слоев и любое количество узлов на каждом слое. Большинство приложений используют трехслойную структуру с

максимальным количеством входных узлов около нескольких сотен. Размер скрытого слоя обычно около 10 % размера входного слоя. В случае обнаружения цели, для выходного слоя нужен только один узел. Выход этого узла пороговый, чтобы обеспечить положительный или отрицательный признак присутствия или отсутствия цели во входных данных.

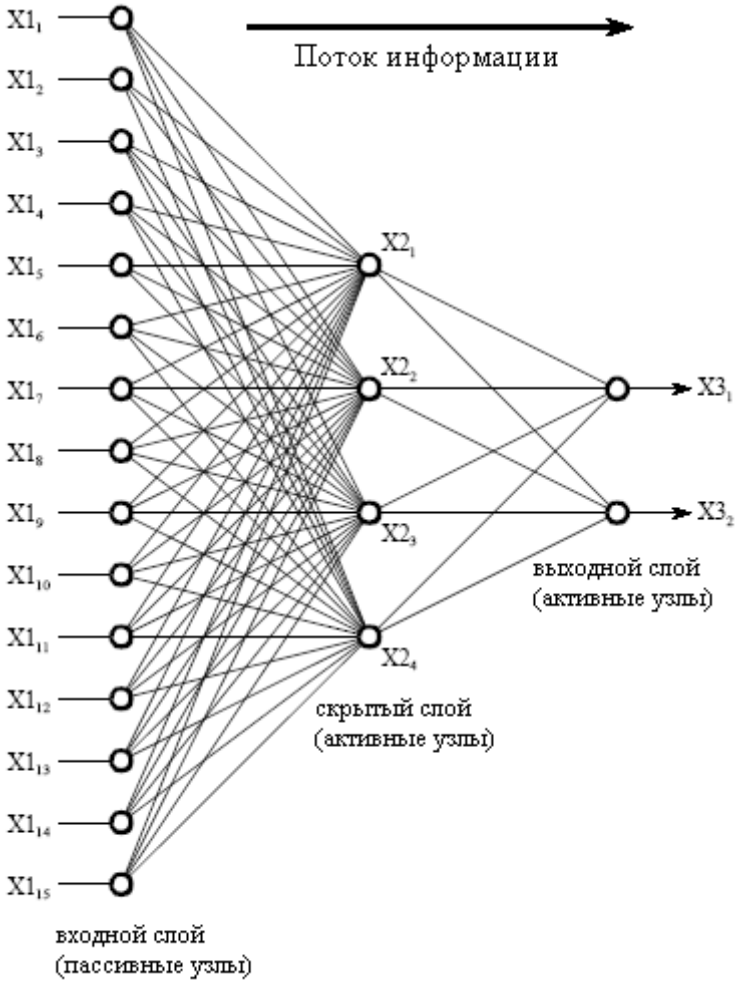


Рис. 26.5 Архитектура нейронной сети

В таблице 26.1 приведена программа, реализующая структурную схему на рис. 26.5. Ключевым пунктом является то, что эта архитектура очень простая и очень обобщенная. Та же самая структурная схема может использоваться для решения многих задач независимо от их особенностей. Способность нейронной сети обеспечивать полезную обработку данных лежит в надлежащем выборе *весов*. Это впечатляющий отход от обычной обработки информации, где решения описываются пошаговыми процедурами.

В качестве примера представьте нейронную сеть для распознавания объектов по звуковому сигналу. Предположим, что в компьютер загружена 1000 отсчетов сигнала. Как компьютер определит, представляют ли эти данные подводную лодку, кита, подводную гору, или вообще ничего? Традиционная ЦОС подошла бы к этой проблеме с математикой и алгоритмами подобными корреляционному анализу и анализу частотного спектра. С нейронной сетью, 1000 отсчетов просто вводятся во входной слой, при этом результирующие значения просто выскакивают из выходного слоя. Выбирая надлежащие веса, выход может быть сконфигурирован так, чтобы выдавать информацию широкого диапазона. Например, могут быть отдельные выходы для: подводной лодки (да/нет), кита (да/нет), подводной горы (да/нет) и т.д. При других весах, выходы могут

классифицировать такие цели как: металл или неметалл, биологический или небиологический, враг или союзник и т.д. Никаких алгоритмов, никаких правил, никаких процедур; только соотношения между входом и выходом, продиктованные отобранными значениями весов.

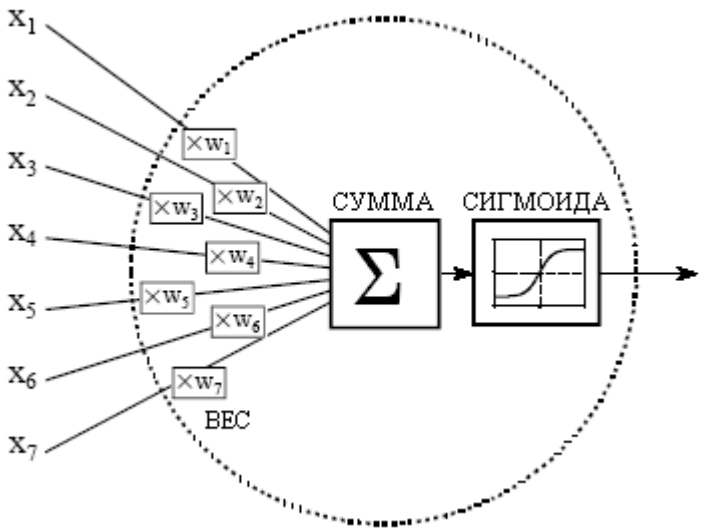


Рис. 26.6 Активный узел нейронной сети

Таблица 26.1

```
100 'NEURAL NETWORK (FOR THE FLOW DIAGRAM IN FIG. 26.5)
110 '
120 DIM X1[15]                    'holds the input values
130 DIM X2[4]                     'holds the values exiting the hidden layer
140 DIM X3[2]                     'holds the values exiting the output layer
150 DIM WH[4,15]                  'holds the hidden layer weights
160 DIM WO[2,4]                   'holds the output layer weights
170 '
180 GOSUB XXXX                    'mythical subroutine to load X1[ ] with the input data
190 GOSUB XXXX                    'mythical subroutine to load the weights, WH[ , ] & WO[ , ]
200 '
210 ' 'FIND THE HIDDEN NODE VALUES, X2[ ]
220 FOR J% = 1 TO 4               'loop for each hidden layer node
230 ACC = 0                       'clear the accumulator variable, ACC
240 FOR I% = 1 TO 15              'weight and sum each input node
250 ACC = ACC + X1[I%] * WH[J%,I%]
260 NEXT I%
270 X2[J%] = 1 / (1 + EXP(-ACC) ) 'pass summed value through the sigmoid
280 NEXT J%
290 '
300 ' 'FIND THE OUTPUT NODE VALUES, X3[ ]
310 FOR J% = 1 TO 2               'loop for each output layer node
320 ACC = 0                       'clear the accumulator variable, ACC
330 FOR I% = 1 TO 4               'weight and sum each hidden node
340 ACC = ACC + X2[I%] * WO[J%,I%]
350 NEXT I%
360 X3[J%] = 1 / (1 + EXP(-ACC) ) 'pass summed value through the sigmoid
370 NEXT J%
```

```
380 '
390 END
```

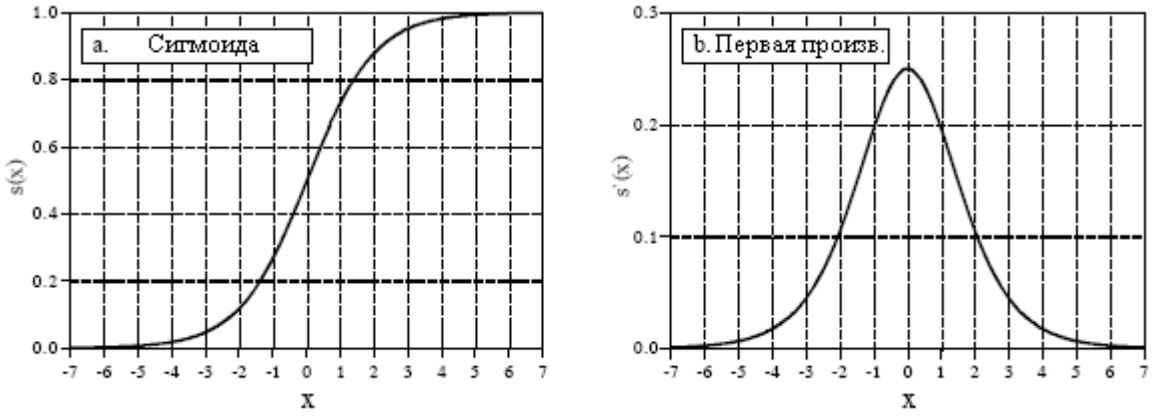


Рис. 26.7 Сигмоидальная функция и ее производная

Рис.26.7 более подробно представляет сигмоиду, математически описываемую уравнением:

$$s(x) = \frac{1}{1+e^{-x}}. \qquad (26.1)$$

Точная форма сигмоиды не важна, важно только то, что это **плавный порог**. Сравните, **простой порог** дает значение *единицы,* когда $x>0$ и значение *нуля,* когда $x<0$. Сигмоида выполняет ту же самую базовую пороговую функцию, но, как это показано на рис. 26.7, она еще и *дифференцируема*. Поскольку в структурной схеме (рис.25.5) производная не используется, нахождение должных весов для применения является критической частью. Поподробнее об этом, но вкратце. Преимущество сигмоиды в том, что есть краткий путь вычисления значения ее производной:

$$s'(x) = s(x)\left[1 - s(x)\right]. \qquad (26.2)$$

Например, если $x = 0$, то $s(x) = 0{,}5$ (в соответствии с уравнением 26.2) и вычисляется первая производная: $s'(x) = 0.5 \times (1 - 0.5) = 0{,}25$. Это не критическая концепция, а только ловкий прием, чтобы сделать алгебру покороче.

Не была бы нейронная сеть более гибкой, если бы сигмоида могла регулироваться влево или вправо и иметь центр на каком-то другом значении, чем $x = 0$? Ответом будет – да, и большинство нейронных сетей позволяют это делать. Это достаточно просто осуществить; во входной слой добавляется дополнительный узел, причем на его входе всегда имеется значение *единицы*. Когда все это умножается на веса скрытого слоя, то обеспечивается *сдвиг* (смещение по постоянному току) каждой сигмоиды. Это добавление называется **узлом сдвига**. С ним обращаются таким же образом, как и с другими узлами, кроме константы на входе.

Могут ли быть сделаны нейронные сети без сигмоиды или подобной нелинейности? Для ответа на этот вопрос посмотрите на трехслойную сеть на рис. 26.5. Если бы сигмоиды не присутствовали, три слоя *коллапсировали* бы в два слоя. Другими словами, суммирование и взвешивание скрытых и выходных слоев могут быть объединены в один слой, в результате получается только двухслойная сеть.

Почему это работает?

Веса, необходимые, для того чтобы сделать нейронную сеть способной выполнять специфическую задачу, находятся с помощью **обучающего алгоритма**, используемого совместно с **тестовыми примерами** того, как система *должна* работать. Так тестовым примером в задаче со звуковым сигналом была бы база данных в несколько сотен (или больше) сегментов по 1000 отсчетов каждый. Некоторые из сегментов тестового примера соответствовали бы субмаринам, другие китам, третьи случайному шуму и т.д. Обучающий алгоритм использует тестовые примеры для вычисления набора весов соответствующего решаемой задаче. Термин **обучающий** широко используется в области нейронных сетей для описания этого процесса; однако, более лучшим описанием могло бы быть: *определение*, *основанное на статистике примеров оптимизированного набора весов*. Независимо от того, как называется этот метод, результирующие веса фактически недоступны для человеческого понимания. В некоторых редких случаях могут быть обозримы отдельные образцы, но, в общем, они кажутся случайными числами. Можно наблюдать, что нейронная сеть, использующая эти веса, имеет хорошее соотношение вход/выход, но *почему* именно эти конкретные веса работают, остается полной загадкой. Это мистическое свойство нейронных сетей стало причиной того, что многие ученые и инженеры стали избегать их. Помните все эти научно-фантастические фильмы о компьютерах-предателях, захватывающих землю?

Несмотря на это, часто слышишь, как адвокаты нейронных сетей делают заявления подобные следующему: "нейронные сети хорошо изучены". Для изучения этого заявления, мы сначала покажем, что возможно выбрать веса нейронной сети посредством традиционных методов ЦОС. Затем, мы продемонстрируем, что обучающие алгоритмы обеспечивают *лучшие* решения, чем традиционные методы. Хотя это и не объясняет, *почему* конкретный набор весов работает, это обеспечивает методу доверие.

С наиболее утонченной точки зрения нейронная сеть это способ обозначения различных областей в *параметрическом пространстве*. Например, рассмотрим нейронную сеть звуковой системы, имеющую 1000 входов и один выход. При соответствующем выборе весов, выходной сигнал будет близок к *единице*, в случае если входным сигналом будет эхо от подводной лодки и близок к *нулю*, если на входе только шум. Это образует параметрическое гиперпространство из 1000 измерений. Нейронная сеть это метод присваивания значения каждому месту в этом гиперпространстве. То есть, 1000 входных значений определяют *место* в гиперпространстве, в то время как выход нейронной сети дает *значение* в этом месте. Эту задачу могла бы отлично выполнять таблица поиска, имеющая загруженное для каждого возможного входного адреса выходное значение. Разница в том, что нейронная сеть *вычисляет* значение в каждом месте (адресе), это предпочтительнее, чем невероятно огромная задача *хранения* каждого значения. В действительности архитектура нейронной сети часто оценивается тем, как хорошо она разделяет гиперпространство для заданного количества весов.

Этот подход дает также ключ к пониманию необходимого количества узлов в скрытом слое. В N мерном параметрическом пространстве для обозначения местоположения требуется N чисел. Идентификация *области* в гиперпространстве требует $2 \times N$ значений (то есть минимальное и максимальное значения на каждой оси определяют гиперпространственное прямоугольное геометрическое тело). Например, эти простые вычисления показали бы, что нейронной сети с 1000 входов для отделения одной области гиперпространства от другой требуется 2000 весовых коэффициентов. В полностью взаимосвязанной сети, это потребовало бы двух скрытых узлов. Число требуемых областей зависит от конкретной проблемы, но можно утверждать, что это число гораздо меньше числа измерений в параметрическом пространстве. Несмотря на то, что это только

грубое приближение, оно, в общем, объясняет, почему большинство нейронных сетей могут работать со скрытым слоем размером от 2% до 30% от размера входного слоя.

Совершенно другой способ понимания нейронных сетей использует концепцию *корреляции* из ЦОС. Как обсуждалось в Главе 7, корреляция оптимальный способ обнаружения, при условии, что внутри сигнала содержится известный образец. Она выполняется путем умножения отсчетов сигнала на копию искомого образца и сложения результатов произведения. Чем больше значение суммы, тем больше сигнал походит на образец. Теперь, рассмотрите рис. 26.5 и думайте о каждом скрытом узле как об узле, ищущем определенный образец во входных данных. То есть, каждый из скрытых узлов *коррелирует* входные данные с набором весов, связанных с этим скрытым узлом. Если образец присутствует, то сумма, переданная в сигмоиду, будет большая, в противном случае маленькая.

Действие сигмоиды с этой точки зрения довольно интересно. Вернемся к рис. 26.1d и отметим, что кривая вероятности, разделяющая два хорошо очерченных колоколообразных распределения, похожа на сигмоиду. Если бы мы вручную изобретали нейронную сеть, мы могли бы сделать так, что бы на выходе каждого скрытого узла выдавалась *дробная вероятность* того, что определенный образец присутствует во входных данных. Выходной слой повторяет эту операцию, делая всю трехслойную структуру корреляцией корреляций, сетью, которая ищет *образцы образцов*.

Традиционная ЦОС основана на двух методах *свертке* и *анализе Фурье*. Интересно, что нейронные сети могут выполнять обе эти операции и даже *гораздо больше*. Представьте, что сигнал, состоящий из N отсчетов, фильтруется с целью получения другого сигнала так же содержащего N отсчетов. В соответствии с взглядом на свертку со стороны выхода, каждый отсчет в выходном сигнале является взвешенной суммой входных отсчетов. Теперь представьте двухслойную нейронную сеть с N узлами в каждом слое. Значение, сформированное каждым узлом выходного уровня, также являются взвешенной суммой входных значений. Если каждый узел выходного уровня, как и все другие выходные узлы, использует те же самые веса, сеть будет обеспечивать линейную свертку. Аналогичным образом, с помощью двухслойной нейронной сети с N узлами в каждом слое может быть вычислено ДПФ. Каждый узел выходного слоя находит амплитуду составляющей для одной единственной частоты. Это выполняется путем изменения весов каждого узла выходного слоя в соответствии с разыскиваемой синусоидой. Полученная сеть коррелирует входной сигнал с каждой базовой функцией синусоид, вычисляя, таким образом, ДПФ. Конечно, двухслойная сеть значительно менее мощная, чем обычная трехслойная архитектура. Это означает, что нейронные сети могут выполнять *нелинейные* операции также хорошо, как и *линейные*.

Предположим, что одна из этих стратегий традиционной ЦОС используется для поиска весов нейронной сети. Можно ли заявить, что сеть является *оптимальной*? Алгоритмы традиционной ЦОС обычно основаны на предположениях относительно характеристик входного сигнала. Например, фильтрация Винера оптимальна для максимизации отношения сигнал/шум, *допуская*, что известны спектры и сигнала, и шума; корреляция оптимальна для обнаружения целей, *допуская*, что шум является белым; обратная свертка противодействует нежелательной свертке, *допуская*, что ядро обратной свертки является инверсией исходного ядра свертки и т.д. Проблема заключается в том, что ученые и инженеры редко имеют четкое представление о входных сигналах, с которыми им предстоит столкнуться. В то время как лежащая в основе математика может быть элегантной, результирующие характеристики ограничены тем, насколько хорошо поняты данные.

Например, представьте проверку алгоритма традиционной ЦОС с использованием фактических входных сигналов. Затем, повторите проверку, слегка изменив алгоритм, скажем, увеличив один из параметров на один процент. Если результат второй проверки лучше, чем первой, исходный алгоритм не является оптимальным для данной задачи.

Почти все алгоритмы традиционной ЦОС могут быть значительно улучшены с помощью оценки методом проб и ошибок, небольших изменений в параметрах и в процедурах алгоритма. Это и есть стратегия нейронной сети.

Обучение нейронной сети

Конструкция нейронной сети лучше всего может быть объяснена на примере. На рис. 26.8 показана проблема, которую мы собираемся атаковать, идентификация индивидуальных букв в изображении текста. Этой задаче распознавания изображений уделялось большое внимание. Она достаточно легка, так что большое количество подходов достигли частичного успеха, но и достаточно сложна настолько, что до сих пор нет ни одного точного решения. На этой проблеме базировалось большое количество коммерческих продуктов такие как: чтение адресов на письмах почтовых отправлений, ввод документов в текстовые редакторы и т.д.

Первый шаг в развитии нейронной сети создать базу данных примеров. Для проблемы распознавания текста это осуществляется распечатыванием 50 раз на листе бумаги 26 заглавных букв латинского алфавита: A, B, C, D, …, Y, Z. Затем эти 1300 букв преобразуются в цифровое изображение с помощью одного из большого числа доступных для персональных компьютеров сканирующих устройств. Это большое цифровое изображение делится затем на десять маленьких изображений размером 10×10 пикселей, каждое из которых содержит одну букву. Эта информация сохраняется в виде базы данных размером 1,3 Мегабайтов: 1300 изображений; 100 пикселей в изображении; 8 битов на один пиксель. Мы будем использовать первые 260 изображений в этой базе данных для *обучения* нейронной сети (т. е. определения весов), а оставшиеся для *тестирования* ее функционирования. База данных должна также включать способ идентификации символа, содержащегося в каждом изображении. Например, к каждому изображению10×10 может быть добавлен дополнительный байт, содержащий ASCII код символа. В другой схеме, показывать, что это за буква может позиция каждого изображения 10×10 в базе данных. Например, изображения от 0 до 49 могут все быть "A", изображения от 50 до 99 могут все быть "B" и т.д.

Рис 26.8 Пример изображения текста

Для этой демонстрации, нейронная сеть будет разработана для произвольно выбранной задачи: определить, какое из изображений 10×10 содержит *гласные*, т. е. A, E, I, O, или U. Может быть, она и не имеет никакого практического применения, зато она иллюстрирует способности нейронной сети изучать проблемы распознавания очень абстрактных образов. Включая десять примеров каждой буквы в обучающий набор, сеть будет (мы надеемся) изучать ключевые признаки, которые отличают образы цели от не цели.

Нейронная сеть, используемая в этом примере, является сетью с традиционной трехслойной полностью взаимосвязанной архитектурой, показанной на рис. 26.5 и 26.6. Имеется 101 узел во входном слое (100 значений пикселей плюс узел сдвига), 10 узлов в скрытом слое, и 1 узел в выходном слое. Когда изображение из 100 пикселей подается на вход сети, мы хотим, чтобы выходное значение было близко к *единице*, если гласная присутствует, и около *нуля*, если гласной нет. Не беспокойтесь, что входной сигнал был получен в виде двумерного массива (10×10), в то время как вход в нейронную сеть является одномерным массивом. Это *Ваше* понимание того, как взаимосвязаны значения пикселей; *нейронная сеть* найдет свои собственные взаимосвязи.

Таблица 26.2 вместе с таблицей 26.3, содержащей три подпрограммы, вызываемых из основной программы, показывает основную программу для вычисления весов нейронной сети. Элементы массива от X1[1] до X1[100] содержат значения входного слоя. Кроме того, X1[101] всегда содержит значение 1, обеспечивая вход в узел сдвига. Выходные величины, поступающие от скрытых узлов, содержатся в элементах массива от X2[1] до X2[10]. Переменная X3 содержит выходное значение сети. Веса скрытого слоя содержатся в массиве WH[,], где первый индекс идентифицирует скрытые узлы (с 1 по 10), а второй индекс является узлом второго слоя (с 1 по 101). Веса выходного слоя находятся с WO[1] по WO[10]. Это составляет всего 1020 весовых значений, определяющих, как будет работать сеть.

Первое действие программы это установить каждый вес в произвольное начальное значение с помощью использования генератора случайных чисел. Как показано в строках, начиная со 190 по 240, весам скрытого слоя приписываются первоначальные значения между -0,0005 и 0,0005, в то время как веса выходного слоя находятся между –0,5 и 0,5. Эти диапазоны по величине выбраны такого же порядка, какого *должны* быть окончательные веса. Выбор основан на: (1) диапазоне значений во входном сигнале, (2) количестве входов суммируемых в каждом узле и (3) диапазоне значений, на протяжении которых сигмоида является активной, значения на входе приблизительно-5 < x < 5, а на выходе от 0 до 1. Например, когда 101 вход с типичным значением 100 умножается на типичное значение веса 0,0002, сумма произведений приблизительно равна 2, что находится в активном диапазоне входа в сигмоиду.

Если бы мы оценили работу нейронной сети, использующей эти случайные веса, то мы бы обнаружили, что это является тем же самым, что и случайное угадывание. Обучающий алгоритм улучшает работу сети путем постепенного изменения величины каждого веса в нужном направлении. Это называется **итеративной** процедурой и управляется в программе циклом FOR-NEXT в строках 270-400. Каждая итерация делает веса чуть-чуть более эффективными при отделении целевых примеров от не целевых. Итерационный цикл обычно выполняется до тех пор, пока не перестанут происходить никакие дальнейшие улучшения. В типичных нейронных сетях, это может быть где-то от десяти до десяти тысяч итераций, но обычно происходит всего несколько сотен. В этом примере выполняется 800 итераций.

Для того чтобы эта итеративная стратегия работала, должен быть *единый* параметр, описывающий, насколько хорошо функционирует эта система в настоящее время. В программе эту функцию выполняет переменная ESUM (для суммарной ошибки). Первое действие внутри итерационного цикла - обнулить переменную ESUM (строка 290), теперь она может использоваться, как аккумулятор. В конце каждой итерации значение ESUM

выводится на видео экран (строка 380), таким образом, оператор может быть уверен, что процесс прогрессирует. В начале значение ESUM большое и постепенно уменьшается по мере того, как нейронная сеть обучается узнавать цели. На рис. 26.9 приведены примеры того, как уменьшается ESUM в процессе итерации.

Таблица 26.2

```
100 'NEURAL NETWORK TRAINING (Determination of weights)
110 '
120                                 'INITIALIZE
130 MU = .000005                    'iteration step size
140 DIM X1[101]                     'holds the input layer signal + bias term
150 DIM X2[10]                      'holds the hidden layer signal
160 DIM WH[10,101]                  'holds hidden layer weights
170 DIM WO[10]                      'holds output layer weights
180 '
190 FOR H% = 1 TO 10                'SET WEIGHTS TO RANDOM VALUES
200 WO[H%] = (RND-0.5)              'output layer weights: -0.5 to 0.5
210 FOR I% = 1 TO 101               'hidden layer weights: -0.0005 to 0.0005
220 WH[H%,I%] = (RND-0.5)/1000
230 NEXT I%
240 NEXT H%
250 '
260                                 'ITERATION LOOP
270 FOR ITER% = 1 TO 800            'loop for 800 iterations
280 '
290 ESUM = 0                        'clear the error accumulator, ESUM
300 '
310 FOR LETTER% = 1 TO 260          'loop for each letter in the training set
320 GOSUB 1000                      'load X1[ ] with training set
330 GOSUB 2000                      'find the error for this letter, ELET
340 ESUM = ESUM + ELET^2            'accumulate error for this iteration
350 GOSUB 3000                      'find the new weights
360 NEXT LETTER%
370 '
380 PRINT ITER% ESUM                'print the progress to the video screen
390 '
400 NEXT ITER%
410 '
420 GOSUB XXXX                      'mythical subroutine to save the weights
430 END
```

Все 260 изображений в обучающем наборе оцениваются в течение каждой итерации под управлением цикла FOR-NEXT в строках 310-360. Подпрограмма 1000 используется, для восстановления изображений из базы данных примеров. Так как здесь это не представляет особого интереса, мы опишем только параметры, поступающие в эту подпрограмму и выходящие из нее. Подпрограмма 1000 вызывается с параметром LETTER%, находящимся между 1 и 260. После возврата из подпрограммы, значения входных узлов с X1[1] по X1[100] содержат величины пикселей, принадлежащих изображению в базе данных с номером, находящемся в переменной LETTER%. Значение узла сдвига X1[101] всегда возвращается в виде постоянной величины, равной *единице*. Подпрограмма 1000 возвращает также другой параметр CORRECT. Он содержит

желаемое выходное значение сети для этой конкретной буквы. То есть если буква в изображении является гласной, CORRECT будет возвращен с *единичным* значением. Если буква в изображении не является гласной, CORRECT будет возвращен с *нулевым* значением.

После того, как обрабатываемое изображение загружено с X1[1] по X1[100], подпрограмма 2000 пропускает данные через текущую конфигурацию нейронной сети, чтобы получить значение выходного узла X3. Другими словами, подпрограмма 2000 такая же, как программа, показанная в таблице 26.1 за исключением другого количества узлов в каждом слое. Эта подпрограмма вычисляет также, насколько хорошо текущая сеть определяет букву, как цель или не цель. В строке 2210, как разница между фактически сгенерированным выходным значением X3 и желаемым значением CORRECT, вычисляется переменная ELET (ошибка буквы). В результате этого ELET принимает значения между –1 и 1. Все 260 значений для ELET собираются (строка 340) для формирования ESUM, квадрат общей ошибки сети для всего обучающего набора.

Строка 2220 показывает выбор, который часто включают в программы при вычислении ошибки: присвоение разной *важности* ошибкам для целей и не целей. В частности, вспомните случай рака, рассмотренный ранее в этой главе и последствия совершения ложноположительной ошибки по сравнению с ложноотрицательной ошибкой. В настоящем примере, мы произвольно заявили, что ошибка в обнаружении цели в пять раз хуже, чем ошибка в обнаружении не цели. В действительности, это заставляет сеть лучше работать с целями, даже если это нарушает работу с не целями.

Подпрограмма 3000 - это сердце стратегии нейронной сети, алгоритм изменения весов на каждой итерации. Мы будем использовать аналогии, чтобы объяснять лежащую в основании математику. Рассмотрим затруднительное положение военного парашютиста, заброшенного в тыл врага. Он приземляется на незнакомой вражеской территории и обнаруживает, что вокруг так темно, что он ничего не может видеть дальше, чем на несколько шагов. Его приказ спуститься в низину ближайшей долины, откуда начать выполнение оставшейся части его миссии. Проблема в том, как он сможет проделать свой путь в низину долины, без возможности видеть больше, чем на несколько шагов? Для прокладывания другой дороги, ему нужен алгоритм, чтобы выверять свое положение на поверхности земли по *x* и по *y*, для того чтобы *минимизировать* свое возвышение. Это аналогично проблеме выверки весов нейронной сети таким образом, чтобы ошибка сети ESUM минимизировалась.

Мы рассмотрим два алгоритма, предназначенных для решения этой проблемы: **эволюции** и **крутого спуска**. В методе эволюции, с места приземления, парашютист прыгает в некотором случайном направлении. Если его новое положение *выше*, чем предыдущее, он проклинает все на свете, возвращается к месту, откуда он стартовал и делает следующую попытку. Если его новое положение *ниже*, он ощущает некоторую удачу и повторяет процесс с нового местоположения. В конечном счете, он достигнет низины долины, хотя и по очень неэффективной и случайной траектории. Этот метод называется *эволюцией*, потому что он представляет собой такой же тип алгоритма, который использует природа в биологической эволюции. Каждое новое поколение особей имеет случайные изменения от предыдущего. Если эти различия идут особям на пользу, они вероятнее всего, будут сохранены и переданы *следующему* поколению. Это является результатом улучшения, позволяющего животному получить больше пищи, избегать своих врагов, произвести большее количество потомков и т.д. Если новая черта вредна, то неудачное животное становится завтраком для какого-нибудь хищника, и изменение отбрасывается. В этом смысле, каждое новое поколение это итерация эволюционной процедуры оптимизации.

Когда эволюция используется, как обучающий алгоритм, каждый вес в нейронной сети слегка изменяется прибавлением значения от генератора случайных чисел. Если

модифицированные веса делают сеть лучше (т.е. более низкое значение для ESUM), изменения сохраняются, в противном случае они отбрасываются. В процессе работы этот

Таблица 26.3

```
1000 'SUBROUTINE TO LOAD X1[ ] WITH IMAGES FROM THE DATABASE
1010 'Variables entering routine: LETTER%
1020 'Variables exiting routine: X1[1] to X1[100], X1[101] = 1, CORRECT
1030 '
1040 'The variable, LETTER%, between 1 and 260, indicates which image in the database is to be
1050 'returned in X1[1] to X1[100]. The bias node, X1[101], always has a value of one. The variable,
1060 'CORRECT, has a value of one if the image being returned is a vowel, and zero otherwise.
1070 '(The details of this subroutine are unimportant, and not listed here).
1900 RETURN
2000 'SUBROUTINE TO CALCULATE THE ERROR WITH THE CURRENT WEIGHTS
2010 'Variables entering routine: X1[ ], X2[ ], WI[ , ], WH[ ], CORRECT
2020 'Variables exiting routine: ELET
2030 '
2040 '                                          'FIND THE HIDDEN NODE VALUES, X2[ ]
2050 FOR H% = 1 TO 10                          'loop for each hidden nodes
2060 ACC = 0                                    'clear the accumulator
2070 FOR I% = 1 TO 101                         'weight and sum each input node
2080 ACC = ACC + X1[I%] * WH[H%,I%]
2090 NEXT I%
2100 X2[H%] = 1 / (1 + EXP(-ACC) )             'pass summed value through sigmoid
2110 NEXT H%
2120 '
2130 '                                          'FIND THE OUTPUT VALUE: X3
2140 ACC = 0                                   'clear the accumulator
2150 FOR H% = 1 TO 10                          'weight and sum each hidden node
2160 ACC = ACC + X2[H%] * WO[H%]
2170 NEXT H%
2180 X3 = 1 / (1 + EXP(-ACC) )                 'pass summed value through sigmoid
2190 '
2200 '                                          'FIND ERROR FOR THIS LETTER, ELET
2210 ELET = (CORRECT - X3)                     'find the error
2220 IF CORRECT = 1 THEN ELET = ELET*5          'give extra weight to targets
2230 '
2240 RETURN
3000 'SUBROUTINE TO FIND NEW WEIGHTS
3010 'Variables entering routine: X1[ ], X2[ ], X3, WI[ , ], WH[ ], ELET, MU
3020 'Variables exiting routine: WI[ , ], WH[ ]
3030 '
3040 '                                          'FIND NEW WEIGHTS FOR HIDDEN LAYER
3050 FOR H% = 1 TO 10
3060 FOR I% = 1 TO 101
3070 SLOPEO = X3 * (1 - X3)
3080 SLOPEH = X2(H%) * (1 - X2[H%])
3090 DX3DW = X1[I%] * SLOPEH * WO[H%] * SLOPEO
3100 WH[H%,I%] = WH[H%,I%] + DX3DW * ELET * MU
3110 NEXT I%
3120 NEXT H%
3130 '
3140 '                                          'FIND NEW WEIGHTS FOR OUTPUT LAYER
3150 FOR H% = 1 TO 10
3160 SLOPEO = X3 * (1 - X3)
3170 DX3DW = X2[H%] * SLOPEO
3180 WO[H%] = WO[H%] + DX3DW * ELET * MU
3190 NEXT H%
3200 '
3210 RETURN
```

алгоритм **сходится** медленно. Это жаргон, используемый для описания непрерывного улучшения совершаемого по направлению к оптимальному решению (низина долины). В более простых выражениях, для достижения решения программе потребуются дни вместо часов и минут.

К счастью алгоритм *крутого спуска* значительно быстрее. Вот каким был бы естественный ответ парашютиста: оцените, какой путь ведет *вниз*, и двигайтесь в этом направлении. Подумаем над ситуацией в этом направлении. Парашютист может сделать один шаг на север и записать изменения в возвышении. После возвращения к своему первоначальному положению, он может сделать один шаг на восток и снова записать изменения в возвышении. Используя эти два значения, он может определить, какое направление ведет вниз. Предположим, что парашютист снижается на 10 сантиметров, когда делает один шаг в северном направлении и снижается на 20 сантиметров, когда делает один шаг в восточном направлении. Чтобы перемещаться прямо вниз, он должен двигаться вдоль каждой оси на величину пропорциональную наклону вдоль этой оси. В таком случае он может переместиться на 10 шагов на север и на 20 шагов на восток. Это перенесет его вниз по самой крутой части склона на расстояние $\sqrt{10^2 + 20^2} = 22{,}4$ шага. Он мог бы двигаться к новому месту и по-другому, 22,4 шага по прямой линии, по диагонали. Ключевым пунктом является то, что *самый крутой спуск достигается при перемещении по каждой оси на расстояние пропорциональное наклону вдоль этой оси*.

Этот самый алгоритм крутого спуска для нахождения требуемых весов сети выполняет подпрограмма 3000. До входа в подпрограмму 3000 на входной слой был подан один из примеров изображений, и информация распространилась к выходу. Это означает, что значения для X1[ ], X2[ ] и X3 все точно определены, как и текущие значения весов WH[,] и WO[ ]. Кроме того, мы знаем ошибку ELET, которую дает сеть для этого конкретного изображения. В строках от 3050 до 3120 происходит обновление весов скрытого слоя, в то время как веса выходного слоя модифицируются в строках от 3150 до 3190. *Это выполняется за счет вычисления наклона для каждого из весов и изменения*, *затем*, *каждого из весов на соответствующую величину*, *пропорциональную этому наклону*. В случае с парашютистом, наклон вдоль оси находится за счет перемещения по оси на небольшое расстояние (скажем Δx), измерения изменения, произошедшего в возвышении (скажем ΔE), а затем деления их друг на друга ($\Delta E/\Delta x$). Наклон веса нейронной сети может быть найден тем же самым способом: прибавьте небольшое приращение к значению веса (Δw), найдите результирующее изменение в выходном сигнале (ΔX3), и поделите их друг на друга ($\Delta X3/\Delta w$). Позже в этой главе мы рассмотрим пример, который вычисляет наклон этим способом. Однако в настоящем примере мы воспользуемся более эффективным методом.

Ранее мы говорили, что нелинейность (сигмоида) должна обладать свойством дифференцируемости. Именно здесь мы и будем использовать это свойство. Если мы знаем наклон в каждой точке нелинейности, мы можем прямо написать уравнение для наклона каждого веса ($\Delta X3/\Delta w$), без фактической необходимости изменять его. Рассмотрим определенный вес, например WO[1], соответствующий первому входу выходного узла. Взгляните на структуру, на рис. 26.5 и 26.6, и задайте себе вопрос: как будет воздействовать на выход (X3), легкое изменение конкретного веса (w), когда все остальные веса сохраняются теми же самыми? Ответом будет:

$$\frac{\Delta X3}{\Delta w} = X2[1] SLOPE_O, \qquad (26.3)$$

где $SLOPE_O$ - первая производная от сигмоиды выходного слоя, оценивающая, где мы работаем на этой кривой. Другими словами, $SLOPE_O$ описывает на сколько изменяется величина на *выходе* сигмоиды в ответ на изменения на ее *входе*. В соответствии с

уравнением (26.2), $SLOPE_O$ может быть вычислена из текущего выходного значения сигмоиды X3. Это вычисление показано в строке 3160. В строке 3170 наклон для этого веса вычислен по уравнению (26.3) и хранится в переменной DX3DW (т.е. $\Delta X3/\Delta w$).

Используя подобный анализ, можно найти наклон для веса в скрытом слое, такого как WH[1,1]:

$$\frac{\Delta X3}{\Delta w} = X1[1]SLOPE_{H1}WO[1]SLOPE_O, \qquad (26.4)$$

где $SLOPE_{H1}$ - первая производная от сигмоиды скрытого слоя, оценивающая, где мы работаем на этой кривой. Другие значения X1[1] и WO[1], просто константы, которые изменение веса встречает на своем пути к выходу. В строках 3070 и 3080 наклон сигмоид вычисляется с помощью уравнения 26.2. Наклон веса скрытого слоя DX3DW вычисляется в строке 3090 через уравнение (26.4)

Теперь, когда мы знаем *наклон* каждого из весов, мы можем видеть, как для следующей итерации изменяется каждый вес. Новое значение для каждого веса находится суммированием текущего веса с некоторой величиной пропорциональной наклону:

$$w_{new} = w_{old} + \frac{\Delta X3}{\Delta w} ELET \quad MU. \qquad (26.5)$$

Это вычисление производится в строке 3100 для скрытого слоя и в строке 3180 для выходного слоя. Пропорциональная постоянная состоит из двух сомножителей, ELET - ошибка сети для данного конкретного входа и MU - постоянная устанавливаемая в начале программы. Для того чтобы понять, зачем в этом вычислении нужна ELET, представьте, что изображение, помещенное на вход, в выходном сигнале дает *небольшую* ошибку. Затем представьте, что другое изображение, поданное на вход, дает *большую* ошибку на выходе. При настройке весов, в случае второго изображения мы хотим дать сети более сильный толчок, чем в случае первого. Если что-то работает плохо, мы хотим это изменить, если работает хорошо, мы хотим все оставить, как есть. Это осуществляется, изменением каждого веса пропорционально текущей ошибке ELET.

Чтобы понять, как воздействует на систему MU, вспомним пример с парашютистом. Как только он определяет направление вниз, он должен решить, как далеко ему пройти, прежде чем сделать повторную оценку наклона ландшафта. Делая эту дистанцию короткой, например в один метр, он будет способен точно следовать за контурами ландшафта и всегда перемещаться в оптимальном направлении. Проблема состоит в том, что он тратит большую часть своего времени, оценивая наклон, а не двигаясь вниз по холму. Для сравнения, он мог бы выбрать дистанцию побольше, скажем 1000 метров. В то время как это позволило бы парашютисту быстро продвигаться по ландшафту, он может промахиваться мимо пути ведущего вниз. Слишком большая дистанция заставляет его прыгать по всей местности, не совершая желаемого прогресса.

В нейронной сети MU управляет величиной изменения весов на каждой итерации. Используемое значение зависит от специфики решаемой задачи и может быть столь же низким как 10^{-6}, или столь же высоким как 0,1. По аналогии с парашютистом, можно ожидать, что слишком малое значение станет причиной того, что сеть будет сходиться слишком медленно. Для сравнения, слишком большое значение станет причиной неустойчивой сходимости и будет демонстрировать хаотическое колебание вокруг конечного решения. К сожалению, способ, которым нейронные сети реагируют на различные значения MU, может оказаться трудным для понимания или предсказания. Поэтому крайне необходимо, чтобы ошибка сети (т.е. ESUM) наблюдалась во время обучения, например, выводилась на видео экран в конце каждой итерации. Если система

не сходится должным образом, остановите программу и пробуйте другое значение для MU.

Оценка результатов

Так, как это работает? Программа обучения для распознания гласных прогонялась три раза, каждый раз используя для первоначальных весов различные случайные значения. Для того чтобы выполнить 800 итераций, при использовании компьютера Pentium с тактовой частотой 100 МГц, требуется приблизительно один час работы. На рис. 26.9 показано, как за этот период меняется ошибка сети ESUM. Постепенный наклон указывает, что сеть обучается задаче, и что веса достигают близкого к оптимальному значения после нескольких сотен итераций. Каждая попытка дает другое решение проблемы, с другими конечными характеристиками. Это аналогично парашютисту, начинающему с разных местоположений, и поэтому заканчивающему в низинах разных долин. Так же как некоторые долины глубже, чем другие, некоторые решения нейронной сети лучше, чем другие. Это означает, что с лучшим решением, взятым из группы как конечное решение, обучающий алгоритм должен быть запущен несколько раз.

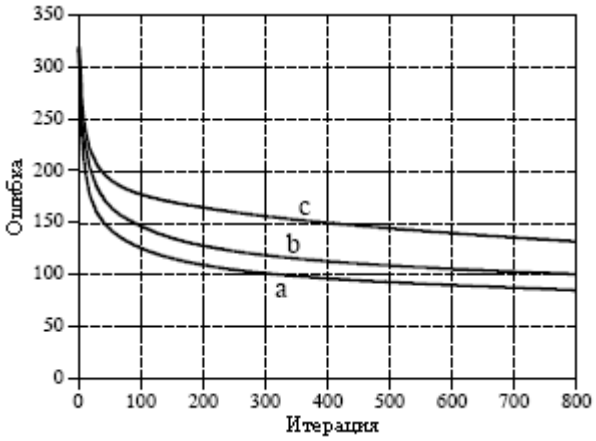


Рис. 26.9 Сходимость нейронной сети

На рис. 26.10 в виде изображения показаны веса скрытого слоя для трех решений. Значения весов результат первого действия, выполненного нейронной сетью, найти корреляцию (умножить и просуммировать) этих изображений с входным сигналом. Они выглядят как случайный шум! Можно *показать*, что значения этих весов работают, но *почему* они работают, остается чем-то из области мистики. Здесь есть над чем подумать. Мозг человека состоит приблизительно из 100 *миллиардов* нейронов, каждый из которых, в среднем, имеет 10000 взаимосвязей. Если мы не можем понять простую нейронную сеть из этого примера, то как же мы можем изучить что-то, что, по крайней мере, в 100000000000000 раз более сложно? Это исследование 21-го века.

На рис. 26.11a показана гистограмма выходов нейронной сети для 260 букв в обучающем наборе. Помните, веса были отобраны так, чтобы на выходе была *единица* для изображений гласных и *ноль* в других случаях. Разделение было идеально достигнуто, без перекрытия между двумя распределениями. Также заметьте, что распределение гласных уже, чем распределение согласных. Это происходит потому, что мы декларировали, что ошибка целевых будет в пять раз более важной, чем ошибка не целевых (см. строку 2220).

Для сравнения, рис. 26.11b показывает гистограмму для изображений в базе данных от 261 до 1300. В то время как целевые и не целевые распределения отчетливо различимы, полностью они не разделены. Почему нейронная сеть лучше функционирует на первых 260 буквах, чем на последних 1040? Рисунок 26.11a обманывает! Легко

выполнить проверку, если Вы уже видели ответы. Другими словами нейронная сеть распознает специфические изображения в обучающем наборе, а не общие модели, опознающие гласные от согласных.

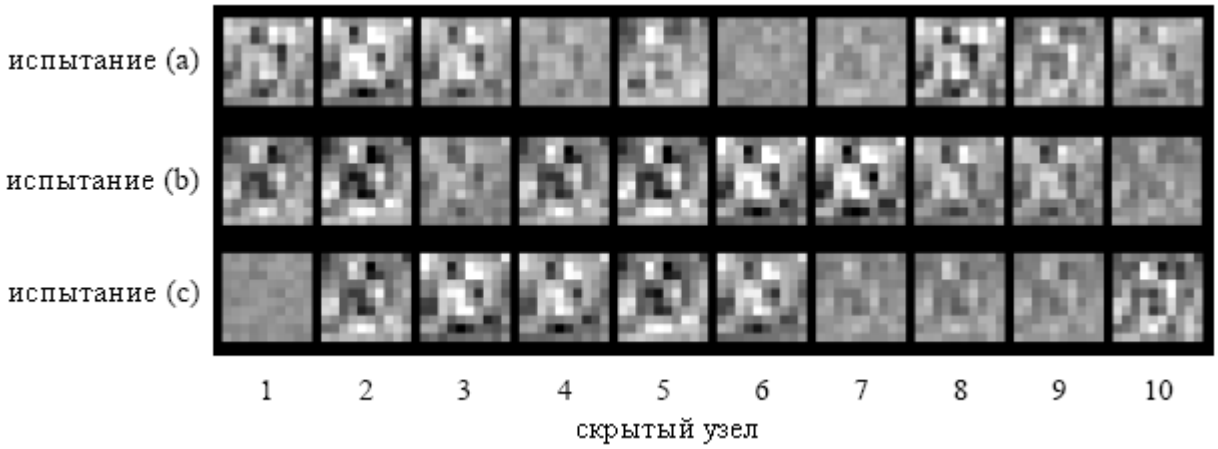


Рис. 26.10 Пример весов нейронной сети

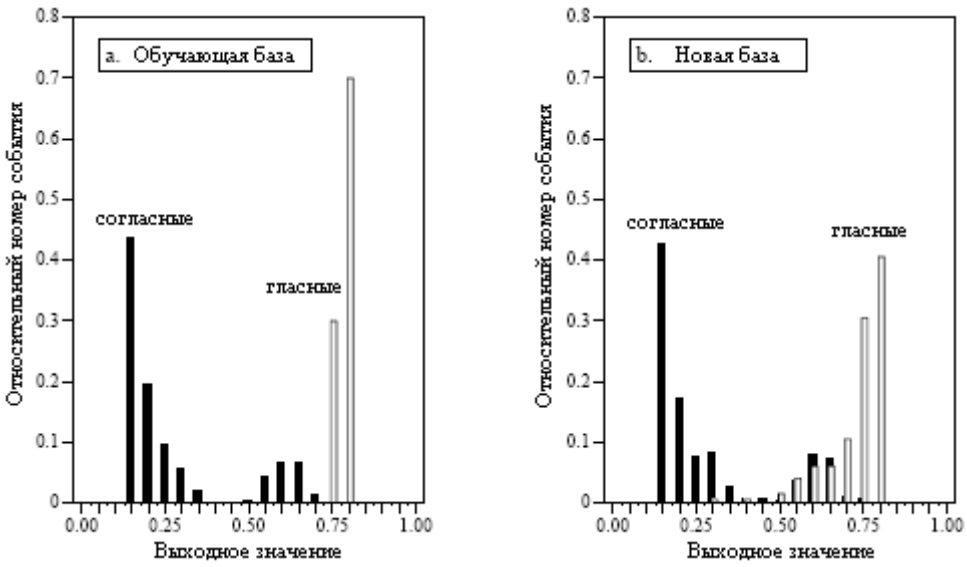


Рис. 26.11 Характеристики нейронной сети

Рисунок 26.12 показывает функционирование трех решений, представленное в виде кривых рабочих характеристик приемника. Испытание (b) обеспечивает значительно лучшую сеть, чем два других. Это дело случая, зависящее от первоначально используемых весов. При одной и той же установке порога нейронная сеть, разработанная в испытании "b" может обнаружить 24 из 25 целей (т.е. 96 % изображений гласных) при соотношении ложных срабатываний всего 1 к 25 не целям (т.е. 4 % изображений согласных). Не плохо, учитывая абстрактный характер этой задачи, и использование самого общего решения.

Несколько последних комментариев по нейронным сетям. Получение нейронной сети сходящейся во время обучения может оказаться делом хитрым. Если ошибка сети (ESUM) уменьшается не стабильно, программа должна быть прервана, изменена и затем повторно запущена. Может быть предпринято несколько попыток, прежде чем будет достигнут успех. Чтобы повлиять на сходимость, могут быть изменены три вещи: (1) MU,

(2) величина начальных случайных весов и (3) число скрытых слоев (в порядке, в котором их следует изменять).

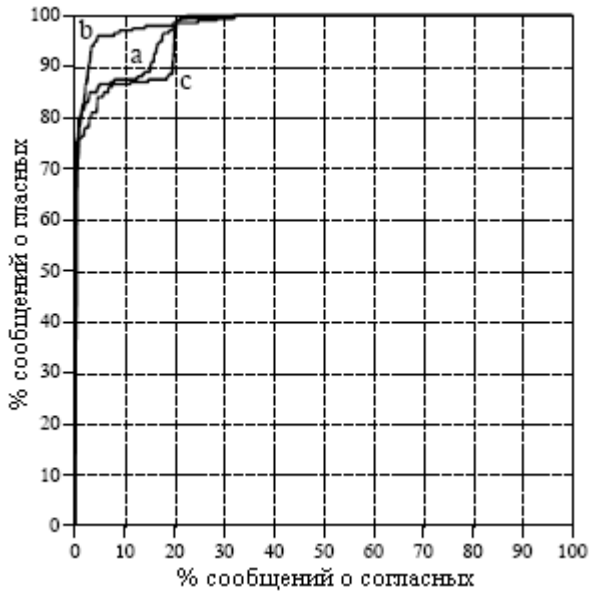


Рис. 26.12 РХП анализ примеров нейронной сети

Наиболее критическим пунктом в разработке нейронной сети является правомерность обучающих примеров. Например, в период, когда разрабатываются новые коммерческие продукты, доступными являются только результаты испытания прототипов, моделирования, умозрительные предположения и т.д. Если нейронная сеть обучается на этой предварительной информации, в конечном приложении она может не функционировать должным образом. Любое различие между обучающей базой данных и возможными данными будет ухудшать функционирование нейронной сети (закон Мэрфи для нейронных сетей). Не пытайтесь предложить другой вариант нейронной сети, касающийся этой проблемы, Вы не сможете!

Разработка рекурсивных фильтров

Главы 19 и 20 показывают, как проектировать рекурсивные фильтры со стандартными частотными характеристиками: фильтры высоких частот, низких частот, полосовые и т.д. А что, если Вам нужно что-нибудь другое? Ответ следующий, проектировать рекурсивный фильтр так, как если бы Вы были нейронной сетью: начать с обобщенного набора рекурсивных коэффициентов и использовать итерации, чтобы медленно превращать их в то, что Вы хотите. Такая техника важна по двум причинам. Первое, она позволяет разрабатывать заказные рекурсивные фильтры без необходимости сталкиваться с математикой z-преобразования. Второе, она показывает, что идеи обычной ЦОС и нейронных сетей могут быть объединены, для формирования превосходных алгоритмов.

Основная программа для этого метода показана в таблице 26.4, с двумя подпрограммами в таблице 26.5. Массив T[ ] содержит желаемую частотную характеристику, некоторую кривую, которую мы сконструировали вручную. Так как эта программа базируется на БПФ, длины сигналов должны быть кратны степени двойки. Эта программа, в том виде как она написана, использует БПФ длиной 256 точек, что определено переменной N% в строке 130. Это означает, что значения от T [0] до T [128] соответствуют частотам, лежащим между 0 и 0,5 от частоты дискретизации. В этом

массиве содержатся только амплитуды; фаза в этом проекте не контролируется и становится такой, какой она становится.

Рекурсивные коэффициенты устанавливаются к своим первоначальным значениям в строках 270-310, обычно их отбирают таким образом, чтобы они представляли систему *тождеств*. Не используйте здесь случайные числа, иначе начальный фильтр будет неустойчивым. Рекурсивные коэффициенты содержатся в массивах A[ ] и B[ ]. Переменная NP% устанавливает число полюсов в разрабатываемом фильтре. Например, если NP% равно 5 коэффициенты “a” пробегают значения от A[0] до A[5], в то время как коэффициенты “b” от B[1]до B[5].

Таблица 26.4

```
100 'ITERATIVE DESIGN OF RECURSIVE FILTER
110 '
120                                     'INITIALIZE
130 N% = 256                            'number of points in FFT
140 NP% = 8                             'number of poles in filter
150 DELTA = .00001                      'perturbation increment
160 MU = .2                             'iteration step size
170 DIM REX[255]                        'real part of signal during FFT
180 DIM IMX[255]                        'imaginary part of signal during FFT
190 DIM T[128]                          'desired frequency response (mag only)
200 DIM A[8]                            'the "a" recursion coefficients
210 DIM B[8]                            'the "b" recursion coefficients
220 DIM SA[8]                           'slope for "a" coefficients
230 DIM SB[8]                           'slope for "b" coefficients
240 '
250 GOSUB XXXX                          'mythical subroutine to load T[ ]
260 '
270 FOR P% = 0 TO NP%                   'initialize coefficients to the identity system
280 A[P%] = 0
290 B[P%] = 0
300 NEXT P%
310 A[0] = 1
320 '
330 '                                   'ITERATION LOOP
340 FOR ITER% = 1 TO 100                'loop for desired number of iterations
350 GOSUB 2000                          'calculate new coefficients
360 PRINT ITER% ENEW MU                 'print current status to video screen
370 IF ENEW > EOLD THEN MU = MU/2       'adjust the value of MU
380 NEXT ITER%
390 '
400 '
410 FOR P% = 0 TO NP%                   'PRINT OUT THE COEFFICIENTS
420 PRINT A[P%] B[P%]
430 NEXT P%
440 '
450 END
```

Как упоминалось ранее, для процедуры итерации нужна *отдельная* величина, описывающая, насколько хорошо функционирует данная система. Это обеспечивается переменной ER (для ошибки) и вычисляется в подпрограмме 3000. Строки от 3040 до 3080

загружают единичный импульс в массив IMX[ ]. Следующие строки 3100-3150 используют этот единичный импульс в качестве входного сигнала рекурсивного фильтра определяемого текущими значениями A[ ] и B[ ]. Таким образом, выходной сигнал этого фильтра будет являться *импульсной характеристикой* текущей системы и сохраняться в массиве REX[ ]. Затем, как это показано в строке 3170, за счет выполнения БПФ импульсной характеристики, находится частотная характеристика системы. Подпрограмма 1000 - это программа БПФ, распечатка которой приведена в таблице 12.4 Главы 12. Эта подпрограмма БПФ возвращает частотную характеристику в прямоугольной форме, перезаписывая массивы REX[ ] и IMX[ ].

Затем, в строках 3200-3250 вычисляется ER, *среднеквадратическая ошибка* между амплитудой текущей частотной характеристики и желаемой частотной характеристики. Обратите пристальное внимание на то, как находится эта ошибка. Итеративная работа программы оптимизирует эту ошибку, делая очень важным способ, которым она определяется. Цикл FOR-NEXT проходит через каждую частоту в частотной характеристике. В строке 3220 для каждой частоты, из данных в прямоугольных координатах, вычисляется текущее значение частотной характеристики. В строке 3230 с помощью вычитания желаемого значения T[ ] из текущего значения MAG, находится величина ошибки на этой частоте. Затем величина этой ошибки возводится в квадрат и добавляется к аккумулирующей переменной ER. После прохождения цикла через все частоты строка 3250 завершает вычисления, определяя среднеквадратическую ошибку всей частотной характеристики ER.

Строки с 340 по 380 управляют итерационным циклом программы. Подпрограмма 2000 - это программа, где производятся изменения рекурсивных коэффициентов. Первым действием этой программы является определение текущего значения ER и сохранения его в другой переменной EOLD (строки 2040 и 2050). После того как система обновит коэффициенты, снова определяется значение ER и присваивается переменной ENEW (строки 2270 и 2280).

Переменная MU управляет размером шага итераций так же, как в предыдущей программе нейронной сети. В этой программе используется более совершенная характеристика: автоматическая приспособляемость к значению MU. Это и есть причина наличия двух переменных EOLD и ENEW. При запуске программы, MU устанавливается к относительно высокому значению порядка 0,2 (строка 160). Это позволит осуществлять быструю сходимость, но будет ограничивать то, как близко фильтр сможет подойти к оптимальному решению. По мере выполнения итераций будет достигнута точка, определяемая тем, что ENEW будет *больше*, чем EOLD, где не будет происходить никакого прогресса. Каждый раз, когда это будет происходить, строка 370 уменьшит значение MU.

Подпрограмма 2000 обновляет рекурсивные коэффициенты в соответствии с методом крутого спуска: вычисляется наклон каждого коэффициента, а затем коэффициент изменяется на величину пропорциональную его наклону. Строки 2080-2130 вычисляют наклон для коэффициентов “a”, сохраняя величины в массиве SA[ ]. Подобно этому строки 2150-2200 вычисляют наклон для коэффициентов “b”, сохраняя величины в массиве SB[ ]. Затем строки 2220-2250 модифицируют каждый из рекурсивных коэффициентов на величину пропорциональную их наклонам. В этой программе константа пропорциональности - это просто размер шага MU. Поскольку существует только *один* пример для сравнения - желаемая частотная характеристика, для константы пропорциональности не требуется никакого условия ошибки.

Последняя проблема это то, как программа вычисляет наклоны рекурсивных коэффициентов. В примере с нейронной сетью *уравнение* для наклона было выведено. Эта процедура не может быть использована здесь, потому что это потребовало бы взятия производной *по всему* ДПФ. Вместо этого, применяется метод решения задачи "в лоб": фактически рекурсивный коэффициент изменяется на величину небольшого приращения,

Таблица 26.5

```
2000 'SUBROUTINE TO CALCULATE THE NEW RECURSION COEFFICIENTS
2010 'Variables entering routine: A[ ], B[ ], DELTA, MU
2020 'Variables exiting routine: A[ ], B[ ], EOLD, ENEW
2030 '
2040 GOSUB 3000                                    'FIND THE CURRENT ERROR
2050 EOLD = ER                                     'store current error in variable, EOLD
2060 '
2070                                               'FIND THE ERROR SLOPES
2080 FOR P% = 0 TO NP%                             'loop through each "a" coefficient
2090 A[P%] = A[P%] + DELTA                         'add a small increment to the coefficient
2100 GOSUB 3000                                    'find the error with the change
2110 SA[P%] = (ER-EOLD)/DELTA                      'calculate the error slope, store in SA[ ]
2120 A[P%] = A[P%] – DELTA                         'return coefficient to original value
2130 NEXT P%
2140 '
2150 FOR P% = 1 TO NP%                             'repeat process for each "b" coefficient
2160 B[P%] = B[P%] + DELTA
2170 GOSUB 3000
2180 SB[P%] = (ER-EOLD)/DELTA                      'calculate the error slope, store in SB[ ]
2190 B[P%] = B[P%] - DELTA
2200 NEXT P%
2210 '                                             'CALCULATE NEW COEFFICIENTS
2220 FOR P% = 0 TO NP%                             'loop through each coefficient
2230 A[P%] = A[P%] - SA[P%] * MU                   'adjust coefficients to move "downhill"
2240 B[P%] = B[P%] - SB[P%] * MU
2250 NEXT P%
2260 '
2270 GOSUB 3000                                    'FIND THE NEW ERROR
2280 ENEW = ER                                     'store new error in variable, ENEW
2290 '
2300 RETURN
3000 'SUBROUTINE TO CALCULATE THE FREQUENCY DOMAIN ERROR
3010 'Variables entering routine: A[ ], B[ ], T[ ]
3020 'Variables exiting routine: ER
3030 '
3040 FOR I% = 0 TO N%-1                            'LOAD SHIFTED IMPULSE INTO IMX[ ]
3050 REX[I%] = 0
3060 IMX[I%] = 0
3070 NEXT I%
3080 IMX[12] = 1
3090 '                                             'CALCULATE IMPULSE RESPONSE
3100 FOR I% = 12 TO N%-1
3110 FOR J% = 0 TO NP%
3120 REX[I%] = REX[I%] + A[J%] * IMX[I%-J%] + B[J%] * REX[I%-J%]
3130 NEXT J%
3140 NEXT I%
3150 IMX[12] = 0
3160 '                                             'CALCULATE THE FFT
3170 GOSUB 1000                                    'Table 12-4, uses REX[ ], IMX[ ], N%
3180 '
3190                                               'FIND FREQUENCY DOMAIN ERROR
3200 ER = 0                                        'zero ER, to use as an accumulator
3210 FOR I% = 0 TO N%/2                            'loop through each positive frequency
3220 MAG = SQR(REX[I%]^2 + IMX[I%]^2)              'rectangular --> polar conversion
3230 ER = ER + ( MAG - T[I%] )^2                   'calculate and accumulate squared error
3240 NEXT I%
3250 ER = SQR( ER/(N%/2+1) )                       'finish calculation of error, ER
3260 '
3270 RETURN
```

и затем непосредственно вычисляется новое значение ER. В этом случае наклон находится как изменение ER, деленное на величину приращения. Конкретно текущее значение ER находится в строках 2040-2050 и сохраняется в переменной EOLD. Цикл в строках 2080-2130 проходит через каждый из коэффициентов “a”. Первым действием внутри этого цикла является сложение небольшого приращения DELTA с рекурсивным коэффициентом, над которым сейчас идет работа (строка 2090). Подпрограмма 3000 вызывается строкой 2100 для нахождения значения ER с модифицированным коэффициентом. Затем строка 2120 вычисляет наклон этих коэффициентов как: (ER - EOLD)/DELTA. Строка 2120 затем восстанавливает модифицированный коэффициент, вычитая значение DELTA.

На рис. 26.13 показано несколько примеров фильтров разработанных с помощью этой программы. Пунктирной линией обозначена желаемая частотная характеристика, в то время как сплошной линией частотная характеристика разработанного фильтра. На компьютере Pentium с тактовой частотой 100 МГц до достижения сходимости каждого из этих фильтров потребовалось несколько минут. На рис.26.13a показан 8 полюсный низкочастотный фильтр с одинаковым весом ошибки по всему частотному спектру (как написана программа). На рис.26.13b показан такой же фильтр за исключением того, что когда происходило вычисление ER, ошибка в полосе заграждения была умножена на *восемь*. Это приводит к тому, что фильтр имеет меньшие пульсации в полосе заграждения за счет большей пульсации в полосе пропускания.

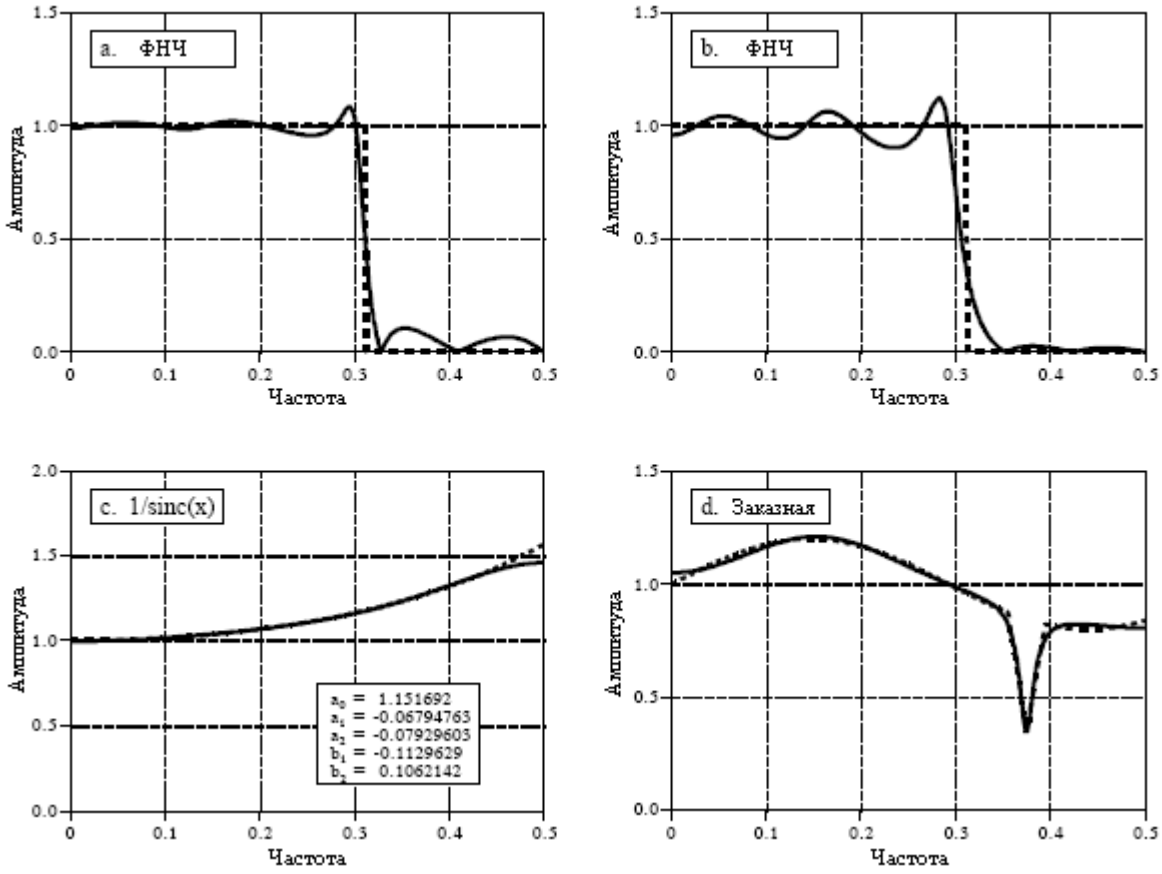


Рис. 26.13 Итеративное проектирование рекурсивных фильтров

Рис. 26.13 с показывает 2 полюсный фильтр для *1/sinc(x)*. Как обсуждалось в Главе 3, он может быть использован для противодействия хранению нулевого порядка во время цифро-аналогового преобразования (см. рис. 3.6). Ошибка в этом фильтре только суммировалась между 0 и 0,45, приводя к лучшему соответствию в этом диапазоне за счет

худшего соответствия между 0,45 и 0,5. Наконец, рис.26.13d показывает произвольную, очень нерегулярную 6 полюсную частотную характеристику с резкой впадиной. Для достижения сходимости были установлены такие же первоначальные рекурсивные коэффициенты, как и у полосно-заграждающих фильтров.

ГЛАВА
27

СЖАТИЕ ДАННЫХ

Передача и хранение данных стоят денег. Чем с большим количеством информации имеют дело, тем больше это стоит. Несмотря на это, большинство цифровых данных хранится не в наиболее компактной форме. Чаще они хранятся каким-нибудь способом, делающим их использование более легким, наподобие: ASCII кода из текстовых процессоров, двоичного кода, который может быть исполнен на компьютере, индивидуальных отсчетов от систем сбора данных и т.д. Обычно, эти легко используемые методы кодирования требуют файлов данных, примерно в два раза больше, чем действительно необходимо для представления информации. Сжатие данных является общим подходом различных алгоритмов и программ, разработанных для обращения с этой проблемой. *Программа сжатия данных* используется для преобразования данных из легкого в использовании формата в формат, оптимизированный с целью достижения компактности. Подобно этому, *программа*, *осуществляющая операцию обратную сжатию данных*, возвращает информацию к ее первоначальной форме. В этой главе мы исследуем пять методов сжатия данных. Первые три являются простыми методами кодирования, носящими название: кодирование длины повторов, кодирование Хаффмана и дельта кодирование. Последние два представляют собой сложные процедуры, зарекомендовавшие себя, как промышленные стандарты: LZW и JPEG.

Стратегии сжатия данных

В таблице 27.1 показаны два различных способа, которыми алгоритмы сжатия данных могут быть разбиты на категории. В таблице 27.1а, методы классифицировались по принципу: либо методы **без потерь**, либо методы **с потерями**. Метод сжатия без потерь означает, что восстановленный файл данных *идентичен* оригиналу. Для многих типов данных это абсолютно необходимо, например, для: исполняемых кодов, файлов текстовых процессоров, чисел сведенных в таблицу и т.д. Вы не можете позволить себе положить не на свое место даже один единственный бит информации этого типа. Для сравнения, файлы данных, представляющих изображения и другие сигналы систем сбора данных, при хранении или передаче не обязательно должны содержаться в безупречном состоянии. В реальном мире все измерения неотъемлемо содержат некоторое количество *шума*. Если внесенные в этот сигнал изменения будут похожи на небольшое количество дополнительного шума, то большого вреда сделано не будет. Методы сжатия, которые допускают подобный тип ухудшения, называются методами **с потерями**. Это различие важно, поскольку при сжатии методы с потерями намного более эффективны, чем методы без потерь. Чем больше коэффициент сжатия, тем к данным добавляется большее количество шума.

Изображения, переданные по интернету - превосходный пример того, почему сжатие данных является таким важным. Предположим, что при помощи компьютерного модема, обладающего скоростью 33,6 Кбит /сек нам нужно загрузить цифровую цветную фотографию. Если изображение не сжато (например, файл с расширением TIFF), он будет

содержать приблизительно *600* килобайт данных. Если бы он был сжат при помощи методов *без потерь* (подобного используемому в формате GIF), он будет около половины от первоначального размера, или *300* килобайт. Если же использовалось сжатие *с потерями* (JPEG файл), он будет приблизительно *50* килобайт. Дело в том, что время загрузки для этих трех эквивалентных файлов составит *142* секунды, *71* секунду и *12* секунд, соответственно. Это большая разница! JPEG является наилучшим выбором для цифровых фотографий, в то время как GIF используется с *нарисованными* изображениями, наподобие эмблем компаний, имеющим большие области одного и того же цвета.

Таблица 27.1a

Без потерь	**С потерями**
кодирование длины повторов кодирование Хаффмана дельта кодирование LZW	CS&Q JPEG MPEG

Таблица 27.1b

Метод	Размер группы:	
	входной	выходной
CS&Q	фиксированный	фиксированный
кодирование Хаффмана	фиксированный	переменный
арифметическое кодирование	переменный	переменный
кодирование длины повторов, LZW	переменный	фиксированный

Наш второй способ классификации методов сжатия данных показан в таблице 27.1b. Большинство программ сжатия данных работает следующим образом, из исходного файла берется группа данных, она некоторым способом сжимается, а затем сжатая группа записывается в выходной файл. Например, одним из методов в этой таблице является метод **CS&Q**, короткий для **грубой дискретизации и/или квантования**. Предположим, что мы осуществляем сжатие оцифрованной формы волны такой, как *12* битовый звуковой сигнал. Мы могли бы считать из исходного файла два соседних отсчета (24 бита), полностью отказаться от одного из отсчетов, отказаться от *4* младших значащих бит из другого отсчета, а затем записать оставшиеся *8* бит в выходной файл. Используя алгоритм с потерями, при *24* битах на входе и *8* битах на выходе, мы осуществили сжатие с отношением *3:1*. Хотя само по себе это сделано довольно грубо, при использовании совместно с методом, называемым *сжатие преобразованием*, это достаточно эффективно. Как мы обсудим позже это основа JPEG.

Как показано в таблице 27.1, CS&Q выполнен по схеме с фиксированным входом и фиксированным выходом. То есть из входного файла считывается фиксированное число битов, и в выходной файл записывается меньшее фиксированное число битов. Другие методы сжатия позволяют считывать или записывать переменное число битов. По мере Вашего знакомства с описаниями каждого из этих методов сжатия, вернитесь к этой таблице, чтобы понять, как они вписываются в эту схему классификации. Почему в этой таблице не приведены JPEG и MPEG? Данные алгоритмы являются составными, объединяющими в себе много различных методов. Они слишком сложны для того, чтобы их можно было классифицировать по таким простым категориям.

Кодирование длины повторов

Файлы данных часто содержат один и тот же повторяющийся подряд в строке символ. Например, в текстовых файлах используются многократные интервалы для разделения предложений, для формирования отступов параграфов, для осуществления форматирования таблиц и диаграмм и т.д. Оцифрованные *сигналы* могут также иметь последовательность повторения одной и той же величины, показывающую, что сигнал в эти моменты не изменяется. Например, изображение ночного неба будет содержать длинные повторы символа или символов, представляющих черный фон. Аналогичным образом, цифровая музыка может иметь между песнями длинные повторы нулей. Кодирование длины повторов является простым методом сжатия файлов такого типа.

Кодирование длины повторов иллюстрируется на рис. 27.1 для последовательности данных, имеющей часто повторяющиеся *нули*. Каждый раз, когда во входных данных сталкиваются с нулем, в выходной файл записываются *два* значения. Первое из этих значений - ноль, флаг указывающий, что начинается сжатие длины повторов. Второе значение – количество нулей в последовательности. Сжатие будет иметь место, если средняя длина повторов больше двух. С другой стороны, огромное количество отдельных нулей в данных могут сделать кодируемый файл большим, чем оригинал.

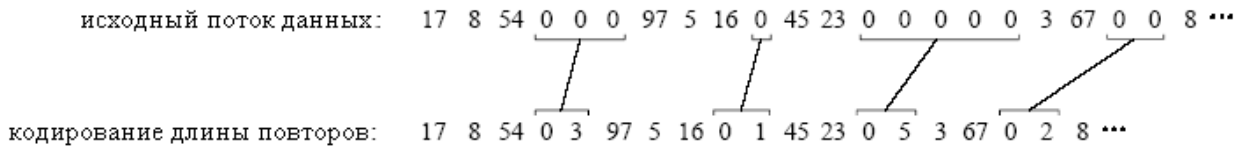


Рис. 27.1 Пример кодирования длинны повторов

Разработано большое число схем кодирования длины повторов. Например, с входными данными можно обращаться, как с индивидуальными байтами или группой байтов, которые представляют что-либо более сложное, наподобие чисел с плавающей запятой. Кодирование длины повторов может быть применено всего к *одному* из символов (так же, как было показано выше с *нулем*), к *нескольким* символам или *всем* символам.

Хорошим примером обобщенной схемы кодирования длины повторов является **PackBits** (упаковщик битов – прим. перев.), созданный для пользователей Макинтош (Макинтош - это сорт яблок. Такое название фирма Эппл, что в переводе означает яблоко, дала одному из типов своих компьютеров – прим. перев.). Каждый байт (восемь битов) из входного файла заменяется девятью битами в сжатом файле. Добавленный девятый бит интерпретируется как *знак* числа. То есть, каждый символ, считанный из входного файла, находится между *0* и *255*, тогда как каждый символ, записанный в закодированный файл, находится между *–255* и *255*. Для того чтобы понять, как это используется, рассмотрим входной файл: 1, 2, 3, 4, 2, 2, 2, 2, 4; и сжатый файл, сгенерированный алгоритмом PackBits: 1, 2, 3, 4, 2,-3, 4. Программа сжатия просто перемещает каждое число из входного файла в сжатый файл, за исключением повторов: 2, 2, 2, 2. Они представляются в сжатом файле двумя числами: 2, -3. Первое число ("2") показывает, из какого символа состоят повторения. Второе число ("-3") показывает, что число символов в повторах находится суммированием его абсолютного значения с единицей. Например, 4, -2 означает 4, 4, 4; 21, -4 означает 21, 21, 21, 21, 21 и т.д.

Неудобство в PackBits заключается в том, что эти девять битов должны быть переформатированы в стандартный восьмибитовый байт, используемый в компьютере при хранении и передаче. Полезная модификация этой схемы может быть сделана тогда, когда на входной файл накладывается ограничение, чтобы он был текстовым в кодах ASCII. Как показано в таблице 27.2, каждый ASCII символ обычно хранится в виде целого байта

(восемь битов), но в действительности для идентификации символа используется всего *семь* битов. Другими словами, значения от 127 до 255 не определены каким-нибудь стандартизированным образом и их не нужно хранить или передавать. Это позволяет использовать восьмой бит для индикации того, происходит или нет кодирование длины повторов.

Кодирование Хаффмана

Этот метод назван по имени Д.А. Хаффмана, который разработал эту процедуру в 1950-х. На рис. 27.2 показана гистограмма оценки байта в большом файле ASCII. Более чем 96% этого файла состоит всего из 31 символа: букв нижнего регистра, пробела, запятой, точки и возврата каретки. Эти наблюдения могут быть использованы для создания подходящей схемы сжатия для этого файла. Для начала, мы присвоим каждому из этих 31 обычных символов пяти-битовый двоичный код: 00000 = "a", 00001 = "b", 00010 = "c" и т.д. Это позволяет уменьшить в размере 96% этого файла, в соотношении 5/8. Флагом показывающим, что передаваемый символ не является одним из 31 обычных символов, будет последний код из пяти-битовых кодов 11111. Для символа, не являющегося одним из 31 обычных символов, следующие за флагом восемь битов показывают, что это за символ в файле, в соответствии со стандартным назначением ASCII. В результате для 4% символов во входном файле требуется 5+8=13 битов. Идея состоит в том, чтобы приписать часто используемым символам меньшее количество битов, а редко используемым символам большее количество битов. В этом примере *среднее* число битов, требуемое на исходный символ, равно: 0,96×5 + 0,04×13 = 5,32. Другими словами, полный коэффициент сжатия: 8 битов/5,32 битов или около 1,5/1.

Таблица 27.2

0	Нуль	32	пробел	64	@	96	`
1	Начало заголовка	33	!	65	A	97	a
2	Начало текста	34	“	66	B	98	b
3	Конец текста	35	#	67	C	99	c
4	Конец передачи данных	36	$	68	D	100	d
5	Запрос	37	%	69	E	101	e
6	Подтверждение	38	&	70	F	102	f
7	Звонок, звуковой сигнал	39	‘	71	G	103	g
8	Возврат на один символ	40	(	72	H	104	h
9	Горизонтальная табуляция	41	)	73	I	105	i
10	Перевод строки	42	*	74	J	106	j
11	Вертикальная табуляция, домой	43	+	75	K	107	k
12	Перевод формы, очистить экран	44	,	76	L	108	l
13	Возврат каретки	45	-	77	M	109	m
14	Сдвиг наружу	46	.	78	N	110	n
15	Сдвиг внутрь	47	/	79	O	111	o
16	Отключение от линии данных	48	0	80	P	112	p
17	Управление устройством 1	49	1	81	Q	113	q
18	Управление устройством 2	50	2	82	R	114	r
19	Управление устройством 3	51	3	83	S	115	s
20	Управление устройством 4	52	4	84	T	116	t
21	Отрицательное подтверждение	53	5	85	U	117	u
22	Холостая синхронизация	54	6	86	V	118	v
23	Конец передав. блока данных	55	7	87	W	119	w

24	Отменить	56	8	88	X	120	x
25	Конец среды	57	9	89	Y	121	y
26	Замена	58	:	90	Z	122	z
27	Выход	59	;	91	[	123	{
28	Разделитель файла	60	<	92	\	124	|
29	Разделитель группы	61	=	93	]	125	}
30	Разделитель записи	62	>	94	^	126	~
31	Разделитель блока	63	?	95	_	127	Удалить

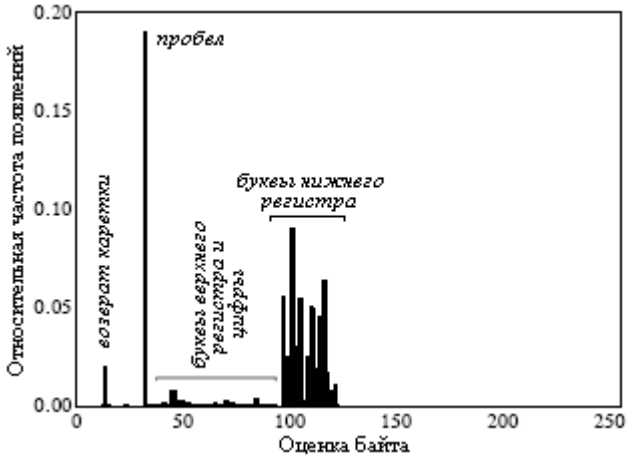


Рис. 27.2 Гистограмма текста

Кодирование Хаффмана доводит эту идею до крайности. Символам, встречающимся наиболее часто, таким как пробел и точка, может быть присвоено всего один или два бита. Нечасто используемые символы, такие как: !, @, #, $ и %, могут потребовать дюжины битов или более. В терминах математики оптимальная ситуация достигается тогда, когда число битов, используемых для каждого символа, пропорционально логарифму вероятности появления символа.

Хитрой особенностью кодирования Хаффмана является то, каким образом коды переменной длины могут быть упакованы вместе. Представьте получение потока последовательных данных, состоящего из единиц и нулей. Если каждый символ представлен с помощью восьми битов, то разбиением потока на куски по 8 битов Вы можете непосредственно отделить один символ от другого. Теперь рассмотрим поток данных с кодированием Хаффмана, в котором каждый символ может иметь различное число битов. Как вам отделить один символ от другого? Ответ находится в надлежащем выборе кодов Хаффмана, дающих правильное разделение. Как это все работает, проиллюстрирует следующий пример.

На рис. 27.3 показана упрощенная схема кодирования Хаффмана. Символы от *A* до *G* появляются в исходном потоке данных с показанными вероятностями. Поскольку символ *A* является наиболее обычным, мы будем представлять его с помощью единственного бита, кодом: 1. Следующий наиболее обычный символ *B* получает два бита, код: 01. Так продолжается по наименее часто встречаемый символ *G*, которому присваивается шесть битов 000011. Как показано на этой иллюстрации, коды переменной длины реорганизуются в группы по восемь битов - стандарт для использования в компьютере.

Когда происходит операция обратная сжатию, все восьмибитовые группы располагаются началом к концу таким образом, чтобы сформировать длинную последовательную строку из единиц и нулей. Посмотрите внимательно на таблицу

кодировки на рис. 27.3 и обратите внимание, что каждый код состоит из двух частей: числа нулей перед *единицей* и произвольного двоичного кода после *единицы*. Это позволяет разделить двоичный поток данных на коды без необходимости в разделителях или других маркерах между кодами. Программа, осуществляющая операцию обратную сжатию, отслеживает поток единиц и нулей пока не сформируется правильный код, а затем начинает снова искать следующий символ. Способ, которым формируются коды, гарантирует отсутствие существования двусмысленности в разделении.

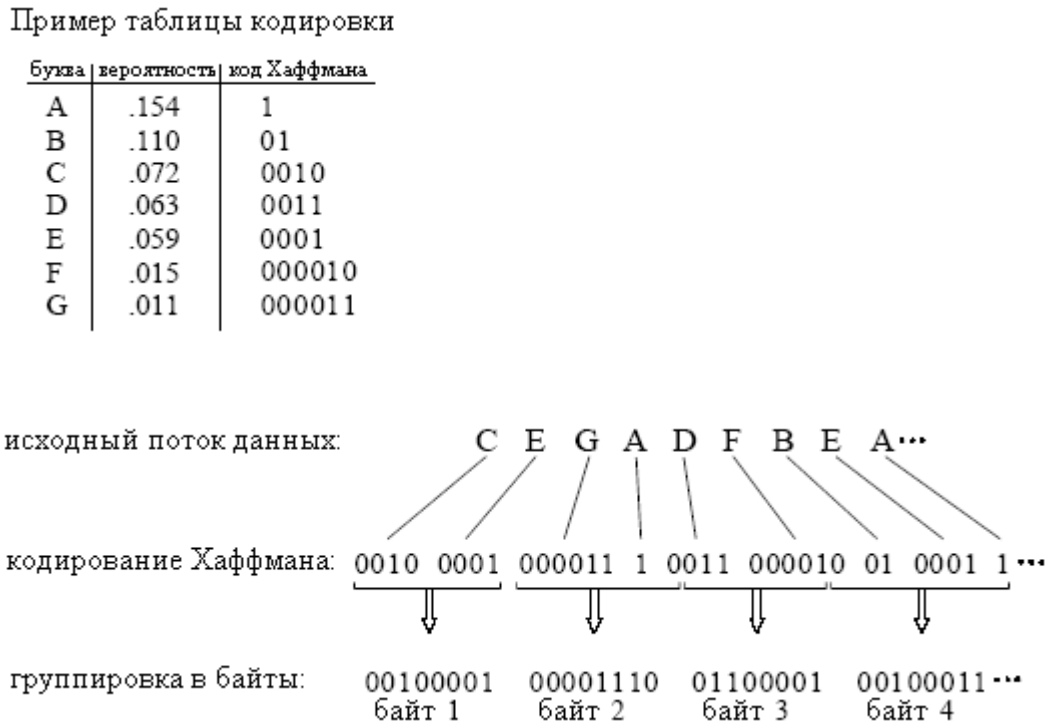


Рис. 27.3 Кодирование Хаффмана

Более искусная версия подхода Хаффмана называется **арифметическим кодированием**. В этой схеме *последовательность* символов представляется при помощи индивидуальных кодов, соответствующих вероятности их появления. Это дает, скажем, 5-10 % преимущества лучшего сжатия данных. Обычной стратегией является также стратегия, когда за каждым кодированием Хаффмана или арифметическим кодированием следует кодирование длины повторов. Как Вы возможно и предполагали, эти виды алгоритмов очень сложны, и обычно их оставляют специалистам по сжатию данных.

Для реализации кодирования Хаффмана или арифметического кодирования алгоритмы сжатия и алгоритмы, осуществляющие операцию обратную сжатию, должны согласовываться с двоичными кодами, используемыми для представления каждого символа (или группы символов). Это может быть сделано одним из двух способов. Самым простым является использование предварительно разработанной, всегда одной и той же, вне зависимости от сжимаемой информации, таблицы кодировки. Более сложные схемы используют кодирование, оптимизированное для конкретных используемых данных. Это требует того, чтобы таблица кодировки включалась в сжимаемый файл для использования ее программой, осуществляющей операцию обратную сжатию. Оба метода являются типичными.

Дельта кодирование

В науке, технике и математике греческая буква *дельта* (*Δ*) используется для обозначения изменения переменной. Термин *дельта кодирование* относится к нескольким методам, в которых вместо того, чтобы непосредственно хранить сами отсчеты, данные хранятся в виде разницы между последовательными отсчетами (или символами). На рис. 27.4 показан пример того, как это осуществляется. Первое значение в дельта закодированном файле является точно таким же, как и первое значение в исходном файле. Все следующие значения в закодированном файле равны разнице (дельте) между соответствующими значениями в исходном файле и *предыдущим* значением в исходном файле.

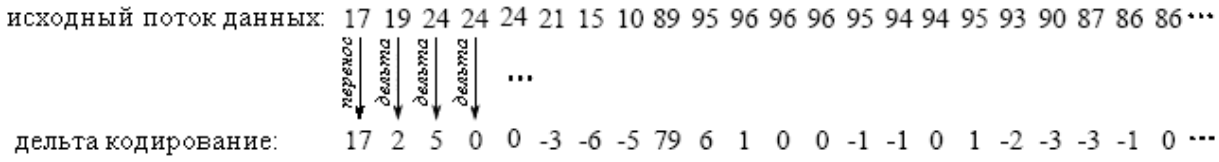


Рис. 27.4 Пример дельта кодирования

Дельта кодирование может использоваться для сжатия данных тогда, когда в исходных данных значения изменяются плавно, то есть, как правило, между соседними значениями существуют только небольшие изменения. Для текста ASCII и исполняемого кода это не тот случай, однако, он является очень типичным тогда, когда файл представляет собой *сигнал*. Например, на рис. 27.5a показан фрагмент звукового сигнала, оцифрованного - каждый отсчет до 8 битов, значение отсчета находится между –127 и 127. На рис. 27.5b показана версия дельта кодирования этого сигнала. Ключевой характеристикой здесь является то, что дельта закодированный сигнал имеет *более низкую амплитуду*, чем исходный сигнал. Другими словами, дельта кодирование увеличивает вероятность того, что значение каждого отсчета будет близким к нулю, и уменьшает вероятность того, что оно будет далеким от нуля. Такая неравная вероятность - это как раз то, что нужно для выполнения кодирования Хаффмана. Если исходный сигнал не изменяется или изменяется вдоль прямой линии, то дельта кодирование даст повторяющиеся, имеющие одно и тоже значение, отсчеты. Это как раз то, что требуется для кодирования длины повторов. Соответственно, дельта кодирование, сопровождаемое кодированием Хаффмана и/или кодированием длины повторов, представляет собой типичную стратегию сжатия сигналов.

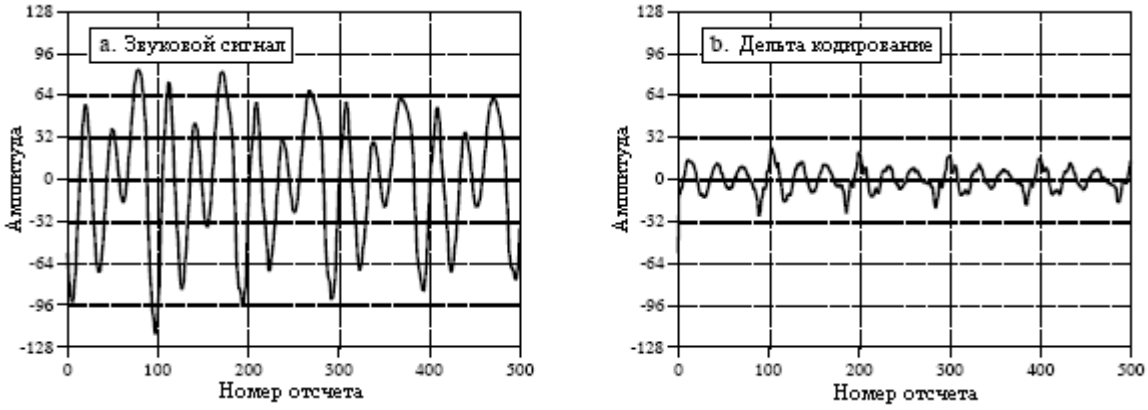


Рис. 27.5 Пример дельта кодирования

Идея, используемая в дельта кодировании, может быть распространена на более сложную технику, называемую **линейным прогнозирующим кодированием** или **LPC**. Чтобы понять LPC, представьте, что первые 99 отсчетов из входного сигнала были уже закодированы, и мы собираемся работать над отсчетом 100. Тогда Мы спрашиваем себя: каким является наиболее вероятное значение отсчета 100, если основываться на первых 99 отсчетах? При дельта кодировании ответом является то, что наиболее вероятным значением отсчета 100 будет то же самое значение, что и предыдущее значение отсчета 99. Это предполагаемое значение используется, как рекомендуемое для кодирования отсчета 100. То есть, в кодируемый файл помещается *разница* между отсчетом и предполагаемым значением. Дополнительным в LPC является формирование наилучшего предположения относительно того, каким является наиболее вероятное значение самого последнего отсчета. Это осуществляется с помощью просмотра нескольких последних отсчетов до просмотра самого последнего отсчета. Алгоритмы, используемые LPC, подобны рекурсивным фильтрам, использующим z-преобразование и другие интенсивные математические методы.

LZW сжатие

LZW сжатие названо по имени его разработчиков, А. Лемпела и Дж. Зева, с более поздними модификациями Тэрри А. Велча. Это передовой метод, предназначенный из-за своей простоты и многосторонности для общих целей сжатия данных. Обычно, Вы можете предполагать, что LZW осуществит сжатие текста, исполняемого кода и файлов данных подобного рода до примерно одной половины от их исходного размера. LZW также хорошо работает с чрезвычайно избыточными файлами данных наподобие сведенных в таблицу чисел, кодами компьютерных источников и полученными сигналами. Для этих случаев типичным является коэффициент сжатия 5:1. LZW представляет собой основу нескольких утилит для персональных компьютеров, призванных *"удвоить емкость вашего жесткого диска"*.

LZW сжатие всегда используется с файлами изображения в формате GIF и предлагается, как дополнительная функция в формате TIFF и PostScript. LZW сжатие защищено патентом США номер 4,558,302, выданным 10 декабря 1985 года корпорации Sperry (теперь корпорация Unisys). Информацию о коммерческом лицензировании можно получить в Welch Licensing Department, Law Department, M/SC2SW1, Unisys Corporation, Blue Bell, Pennsylvania, 19424-0001.

Как показано на рис. 27.6, LZW сжатие использует **таблицу кодировки**. Типичным выбором является обеспечение в таблице 4096 строк. В этом случае закодированные LZW данные состоят всего из 12 битовых кодов, каждый из которых относится к одной из строк в таблице кодировки. Операция обратная сжатию достигается путем взятия из сжатого файла каждого кода и перевода его при помощи таблицы кодировки с целью определения, какой символ или символы он представляет. Коды 0-255 в таблице кодировки всегда предназначены для представления отдельных байтов из входного файла. Например, если будут использоваться только эти первые 256 кодов, каждый байт исходного файла будет преобразован в 12 бит файла с LZW сжатием, давая в результате файл на 50% больше по размеру. Во время операции обратной сжатию каждый 12 битовый код посредством таблицы кодировки будет переведен обратно в отдельные байты. Конечно такая ситуация не имела бы никакой пользы.

В методе LZW сжатие достигается за счет использования кодов с 256 по 4095, представляющих *последовательности* байтов. Например, код 523 может представлять последовательность трех байтов: 231 124 234. Каждый раз, когда алгоритм сжатия сталкивается с этой последовательностью во входном файле, в кодируемый файл помещается код 523. Во время операции обратной сжатию посредством таблицы осуществляется перевод кода 523 для воссоздания правильной 3 байтовой

последовательности. Чем длиннее присваиваемая отдельному коду последовательность и, чем чаще она повторяется, тем достигается более высокая степень сжатия.

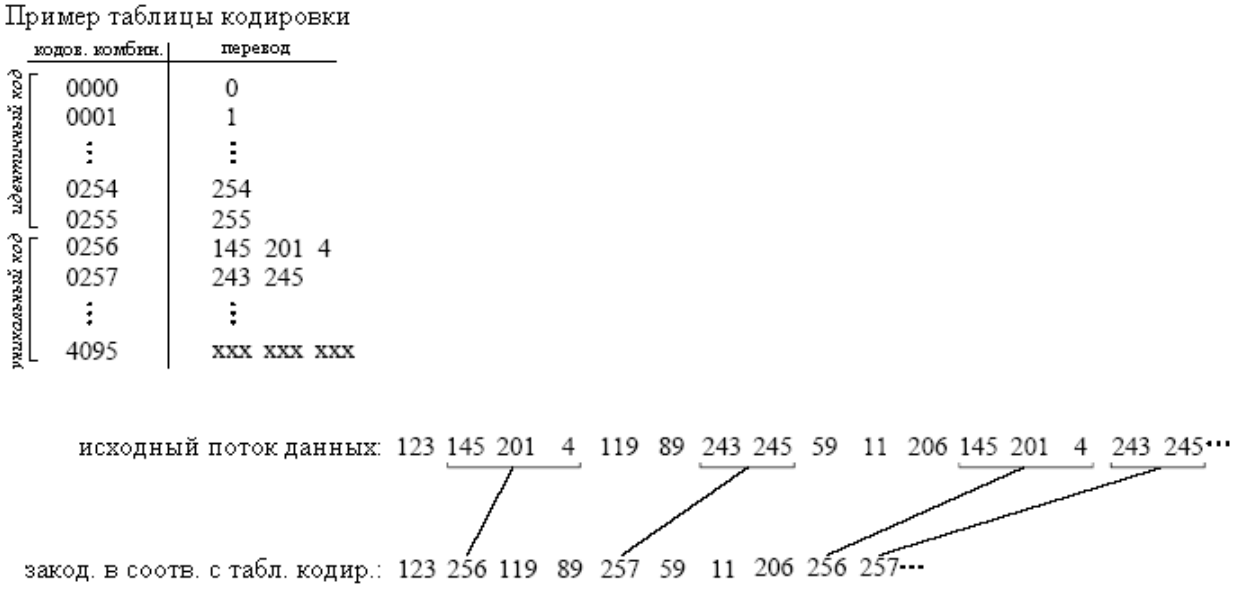


Рис. 27.6 Пример сжатия с использованием таблицы кодировки

Хотя это и является простым подходом, существуют два главных препятствия, которые должны быть преодолены: (1), как определить, какие последовательности должны быть включены в таблицу кодировки, и (2), как обеспечить программу, осуществляющую операцию обратную сжатию такой же таблицей кодировки, которая используется программой сжатия. Алгоритм LZW изящно решает обе эти проблемы.

Когда программа LZW начинает кодировать файл, в таблице кодировки заполнены только первые 256 строк, а остальная часть таблицы в этот момент пуста. Это означает, что первые коды, входящие в сжатый файл, являются просто отдельными байтами из входного файла, преобразованными до 12 бит. По мере продолжения кодирования LZW алгоритм определяет в данных повторяющиеся последовательности и добавляет их в таблицу кодировки. Если последовательность встречается, сжатие начинается второй раз. Ключевым моментом является то, что последовательность из входного файла не добавляется в таблицу кодировки, если ранее она уже не была помещена в сжатый файл в качестве индивидуального символа (коды от 0 до 255). Это важно, поскольку это позволяет программе, осуществляющей операцию обратную сжатию, непосредственно из сжатых данных *восстанавливать* таблицу кодировки, без необходимости отдельно передавать таблицу кодировки.

На рис. 27.7 показана блок-схема алгоритма LZW сжатия. В таблице 27.3 приведены подробнейшие детали для взятого в качестве примера, состоящего из 45 байтов текстовой строки в ASCII кодах, входного файла: *the/rain/in/Spain/falls/mainly/on/the/plain*. Когда мы говорим, что LZW алгоритм считывает из входного файла символ "a", мы подразумеваем, что он считывает величину 01100001 (97, выраженное в виде 8 битов), где 97 представляет собой "a" в ASCII кодах. Когда мы говорим, что он записывает символ "a" в закодированный файл, мы подразумеваем, что он пишет: 000001100001 (97, выраженное в виде 12 битов).

Алгоритм сжатия использует две переменные: *CHAR* и *STRING*. Переменная *CHAR* содержит один символ, т.е. значение отдельного байта, лежащее между 0 и 255. Переменная *STRING* является строкой переменной длины, т.е. группой из одного и более символов, причем каждый символ группы является отдельным байтом. В блоке 1 на рис. 27.7 программа начинает с того, что берет первый байт из входного файла и помещает его

в переменную *STRING*. В таблице 27.3 это действие показано в строке 1. За этим следует зацикливание алгоритма для каждого дополнительного байта во входном файле, управляемое в блок-схеме блоком 8. Каждый раз, когда из входного файла считывается байт (блок 2), он сохраняется в переменной *CHAR*. Затем просматривается таблица данных, для того чтобы определить, присвоен ли уже код (блок 3) конкатенации двух переменных *STRING+CHAR*.

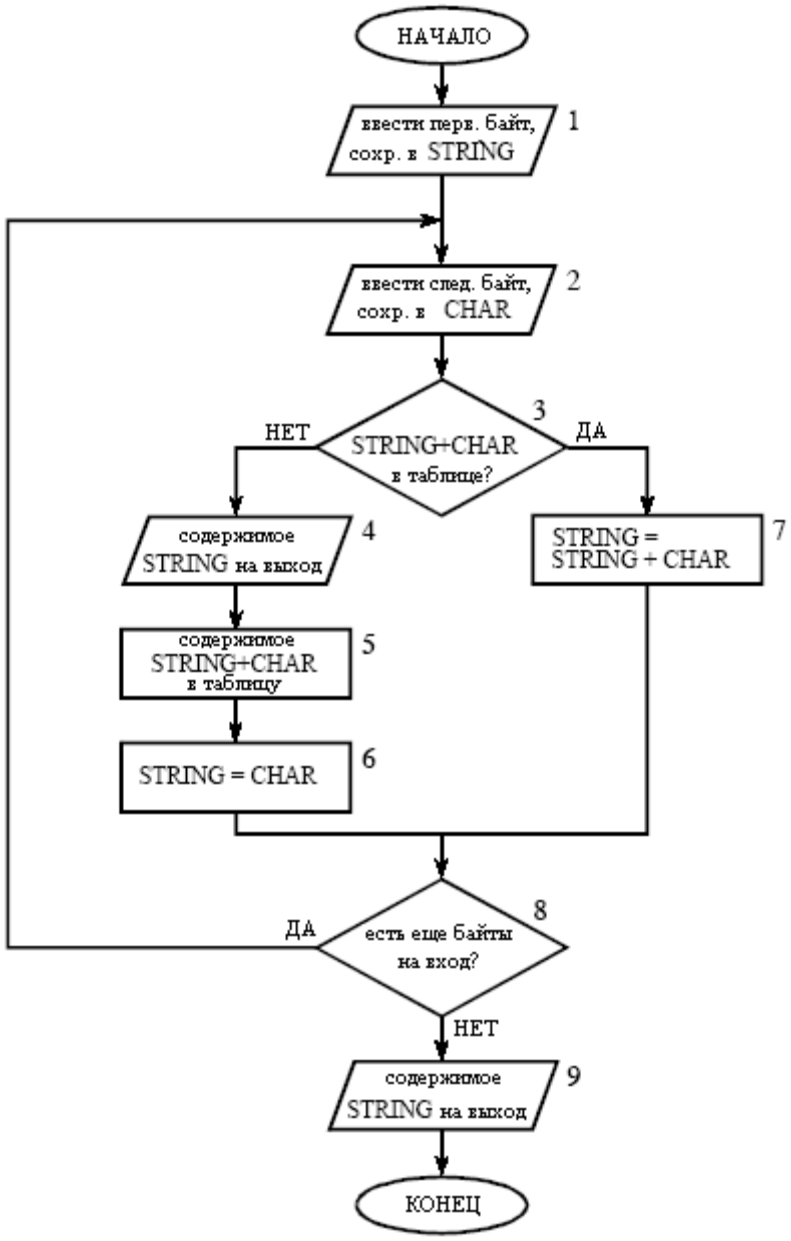


Рис. 27.7 Блок-схема алгоритма LZW сжатия

Если в таблице кодировки соответствие *не* найдено, производятся три действия, показанные в блоках 4, 5 и 6. В блоке 4, в сжатый файл записывается 12 битовый код, соответствующий содержимому переменной *STRING*. В блоке 5, в таблице создается новый код для конкатенации *STRING+CHAR*. В блоке 6, переменная *STRING* принимает значение переменной *CHAR*. Образец выполнения этих действий для первых 10 байтов файла примера показан в таблице 27.3 в строках со 2 по 10.

Таблица 27.3

№п/п	*CHAR*	*STRING +CHAR*	В таблице?	На выход	Добавить в таблицу	Новая переменная *STRING*	Комментарий
1	t	t				t	перв. символ – нет действий
2	h	th	нет	t	256 = th	h	
3	e	he	нет	h	257 = he	e	
4	/	e/	нет	e	258 = e/	/	
5	r	/r	нет	/	259 = /r	r	
6	a	ra	нет	r	260 = ra	a	
7	i	ai	нет	a	261 = ai	i	
8	n	in	нет	i	262 = in	n	
9	/	n/	нет	n	263 = n/	/	
10	i	/i	нет	/	264 = /i	i	
11	n	in	да (262)			in	перв. соответствие найдено
12	/	in/	нет	262	265 = in/	/	
13	S	/S	нет	/	266 = /S	S	
14	p	Sp	нет	S	267 = Sp	p	
15	a	pa	нет	p	268 = pa	a	
16	i	ai	да (261)			ai	соответствия ai, ain еще не в таблице
17	n	ain	нет	261	269 = ain	n	ain добавлено в таблицу
18	/	n/	да (263)			n/	
19	f	n/f	нет	263	270 = n/f	f	
20	a	fa	нет	f	271 = fa	a	
21	l	al	нет	a	272 = al	l	
22	l	ll	нет	l	273 = ll	l	
23	s	ls	нет	l	274 = ls	s	
24	/	s/	нет	s	275 = s/	/	
25	m	/m	нет	/	276 = /m	m	
26	a	ma	нет	m	277 = ma	a	
27	i	ai	да (261)			ai	соответствие ai
28	n	ain	да (269)			ain	соответствие длинной строке ain
29	l	ainl	нет	269	278 = ainl	l	
30	y	ly	нет	l	279 = ly	y	
31	/	y/	нет	y	280 = y/	/	
32	o	/o	нет	/	281 = /o	o	
33	n	on	нет	o	282 = on	n	
34	/	n/	да (263)			n/	
35	t	n/t	нет	263	283 = n/t	t	
36	h	th	да (256)			th	соответствия th, the еще не в таблице
37	e	the	нет	256	284 = the	e	the добавлено в таблицу
38	/	e/	да			e/	
39	p	e/p	нет	258	285 = e/p	p	
40	l	pl	нет	p	286 = pl	l	
41	a	la	нет	l	287 = la	a	
42	i	ai	да (261)			ai	соответствие ai
43	n	ain	да (269)			ain	соответствие длинной строке ain
44	/	ain/	нет	269	288 = ain/	/	
45	EOF	/		/			конец файла, выход *STRING*

Когда соответствие в таблице кодировки *найдено* (блок 3), в переменную *STRING*, без каких-либо других, имеющих место действий, заносится конкатенация *STRING+CHAR* (блок 7). То есть, если в таблице найдена соответствующая последовательность, то до

определения, нет ли также в таблице еще *более длинной* соответствующей последовательности, никаких действий не должно быть предпринято. Пример этого показан в строке 11, где последовательность *STRING+CHAR = in* определяется, как уже имеющая код в таблице. В строке 12 к последовательности добавляется следующий символ для входного файла /, а в таблице кодировки производится поиск in/. Поскольку этой длинной последовательности в таблице нет, программа *добавляет* ее в таблицу, выводит код для имеющейся в таблице более короткой последовательности (код 262) и начинает поиск последовательностей, начинающихся с символа /. Такая последовательность событий продолжается до тех пор, пока во входном файле не останется символов. Программа завершается записью в сжатый файл кода, соответствующего текущему значению переменной *STRING* (как показано в блоке 9 на рис. 27.7 и строке 45 таблицы 27.3).

Блок схема алгоритма, осуществляющего операцию обратную сжатию, показана на рис. 27.8. Каждый код считывается из сжатого файла и для обеспечения перевода сравнивается с таблицей кодировки. По мере обработки каждого кода в такой манере таблица кодировки обновляется так, что она постоянно соответствует таблице, используемой во время сжатия. Однако в программе осуществляющей операцию обратную сжатию имеются небольшие сложности. Существуют некоторые комбинации данных, которые приводят к получению кодов в алгоритме, осуществляющем операцию обратную сжатию, которых в его таблице кодировки еще нет. Такие непредвиденные обстоятельства обрабатываются в блоках 4, 5 и 6.

Для реализации наиболее простых LZW программ требуется всего несколько дюжин строк команд. Настоящие трудности заключаются в эффективном управлении таблицей кодировки. Подходы решения проблемы "в лоб" заканчиваются потребностью в большой памяти и медленным исполнением программы. В коммерческих программах LZW используются несколько ухищрений, улучшающих их работу. Например, проблема памяти возникает потому, что заранее не известно, какой будет длина строки для каждого кода символа. Большинство программ LZW обрабатывают эту проблему, используя одно из преимуществ таблицы кодировки, ее избыточный характер. Например, взгляните на строку 29 в таблице 27.3, где код 278 определен как ainl. Вместо хранения этих четырех байтов, код 278 мог бы быть сохранен как: *код 269+l*, где код 269 был предварительно определен в строке 17 как ain. Аналогично, код 269 был бы сохранен как: *код 261+n*, где код 261 был предварительно определен в строке 7 как ai. Такой образец всегда содержит: каждый код, который может быть выражен как предыдущий код, плюс один новый символ.

Время исполнения алгоритма сжатия ограничивается просмотром таблицы кодировки для определения наличия соответствия. В качестве аналогии представьте, что Вы хотите обнаружить, приведено ли в телефонном справочнике имя вашего друга. Ловушка состоит в том, что единственный справочник, которым Вы располагаете, составлен не в алфавитном порядке, а по телефонным номерам. Это заставляет Вас, в попытке отыскать необходимое Вам имя, просматривать страницу за страницей. Эта неэффективная ситуация точно такая же, как и просмотр всех 4096 кодов в поисках их соответствия определенной строке символов. Ответ: организуйте таблицу кодировки так, чтобы то, что Вы ищите, говорило Вам, где искать (подобно частично расположенному в алфавитном порядке телефонному справочнику). Другими словами, не присваивайте 4096 кодам последовательных позиций в памяти. А лучше разделите память на разделы, основываясь на том, какие там будут храниться последовательности. Например, предположим, что мы хотим найти, находится ли в таблице кодировки последовательность: *код 329+x*. Таблица кодировки должна быть организована так, чтобы "*x*" показывал место, откуда следует начинать поиск. Существует большое число схем управления таблицей кодировки подобного рода, и они могут быть весьма сложными.

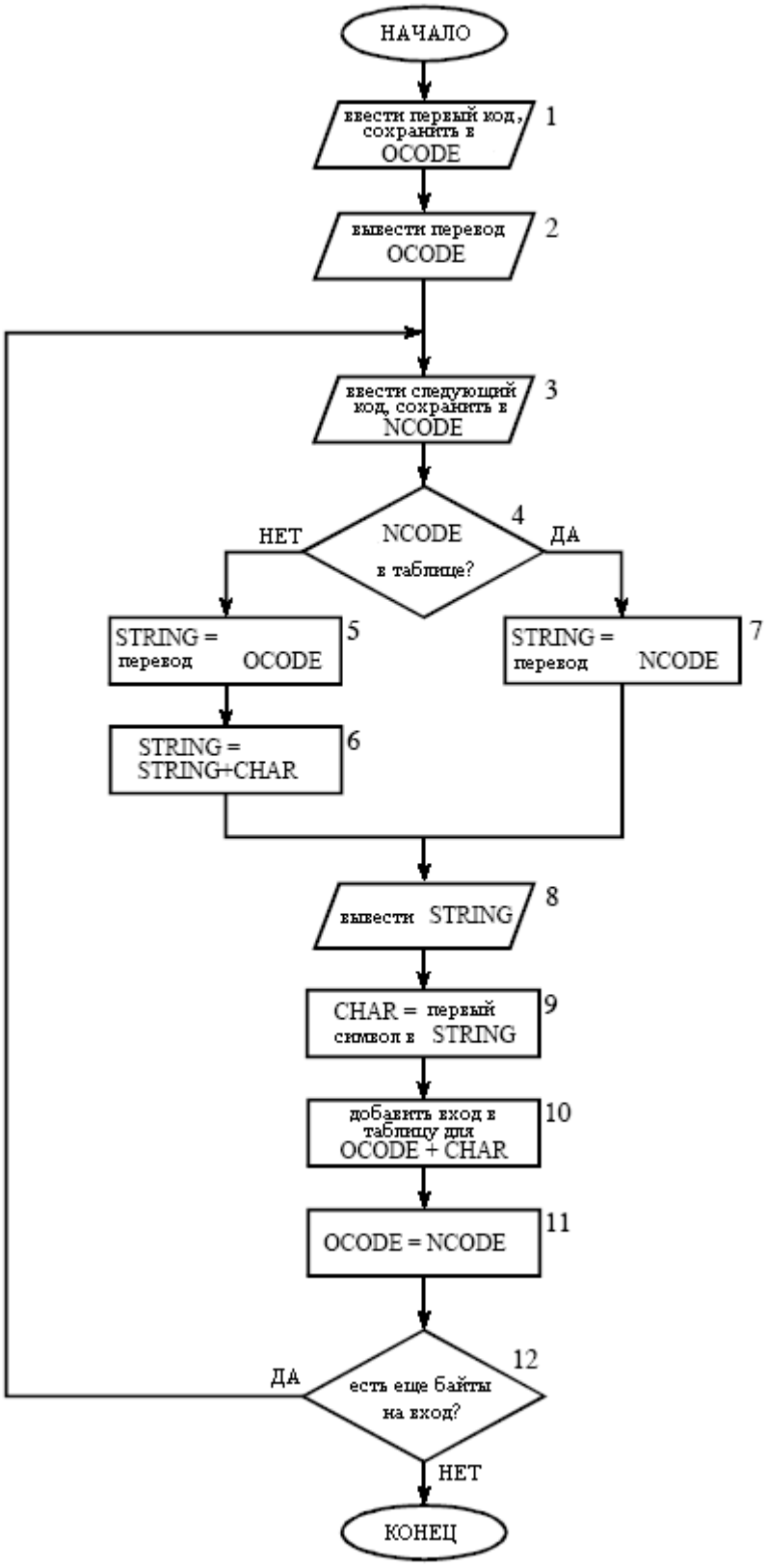


Рис. 27.8 Блок схема алгоритма осуществляющего операцию обратную LZW сжатию

Это приводит к последнему комментарию относительно LZW и подобных схем сжатия: *данная область – область большой конкуренции*. В то время как основы сжатия данных относительно просты, виды программ, продаваемых, как коммерческие продукты являются чрезвычайно сложными. Продавая Вам программы осуществляющие сжатие,

компании делают деньги и ревностно защищают свои профессиональные секреты при помощи патентов и т.п. Не надейтесь всего за несколько часов работы достичь такого же уровня качества, как и у этих программ.

JPEG (сжатие преобразованием)

Разработано большое число методов сжатия с потерями; однако, самым ценным оказалось семейство методов, называемых *сжатие преобразованием*. Наилучшим примером сжатия преобразованием является сжатие преобразованием, воплощенное в популярном стандарте кодирования изображения JPEG. JPEG назван по своему происхождению из *Joint Photographers Experts Group* (объединенной группы экспертов фотографов). Для иллюстрации того, как работает сжатие с потерями, Мы приведем описание функционирования JPEG.

Мы уже обсудили простой метод сжатия данных с потерями - *грубой дискретизации и/или квантования* (CS&Q в таблице 27.1). Он включает сокращение числа битов в отсчете или полный отказ от некоторых из отсчетов. Обе эти процедуры обладают желаемым эффектом: за счет качества сигнала файл данных становится меньше. Как Вы возможно и предполагали, такие простые методы работают не очень хорошо.

Сжатие преобразованием основывается на простой предпосылке: когда сигнал проходит через преобразование Фурье (или другое преобразование), величины получающихся в результате данных, больше не будут одинаковы по своей роли нести информацию. В частности низкочастотные составляющие сигнала являются более важными, чем высокочастотные составляющие. Скажем, удаление 50% битов от высокочастотных составляющих может удалить всего 5 % закодированной информации.

Как показано на рис. 27.9, JPEG сжатие начинается с разбивки изображения на группы 8×8 пикселей. В полном алгоритме JPEG на один пиксель может приходиться широкий диапазон битов, включая биты, использующие информацию о цвете. В данном примере каждый пиксель представляет собой отдельный байт - величину градации серого между 0 и 255. Эти 8×8 группы пикселей во время сжатия обрабатываются независимо. То есть каждая группа первоначально представляется 64 байтами. После выполнения преобразования и удаления данных, каждая группа представляется с помощью, скажем, от 2 до 20 байт. Во время осуществления операции обратной сжатию для создания аппроксимации исходной группы 8×8 выполняется обратное преобразование этих от 2 до 20 байт. Затем эти аппроксимированные группы собираются вместе для формирования несжатого изображения. Почему используются группы 8×8 пикселей вместо, например, 16×16? Группирование 8×8 базировалось на максимальном размере, который могли обрабатывать интегральные микросхемы, выполненные по технологии того времени, когда был разработан стандарт. В любом случае, размер 8×8 работает хорошо, и в будущем он может быть, а может и не быть изменен.

Для сжатия данных были исследованы многие различные преобразования, некоторые из них специально изобретены для этой цели. Например, преобразование *Кархунена-Лойва* дает наилучший из возможных коэффициент сжатия, но трудно осуществимо. *Преобразование Фурье* просто в использовании, но не обеспечивает адекватного сжатия. После большого числа состязаний, победителем стало родственное преобразованию Фурье, **дискретное косинусное преобразование (ДКП)**.

Так же, как преобразование Фурье для представления сигнала использует синусные и косинусные волны, ДКП использует только косинусные волны. Существует несколько версий ДКП, слегка отличающихся по своей математике. В качестве примера одной из версий, представьте 129 точечный сигнал, изменяющийся от отсчета 0 до отсчета 128. Теперь, дублированием отсчетов с 1 по 127 и добавлением их, как отсчетов с 255 по 129, сделайте его сигналом из 256 точек. То есть: 0, 1, 2, …, 127, 128, 127, …, 2, 1. Взятие преобразования Фурье от этого 256 точечного сигнала в результате дает частотный спектр

из 129 точек, простирающийся между 0 и 128. Поскольку сигнал временной области принудительно был сделан симметричным, мнимая часть спектра будет состоять из одних нулей. Другими словами, мы начали с сигнала временной области в 129 точек, и закончили частотным спектром в 129 точек с амплитудой косинусной волны каждая. Вуаля - ДКП!

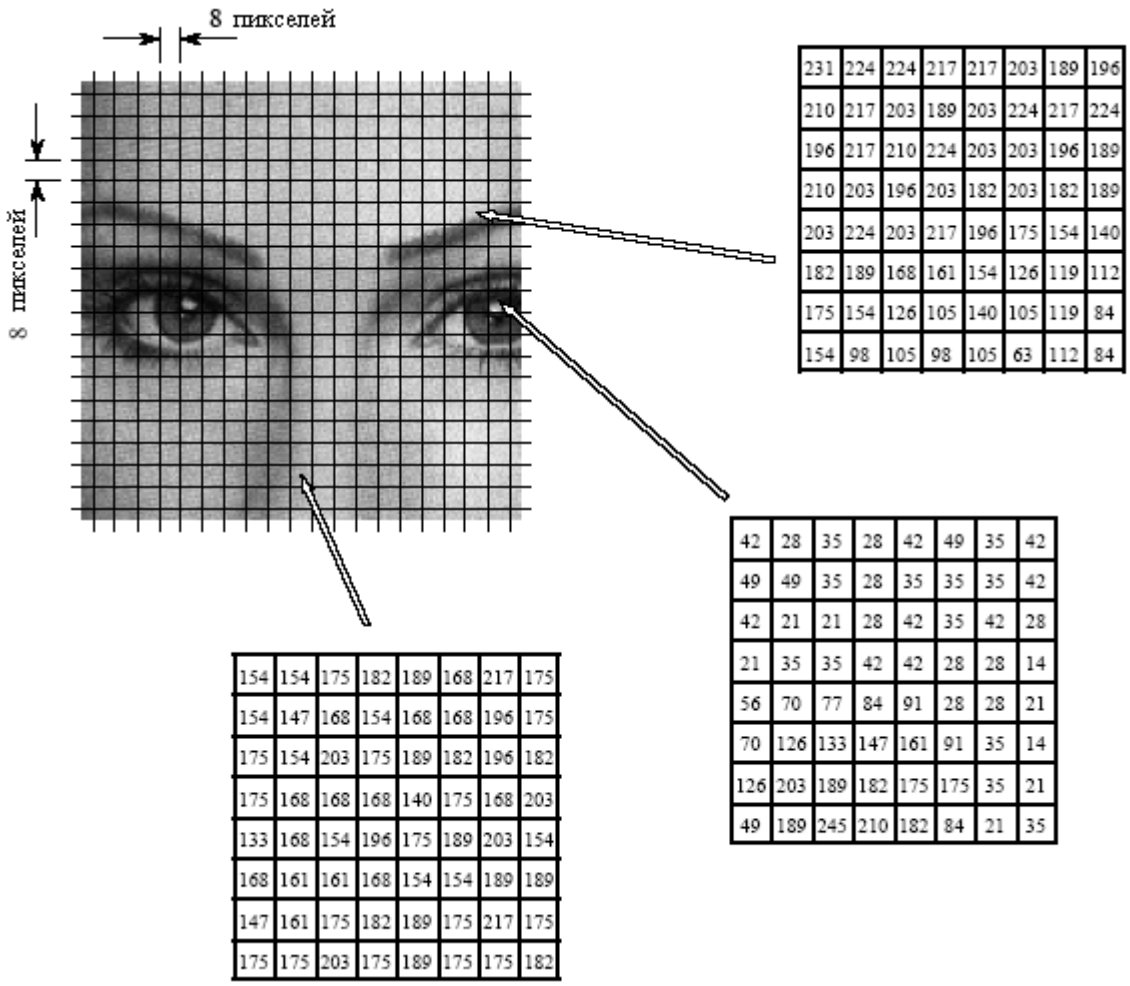


Рис. 27.9 Разбивка изображения в JPEG

Когда ДКП берется от группы 8×8, результатом является спектр 8×8. Иными словами, 64 числа заменяются другими 64 числами. Все эти значения являются *действительными*; здесь нет комплексной математики. Так же, как и в анализе Фурье, каждое значение в спектре является амплитудой **базисной функции**. На рис. 27.10, в соответствии с тем, где расположена в спектре амплитуда, показаны 6 из 64 базисных функций, используемых в ДКП 8×8. Базисные функции ДКП 8×8 задаются как:

$$b[x,y] = \cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right], \qquad (27.1)$$

где x и y – индексы в пространственной области; u и v – индексы в частотном спектре.

Низкие частоты находятся в верхнем левом углу спектра, а высокие частоты находятся в нижем правом углу. Постоянная составляющая находится в [0, 0] - наибольшее верхнее левое значение. Базисной функцией для [0, 1] в одном направлении

является полупериод косинусной волны, а в другом постоянная величина. Базисная функция для [1, 0] такая же, но повернута на 90°.

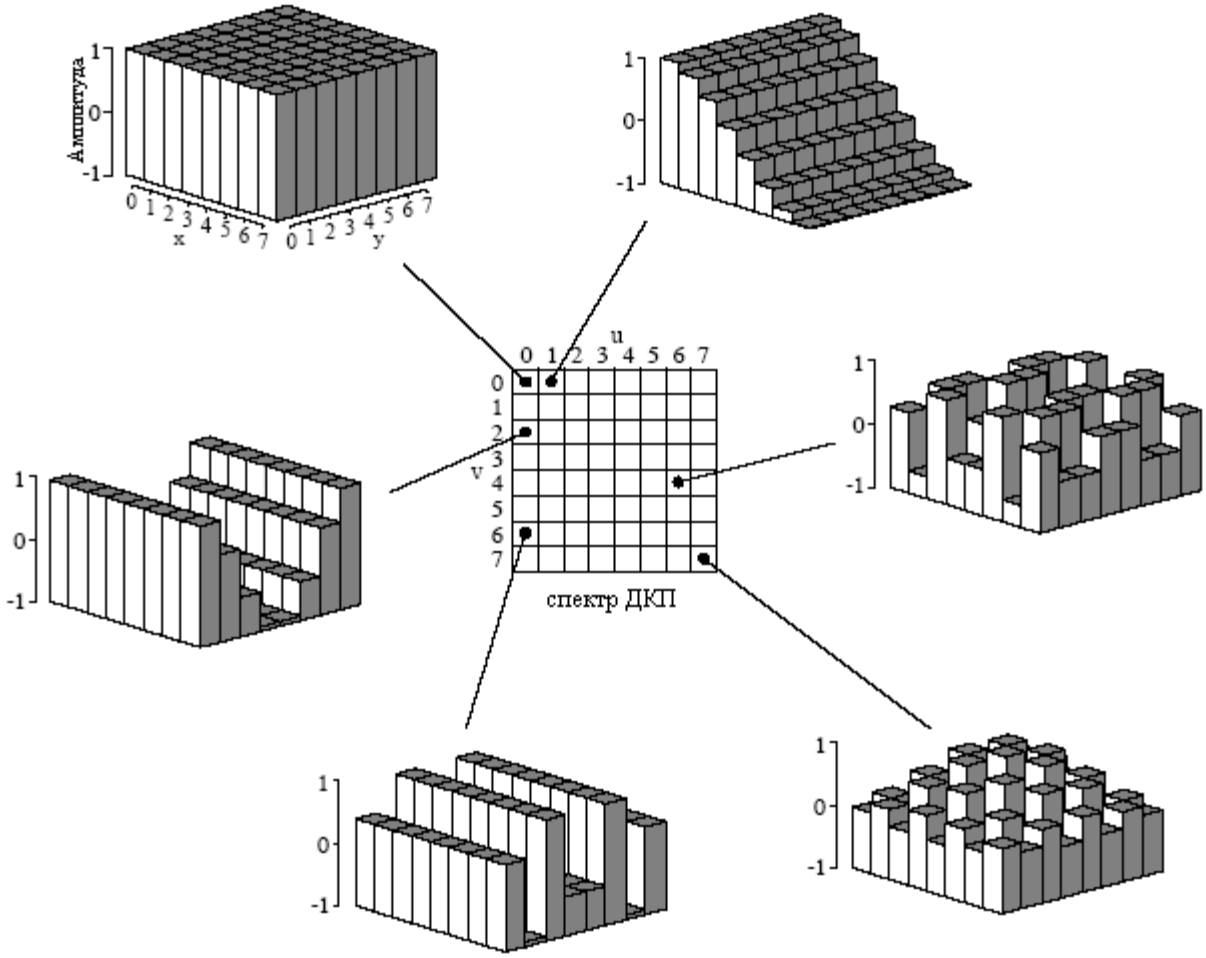


Рис. 27.10 Базисные функции ДКП

ДКП определяет спектр при помощи вычисления *корреляции* группы 8×8 пикселей с каждой из базисных функций. То есть, каждое значение спектра находится умножением соответствующей базисной функции на группу 8×8 пикселей, а затем сложением результатов произведений. Затем, для того чтобы закончить вычисление ДКП (так же, как и в случае с преобразованием Фурье), требуются две подгонки. Во-первых, разделите 15 значений спектра в строке 0 и столбце 0 на *два*. Во-вторых, разделите все 64 значения в спектре на 16. Обратное ДКП вычисляется дописыванием к каждой из амплитуд спектра соответствующей базисной функции и для восстановления пространственной области, их последующим суммированием. Никаких дополнительных шагов не требуется. Это точно такая же концепция, как и в анализе Фурье только с другими базисными функциями.

JPEG кодирование для трех групп 8×8, определенных на рис. 27.9, иллюстрируется на рис. 27.11. Левый столбец, рис. 27.11a, рис. 27.11b и рис. 27.11c, показывает исходные значения пикселей. Центральный столбец, рис. 27.11d, рис. 27.11e и рис. 27.11f, показывает спектры ДКП этих групп. Правый столбец, рис.27.11g, рис.27.11h и рис.27.11i, показывает эффект от уменьшения числа битов используемых для представления каждой составляющей в частотном спектре. Например, (g) формируется усечением каждого из отсчетов в (d) до десяти битов, взятием обратного ДКП, а затем вычитанием восстановленного изображения из оригинала. Аналогично, (h) и (i) сформированы усечением каждого отсчета в спектре до восьми и пяти битов, соответственно. Как и

предполагалось, по мере уменьшения числа битов, используемых для представления данных, ошибка в восстановлении увеличивается. В качестве примера такого усечения битов, спектры, показанные в центральном столбце, представлены на одно спектральное значение 8 битами, изменяющимися от 0 до 255 для постоянной составляющей и от -127 до 127 для других величин.

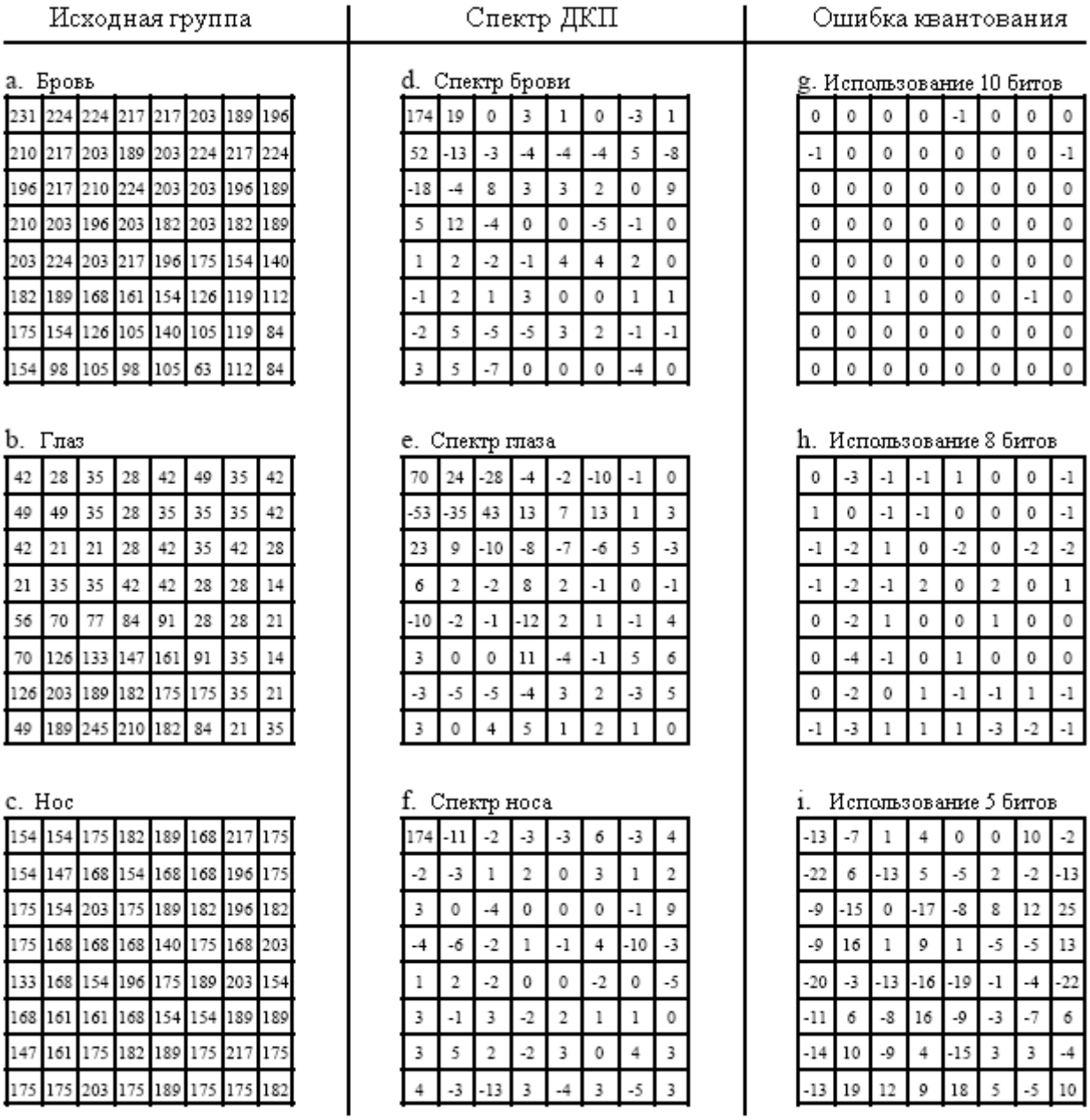
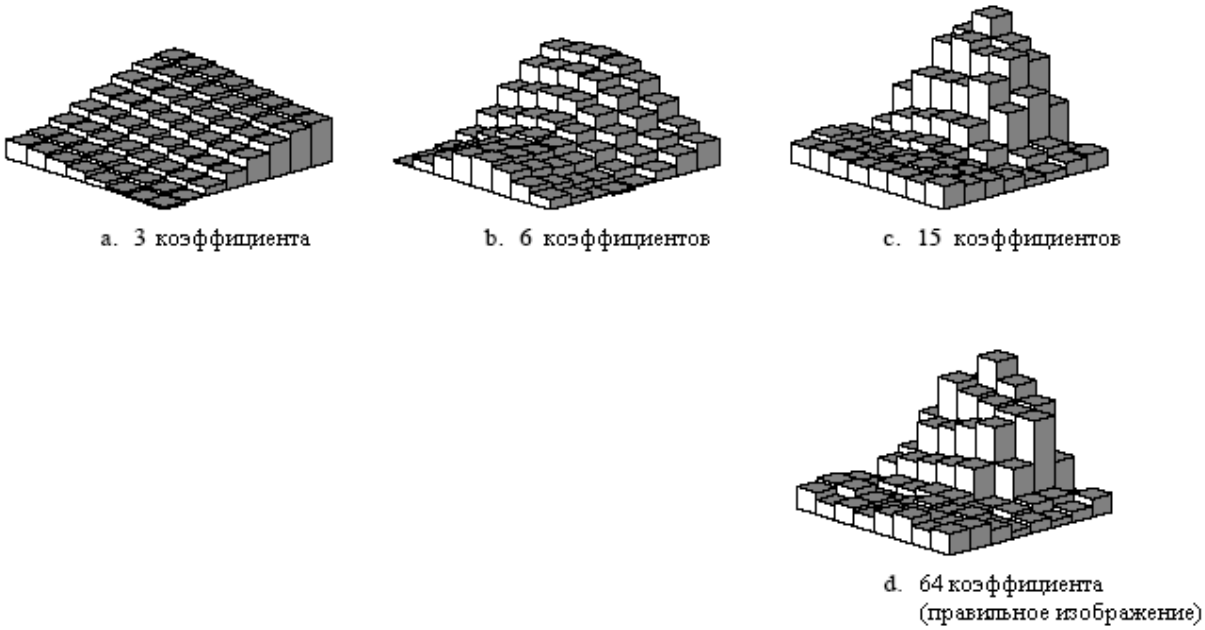


Рис. 27.11 Пример JPEG кодирования

Второй метод сжатия частотной области состоит в том, чтобы отказаться от некоторых из 64 значений спектра. Как показано спектрами на рис. 27.11, почти весь сигнал содержится в низкочастотных составляющих. Это означает, что составляющие высших частот могут быть удалены, приводя всего к небольшому ухудшению сигнала. На рис. 27.12 показан пример искажения изображения, появляющегося при удалении различного числа составляющих высоких частот. Используемая в этом примере группа 8×8 представляет собой изображение *глаза* с рис. 27.9. На рис. 27.12d показано точное восстановление, использующее все 64 спектральных значения. Оставшиеся рисунки показывают восстановление с использованием указанного на них числа составляющих

низших частот. Как показано на рис. 27.12c, даже удаление трех четвертей составляющих высших частот, при восстановлении дает маленькую ошибку. Даже лучше, появляющаяся ошибка, выглядит очень похожей на случайный шум.



a. 3 коэффициента

b. 6 коэффициентов

c. 15 коэффициентов

d. 64 коэффициента (правильное изображение)

Рис.27.12 Пример JPEG восстановления

a. Низкое сжатие

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

b. Высокое сжатие

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

Рис.27.13 JPEG таблица квантования

JPEG является хорошим примером тому, как для достижения большей эффективности, могут быть объединены несколько схем сжатия данных. Вся процедура JPEG в общих чертах заключается следующих шагах. Во-первых, изображение разбивается на группы 8×8. Во-вторых, от каждой группы берется ДКП. В третьих, каждый спектр 8×8 сжимается при помощи вышеупомянутых методов: уменьшение числа битов и удаление некоторых составляющих. Это происходит на отдельном этапе, управляемом **таблицей квантования**. На рис. 27.13 показаны два примера таблицы квантования. Каждое значение в спектре делится на соответствующую величину в таблице квантования и результат округляется до ближайшей целой величины. Например, значением в верхнем левом углу таблицы квантования является *единица*, в результате значение постоянной составляющей остается без изменений. Для сравнения, в нижнем правом углу на рис. 27.13a находится 16, что означает, что диапазон исходных значений от –127 до 127 уменьшается до величины всего от –7 до 7. Другими словами, по точности

вычислений значение было сокращено от восьми битов до четырех битов. В наиболее экстремальном случае, как на рис. 27.13b, в правом нижнем углу содержится 256, полностью устраняющее спектральное значение.

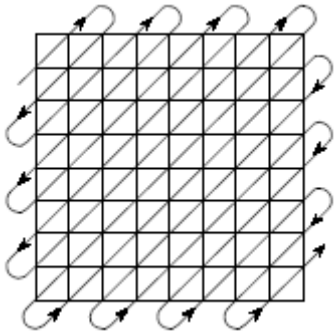


Рис. 27.14 Последовательность преобразования в JPEG

В четвертом шаге JPEG кодирования модифицированный спектр преобразуется из массива 8×8 в линейную последовательность. На этом шаге используется образец серпантина, показанный на рис. 27.14, размещающий все составляющие высоких частот вместе в конце линейной последовательности. Это группирует *нули* от удаленных составляющих в длинные последовательности. На пятом шаге с помощью кодирования длины повторов осуществляется сжатие этих последовательностей нулей. На шестом шаге, для того чтобы сформировать окончательно сжатый файл, последовательность кодируется с помощью кодирования Хаффмана или арифметического кодирования.

Величина сжатия и результирующие потери качества изображения, могут быть выбраны в процессе работы JPEG программы. На рис. 27.15 показан вид искажений изображения, возникающих из-за высоких коэффициентов сжатия. При показанном коэффициенте сжатия 45:1, каждая из групп 8×8 представляется с помощью всего около 12 битов. Близкое рассматривание этого изображения показывает, что до некоторой степени здесь представлены шесть базовых функций низших частот.

Почему для сжатия изображений ДКП лучше, чем преобразование Фурье? Главная причина состоит в том, что ДКП обладает полупериодными базисными функциями, т.е. S[0, 1] и S[1, 0]. Как показано на рис. 27.10, они изменяются постепенно от одной стороны массива к другой. Для сравнения, низкие частоты в преобразовании Фурье формируют *один полный период*. Изображения почти всегда содержат области, в которых яркость по области постепенно меняется. Использование базисной функции, соответствующей такому основному образцу, обеспечивает лучшее сжатие.

MPEG

MPEG представляет собой стандарт сжатия для цифровых видео последовательностей наподобие используемых в компьютерном видео и цифровых телевизионных сетях. Кроме того, MPEG обеспечивает также сжатие связанной с видео звуковой дорожки. MPEG назван по своему происхождению из *Moving Pictures Experts Group* (группа экспертов движущихся картинок). Если Вы считаете, что JPEG является сложным, то MPEG представляет собой ночной кошмар! MPEG - это что-то, что Вы покупаете, а не пытаетесь написать самостоятельно. Будущим этой технологии является установка алгоритма сжатия и алгоритма, осуществляющего операцию обратную сжатию непосредственно в интегральные микросхемы. Потенциал MPEG обширен. Подумайте о тысячах видеоканалов, передаваемых по одному оптическому волокну идущему в Ваш дом. Это - ключевая технология 21-ого столетия.

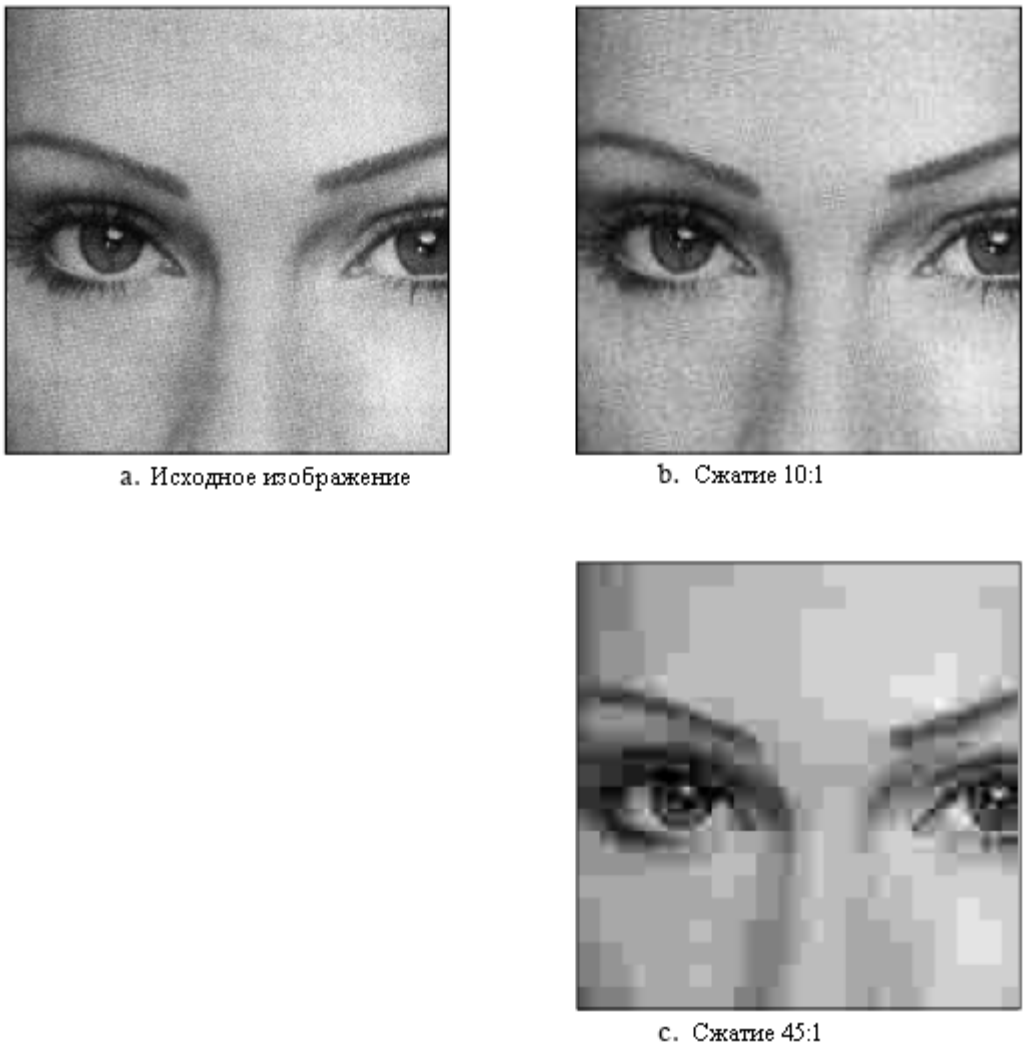


Рис. 27.15 Пример искажения в JPEG

В дополнение к сокращению интенсивности передачи данных, MPEG обладает несколькими важными особенностями. Кинофильм может быть воспроизведен, как в *прямом*, так и в *обратном* направлении, и как на *нормальной*, так и на *быстрой* скорости. К закодированной информации осуществляется *произвольный доступ*, т.е. любой индивидуальный кадр в последовательности может быть легко показан в виде неподвижной картинки. Одновременно с этим фильм делается *редактируемым*, что означает, что короткие фрагменты из фильма могут кодироваться со ссылкой на самих себя, а не на всю последовательность. MPEG разработан таким образом, чтобы быть устойчивым к ошибкам. Чтобы ошибка в один бит вызвала разрушение всего фильма – это то, чего Вы хотите в последнюю очередь.

Подход, использованный в MPEG, может быть разделен на два вида сжатия: *в пределах кадра* и *между кадрами*. Сжатие в пределах кадра означает, что индивидуальные кадры, составляющие видеопоследовательность кодируются так, как если бы они были обычными неподвижными изображениями. Такое сжатие выполняется с использованием стандарта JPEG только с несколькими изменениями. В терминологии MPEG кадр, который был закодирован таким способом, называется интра - кодированным или **I-картинкой**.

В видеопоследовательности большинство пикселей от одного кадра к следующему отличаются очень слабо. Если камера не перемещается, большая часть изображения состоит из фона, который остается постоянным более чем в дюжине кадров. MPEG тонким образом использует это преимущество для сжатия избыточной информации

между кадрами с помощью *дельта кодирования*. После сжатия одного из кадров до I-картинки MPEG кодирует следующие кадры, как предварительно кодированные или **P-картинки**. То есть в P-картинку включаются только те пиксели, которые претерпели изменения с тех пор, как были включены в I-картинку.

Тогда как эти две схемы сжатия образуют основу MPEG, фактическая его реализация является значительно более сложной, чем здесь описанная. Например, P-картинка может ссылаться на I-картинку, которая из-за движения объектов, уже была смещена в последовательности изображений. Существует также двунаправленное предварительное кодирование или **B-картинки**. Здесь ссылки имеются как на предыдущие, так и на последующие I-картинки. Это позволяет обрабатывать постепенно изменяющиеся на протяжении большого числа кадров области изображений. В сжатых данных для поддержки надлежащей последовательности I, P, и B-картинок в нарушение порядка могут быть также сохранены отдельные кадры. Добавление цвета и звука делает все это еще более сложным.

Основные искажения, связанные с MPEG, происходят тогда, когда большие куски изображения быстро меняются. В действительности, чтобы не отставать от быстро изменяющихся сцен, необходим взрыв информации. Если скорость передачи данных фиксирована, то при переходе от одной сцены к другой зритель замечает "заблокированные" образцы. Последнее может быть минимизировано в сетях подобных кабельному телевидению, передающих большое число видеоканалов одновременно. Внезапный взрыв информации, необходимый для поддержания быстро меняющейся сцены в одном видеоканале, усредняется скромными требованиями статических сцен в других каналах.

ГЛАВА

28

ЦИФРОВЫЕ ПРОЦЕССОРЫ ОБРАБОТКИ СИГНАЛОВ

Цифровая Обработка Сигнала выполняется с помощью математических операций. Для сравнения, обработка текстов и подобные программы просто осуществляют реорганизацию хранимых данных. Это означает, что компьютеры, разработанные для бизнеса и других общих приложений, не оптимизированы для алгоритмов подобных цифровой фильтрации и Фурье анализу. *Цифровые процессоры обработки сигналов* (ЦПОС) представляют собой микропроцессоры, специально разработанные для решения задач цифровой обработки сигналов. За последнее десятилетие эти устройства получили широкое распространение, они, находят применение практически во всем, от сотовых телефонов до передовых научных приборов. Фактически, инженеры-аппаратчики используют обозначение "ЦПОС" в значении *Цифровой процессор обработки сигналов* (часто используется термин "*сигнальный процессор*" – прим. перев.), а разработчики алгоритма используют обозначение "ЦОС" в значении *Цифровая обработка сигналов*. В этой главе рассматривается то, чем ЦПОС отличаются от других типов микропроцессоров, как решить, хорош ли ЦПОС для Ваших приложений и с чего начать в этой захватывающей новой области. В следующей главе мы более детально посмотрим на один из этих сложных продуктов: семейство Analog Devices SHARC®.

Чем отличаются ЦПОС от других микропроцессоров

В 1960-х было предсказано, что искусственный интеллект революционизирует способ общения людей с компьютерами и другими машинами. Полагалось, что к концу столетия у нас будут роботы, убирающие наши дома, компьютеры, управляющие нашими автомобилями, и голосовые интерфейсы, управляющие хранением и поиском информации. Этого не случилось; эти абстрактные задачи оказались более сложными, чем ожидалось, и трудновыполнимыми для пошаговой логики, которую обеспечивают цифровые компьютеры.

Однако, последние 40 лет показали, что компьютеры чрезвычайно способны в двух широких областях (1) **манипулирования данными,** вроде обработки текстов и управления базами данных, и (2) **математических вычислений**, используемых в науке, технике и цифровой обработке сигналов. Все микропроцессоры могут выполнять обе задачи; однако, очень трудно (дорого) сделать прибор *оптимальный* для них обеих. В проектировании аппаратной части существуют технические соглашения такие, как размер набора команд и как осуществлять управление прерываниями. Что еще более важно, сюда включаются еще и *торговые* вопросы: стоимость разработки и производства, конкурентоспособность, срок службы изделия и т.д. Проводя широкое обобщение, эти факторы и создали традиционный микропроцессор такой, как Pentium®, прежде всего направленный на манипуляцию данными. Подобным образом, ЦПОС разработаны для выполнения математических вычислений, необходимых в цифровой обработке сигнала.

На рис. 28.1 приведен перечень наиболее важных различий между этими двумя категориями. Манипулирование данными включает хранение и сортировку информации.

Например, рассмотрим программу обработки текстов. Основной задачей здесь является хранение информации (набранной оператором), организация информации (копирование и вставка, проверка правописания, разметка страницы и т.д.) и затем восстановление информации (вроде сохранения документа на дискете или печати его на лазерном принтере). Такие задачи выполняются *перемещением* данных из одного места в другое и *проверкой* неэквивалентности ($A=B$, $A<B$ и т.д.). В качестве примера представьте сортировку списка слов в алфавитном порядке. Каждое слово представлено числом из 8 битов, значением символа ASCII (американский стандартный код для обмена информацией – прим. перев.) первой буквы в слове. Расположение по алфавиту означает реорганизацию порядка слов так, что значение кода ASCII будет непрерывно увеличиваться от начала к концу списка. Это может быть выполнено повторением снова и снова двух шагов до тех пор, пока расстановка по алфавиту не завершиться. Во-первых, проверьте две соседние записи, расположены ли они в алфавитном порядке (IF A>B THEN ...). Во-вторых, если две записи не в алфавитном порядке, переставьте их так, чтобы они были в алфавитном порядке (A=B). Когда этот двух шаговый процесс повторится многократно над всеми соседними парами, список в конечном счете будет расположен в алфавитном порядке.

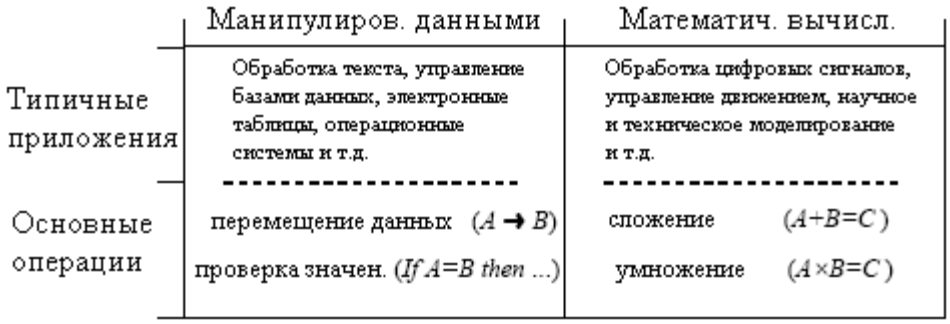


Рис. 28.1 Манипулирование данными против математических вычислений

В качестве другого примера, рассмотрим, как печатается документ из текстового процессора. Компьютер постоянно проверяет входное устройство (мышь или клавиатуру) на наличие двоичного кода указывающего “напечатать документ”. Когда такой код обнаружен, программа перемещает данные из памяти компьютера на печатающее устройство. Здесь у нас те же самые две основные операции: перемещение данных и проверка неэквивалентности. Хотя в этом виде приложения иногда используется математика, это происходит нечасто и не оказывает значительного влияния на общую производительность.

Для сравнения, производительность большинства алгоритмов ЦОС почти полностью ограничивается количеством необходимых умножений и сложений. Например, на рис. 28.2 показан самый обычный технический прием ЦОС, реализация цифрового КИХ-фильтра. Используя стандартную систему обозначений, входной сигнал обозначается здесь как $x[\ ]$, в то время как выходной сигнал обозначен как $y[\ ]$. Наша задача вычислить в выходном сигнале отсчет с номером n, т.е. $y[n]$. КИХ-фильтр выполняет эти вычисления при помощи умножения соответствующих отсчетов входного сигнала на группу коэффициентов обозначенных как: a_0, a_1, a_2, a_3,…, а затем сложения результатов произведений. В виде уравнения $y[n]$ находится как:

$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + a_3x[n-3] + a_4x[n-4] + \ldots.$$

Проще сказать, что осуществляется *свертка* входного сигнала с ядром фильтра (т.е. импульсным откликом), состоящим из: a_0, a_1, a_2, a_3,…. В зависимости от приложения в ядре фильтра может быть от нескольких до нескольких тысяч коэффициентов. Хотя в этом алгоритме есть и передача данных, и оценка неэквивалентности, такие как отслеживание промежуточных результатов и управление циклами, математические операции во времени исполнения доминируют.

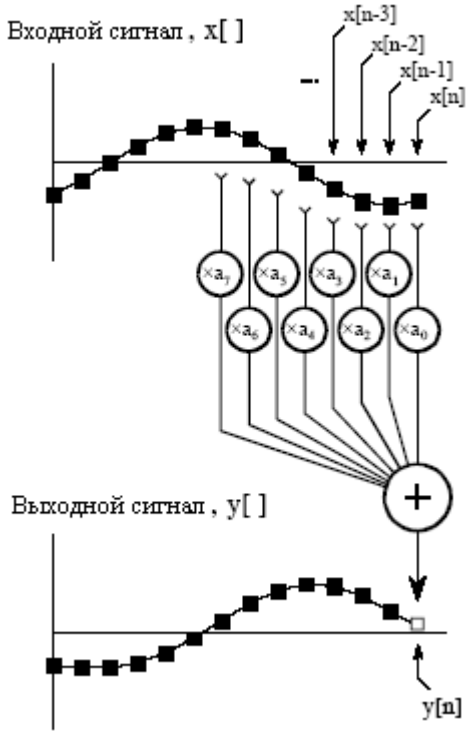


Рис. 28.2 Цифровой КИХ фильтр

В дополнение к очень быстрому исполнению математических операций, ЦПОС должен также обладать *предсказуемым* временем исполнения. Предположим, что Вы запускаете некоторую задачу на Вашем настольном компьютере, скажем, преобразование обработанного текстового документа из одной формы в другую. Не имеет никакого значения, займет ли обработка десять *миллисекунд* или десять *секунд*; Вы просто ждете, когда работа будет завершена, чтобы дать компьютеру следующее задание.

Для сравнения, большинство ЦПОС используются в приложениях, где обработка происходит *непрерывно*, не имеет определенного начала или конца. Например, рассмотрим инженера, проектирующего систему ЦОС для звуковых сигналов, типа слухового аппарата. Если звуковой сигнал поступает со скоростью 20000 отсчетов в секунду, ЦПОС должен быть способен сохранять устойчивую пропускную способность 20000 отсчетов в секунду. Однако существуют важные причины не делать его быстрее, чем нужно. По мере увеличения скорости растут и *цена*, и *потребляемая мощность*, и *сложность проектирования* и т.д. Это делает достоверное знание времени исполнения критическими для выбора соответствующего прибора, а также алгоритмов, которые могут быть применены.

Кольцевая буферизация

Цифровые процессоры для обработки сигналов сконструированы для быстрой реализации КИХ-фильтров и подобной техники. Для того чтобы понять *аппаратную* часть, сначала мы должны понять *алгоритмы*. В этом разделе мы составим детальный

перечень шагов необходимых для реализации КИХ-фильтра. В следующем разделе мы посмотрим, как сконструированы ЦПОС для того, чтобы выполнить эти шаги настолько эффективно, насколько это возможно.

Для начала мы должны провести различие между **обработкой в автономном режиме** и **обработкой в режиме реального времени**. В автономном режиме обработки *весь* входной сигнал в один и тот же момент времени находится в компьютере. Например, геофизик может использовать сейсмометр для записи подвижки земной коры во время землетрясений. После того, как землетрясение закончилось, информация может быть прочитана компьютером и некоторым образом обработана. Другим примером автономного режима обработки являются медицинские изображения, такие как компьютерная томография и магниторезонансные изображения. Сбор данных происходит в то время, когда пациент находится внутри аппарата, а реконструкция изображения может быть перенесена на более позднее время. Ключевым моментом здесь является то, что обрабатывающей программе одновременно доступна *вся* информация. Это типично для научных исследований и инженерной практики, но не для потребительских товаров. Автономный режим обработки - это царство персональных компьютеров и универсальных ЭВМ.

При обработке в режиме реального времени выходной сигнал вырабатывается в то же самое время, когда осуществляется прием входного сигнала. Например, это нужно в телефонной связи, слуховых аппаратах и радиолокаторах. Такие приложения должны иметь немедленный доступ к информации, хотя на короткий промежуток она может иметь задержку. Например, при телефонном разговоре задержка в 10 миллисекунд не может быть обнаружена ни говорящим, ни слушающим. Аналогично, нет никакой разницы в том, если сигнал от радиолокатора задержится на несколько секунд до того, как будет показан оператору. Приложения реального времени вводят отсчет, выполняют алгоритм, и выводят отсчет и так, опять и опять. Альтернативно, они могут вводить группу отсчетов, выполнять алгоритм и выводить группу отсчетов. Это и есть мир цифровых процессоров обработки сигналов.

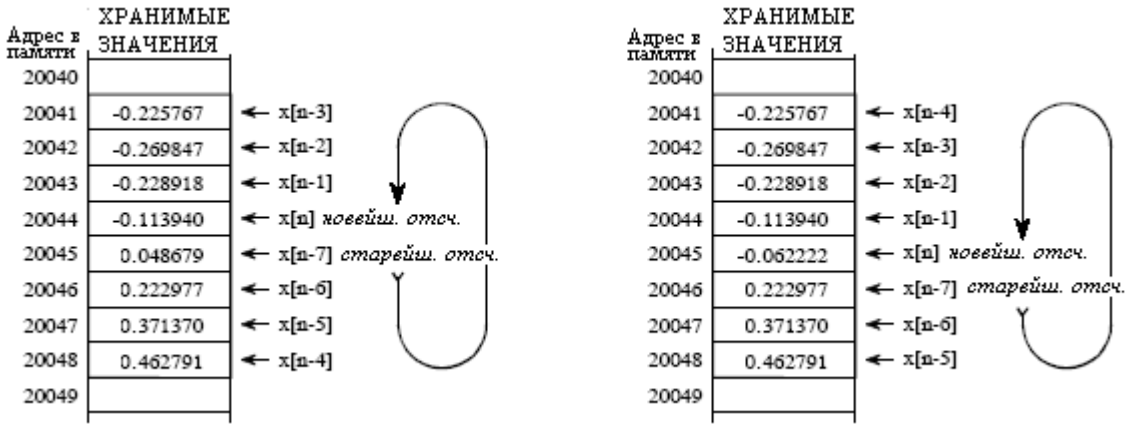


Рис. 28.3 Работа кольцевого буфера

Теперь вернемся назад к рис. 28.2 и представим, что это КИХ-фильтр, реализованный в *реальном времени*. Для вычисления выходного отсчета мы должны иметь доступ к некоторому числу самых недавних отсчетов с входа. Например, предположим, что мы используем в этом фильтре восемь коэффициентов a_0, a_1, …, a_7. Это означает, что мы должны знать значения восьми самых недавних отсчетов из входного сигнала $x[n]$, $x[n-1]$, …, $x[n-7]$. Эти восемь отсчетов должны быть сохранены в памяти, и непрерывно обновляться по мере поступления новых отсчетов. Какой способ управления этими

хранящимися отсчетами является наилучшим? Ответом будет - **кольцевая буферизация** (кольцевой стек – прим. перев.).

На рис. 28.3 показан кольцевой буфер на восемь отсчетов. Мы разместили этот кольцевой буфер в восьми последовательных ячейках памяти с 20041 по 20048. На рис. 28.3a показано, как восемь поступающих с входа отсчетов могли бы быть загружены в память в данный конкретный момент времени, а на рис. 28.3b показаны изменения, произошедшие после получения следующего отсчета. Идея кольцевой буферизации заключается в том, что конец данного линейного массива соединен с его началом; ячейка памяти с адресом 20041 рассматривается, как идущая следующей за ячейкой 20048 так же, как ячейка с адресом 20044, следующая за ячейкой 20045. Вы отслеживаете массив с помощью **указателя** (переменной, значение которой является *адресом*), показывающего, где находится самый недавний отсчет, полученный с входа. Например, на рис. 28.3a указатель содержит адрес 20044, в то время как на рис. 28.3b он содержит 20045. Когда происходит получение нового отсчета, он записывается на место самого старейшего отсчета в массиве, а содержимое указателя продвигается на один адрес вперед. Кольцевые буферы эффективны потому, что при получении нового отсчета нужно изменить всего одну величину.

Для того чтобы управлять кольцевым буфером, необходимы четыре параметра. Во-первых, должен быть указатель, показывающий начало кольцевого буфера в памяти (в данном примере 20041). Во-вторых, должен быть указатель, показывающий конец массива (т.е. 20048) или переменная, содержащая длину массива (т.е. 8). В-третьих, должен быть задан размер шага адресации памяти. На рис. 28.3 размер шага равен *единице*, например: адрес 20043 содержит один отсчет, адрес 20044 содержит другой отсчет и т.д. Но часто это не так. Например, адресация может относиться к байтам, а каждый отсчет для хранения своего значения может потребовать два или четыре байта. В этих случаях, размер шага соответственно будет два или четыре.

Эти три величины определяют размер и конфигурацию кольцевого буфера, и во время работы программы изменяться не будут. Четвертая величина, указатель самого недавнего отсчета, должна изменяться каждый раз, когда поступает новый отсчет. Другими словами, должна существовать программная логика, управляющая тем, как обновлять эту четвертую величину, основываясь на значениях первых трех величин. Хотя эта логика совсем проста, она должна иметь высокое быстродействие. Вот и весь вопрос этого обсуждения; ЦПОС должен быть оптимизирован на управление кольцевыми буферами для достижения наиболее высокой возможной скорости выполнения.

В качестве дополнения отметим, что кольцевая буферизация используется и при обработке в *автономном режиме*. Рассмотрим программу, в которой оба, входной и выходной сигналы, полностью содержатся в памяти. Поскольку к каждому отсчету имеется немедленный доступ, кольцевая буферизация для осуществления вычисления свертки не нужна. Тем не менее, большое число алгоритмов реализуются *поэтапно* и создают на каждом этапе промежуточные сигналы. Например, таким способом работает рекурсивный фильтр, выполненный в виде последовательности биквадратов. Методом решения в лоб является, сохранение в памяти каждого промежуточного сигнала на всю его длину. Кольцевая буферизация дает другую возможность: сохранение только тех промежуточных отсчетов, которые необходимы для вычислений в ближайшее время. Это снижает требуемый объем памяти за счет более сложного алгоритма. Важным является то, что кольцевые буферы *полезны* для обработки в автономном режиме и *крайне необходимы* для приложений работающих в реальном времени.

Теперь мы можем посмотреть на шаги, необходимые для реализации КИХ-фильтра с использованием кольцевых буферов и для входного сигнала, и для коэффициентов. Этот перечень может показаться тривиальным и более чем исследованными, но это не так! Эффективная обработка таких индивидуальных задач является тем, что отделяет ЦПОС от

традиционного микропроцессора. Для каждого нового отсчета должны быть выполнены все следующие шаги:

Таблица 28.1

1. Получить отсчет от АЦП; сгенерировать прерывание
2. Обнаружить и обработать прерывание
3. Поместить отсчет в кольцевой буфер входного сигнала
4. Обновить указатель кольцевого буфера входного сигнала
5. Обнулить аккумулятор
6. Управлять циклом по каждому из коэффициентов
7. Считать коэффициент из кольцевого буфера коэффициентов
8. Обновить указатель кольцевого буфера коэффициентов
9. Считать отсчет из кольцевого буфера входного сигнала
10. Обновить указатель кольцевого буфера входного сигнала
11. Перемножить коэффициент с отсчетом
12. Сложить результат с содержимым аккумулятора
13. Переместить выходной отсчет (содержимое аккумулятора) в буфер хранения
14. Переместите выходной отсчет в ЦАП

Цель заключается в том, чтобы сделать эти шаги быстро исполняющимися. Поскольку шаги с 6 по 12 многократно повторяются (один раз для каждого коэффициента фильтра), этим операциям должно быть уделено специальное внимание. Традиционные микропроцессоры, вообще должны выполнять эти 14 операций *последовательно* (одна за другой), в то время как ЦПОС спроектированы для выполнения их *параллельно*. В некоторых случаях все операции в цикле (шаги 6-12) могут быть выполнены *за один такт*. Давайте посмотрим на внутреннюю архитектуру, которая обеспечивает такую великолепную производительность.

Архитектура цифрового процессора обработки сигналов

Одним из наиболее узких мест в выполнении алгоритмов ЦОС является передача информации в память и из памяти. Сюда относятся *данные*, подобные отсчетам входного сигнала и коэффициентам фильтра, а также *команды программы* - двоичные коды, поступающие в программный секвенсер (Устройство, устанавливающее порядок выполнения команд программы. Секвенсер поддерживает условные переходы, вызовы подпрограмм и возврат в отдельный цикл. Благодаря внутренним счетчикам и стекам цикла, для его поддержания никаких определенных команд перехода не требуется. Прим. перев.). Например, предположим нам нужно перемножить два числа, которые находятся где-то в памяти. Для того чтобы это сделать мы должны считать из памяти три двоичных величины, числа, которые следует перемножить плюс команду программы, описывающую, что нужно сделать.

На рис. 28.4 показано, как эта, кажется простая задача, выполняется на традиционном микропроцессоре. Такая реализация часто называется **Неймановской архитектурой**, по имени блестящего Американского Математика Джона фон Неймана (1903-1957). Нейман был руководителем математических вопросов многих важных открытий на заре двадцатого столетия. К большому числу его достижений относятся: развитие концепции компьютера с хранимой внутри программой, формализация математики квантовой механики и работы по атомной бомбе. Если это было ново и возбуждающе, фон Нейман был там!

Как показано на рис. 28.4а, Неймановская архитектура состоит из одного блока памяти и одной шины для передачи данных в центральный процессор (ЦП) и из него.

Умножение двух чисел требует, по меньшей мере, трех тактовых периодов, по одному для каждого из трех чисел на передачу по шине из памяти в ЦП. Мы не учитываем время на передачу результата обратно в память потому, что мы предполагаем, что он остается в ЦП для дополнительной манипуляции (такой, как суммирование произведений в КИХ-фильтре). Если Вы собираетесь выполнять все требуемые задачи последовательно, то конструкция фон Неймана весьма удовлетворительна. Действительно, сегодня большинство компьютеров имеют конструкцию фон Неймана. Другая архитектура нужна нам тогда, когда требуются очень быстрая обработка, и мы готовы платить цену за увеличение сложности.

Это приводит нас к **Гарвардской архитектуре**, показанной на рис. 28.4b. Она названа по работе проделанной в Гарвардском университете в 1940-х под руководством Говорда Айкена (1900-1973). Как показано на этой иллюстрации, Айкен настоял на отдельных блоках памяти для данных и кодов команд с отдельными для каждой из них шинами. Поскольку шины работают независимо, коды команд и данные могут быть считаны в одно и тоже время, увеличивая скорость обмена по сравнению с конструкцией с одной шиной. Большинство сегодняшних ЦПОС используют такую двухшинную архитектуру.

На рис. 28.4c иллюстрируется следующий уровень сложности, **супер Гарвардская архитектура**. Этот термин был придуман фирмой Analog Devices для описания внутренней работы их ADSP-2106x и нового семейства цифровых процессоров обработки сигналов ADSP-211xx. Они называются ЦПОС **SHARC®**, сокращение от более длинного термина Super Harvard ARChitecture. Идея состоит в том, чтобы надстроить Гарвардскую архитектуру путем добавления характеристик, улучшающих пропускную способность. Несмотря на то, что ЦПОС SHARC оптимизированы дюжинами различных способов, две области являются достаточно важными, чтобы быть включенными в рис. 28.4c: *кэш команд* и *контроллер ввода/вывода*.

Во-первых, давайте посмотрим, как кэш команд улучшает производительность Гарвардской архитектуры. Недостатком базовой Гарвардской разработки является то, что шина памяти данных занята больше, чем шина памяти программ. Когда умножаются два числа, по шине памяти данных должны проследовать две двоичных величины (два числа), в то время как по шине памяти программ проследует только одна двоичная величина (код команды). Для улучшения этой ситуации начнем с перемещения части “данных в память программ”. Например, при хранении входного сигнала в памяти данных, мы можем переместить коэффициенты фильтра в память программ. (Эти перемещенные данные, на иллюстрации называются "вторичными данными".) На первый взгляд, кажется, это никак не помогает ситуации; теперь мы должны передать одно значение по шине памяти данных (входной отсчет сигнала), и два значения по шине памяти программ (команда программы и коэффициент). Фактически, если бы мы выполняли случайные команды, такая ситуация вообще была бы не лучше.

Однако алгоритмы ЦОС в основном тратят большую часть своего времени выполнения на циклы, подобные инструкциям с 6 по 12 в таблице 28.1. Это означает, что один и тот же набор команд программы будет непрерывно поступать от памяти программ к ЦП. В этой ситуации супер Гарвардская архитектура имеет преимущество за счет включения **кэша команд** в состав ЦП. Это небольшая память, которая содержит около 32 наиболее недавних команд программы. Команды программы должны пройти по шине памяти программ только при первом проходе через цикл. Это выражается в более медленной работе из-за конфликта с коэффициентами, которые также должны быть получены по этому пути. Однако, при последующих выполнениях цикла, коды команд могут браться из кэша команд. Это означает, что передача всей информации из памяти в ЦП может быть завершена за один тактовый период: отсчет входного сигнала проходит через шину памяти данных, коэффициент проходит через шину памяти программ и код команды приходит из кэша команд. На жаргоне этой области такая эффективная передача

данных называется *доступ к памяти с большой шириной диапазона* (*high memory-access bandwidth*). На рис. 28.5 представлен более детальный взгляд на архитектуру SHARC, показывающий присоединенный к памяти данных **контроллер ввода/вывода**. Это - путь, по которому сигналы входят в систему и выходят из нее. Например, ЦПОС SHARC снабжен последовательным и параллельным коммуникационными портами. Это чрезвычайно высокоскоростные соединения. В частности, при тактовой частоте 40 МГц имеются два последовательных порта, работающие при 40 Мбит/сек каждый, в то время как шесть параллельных портов обеспечивают передачу данных, при 40 Мбайт/сек каждый. Когда все шесть параллельных портов используются вместе, скорость передачи данных невероятна - 240 Мбайт/сек.

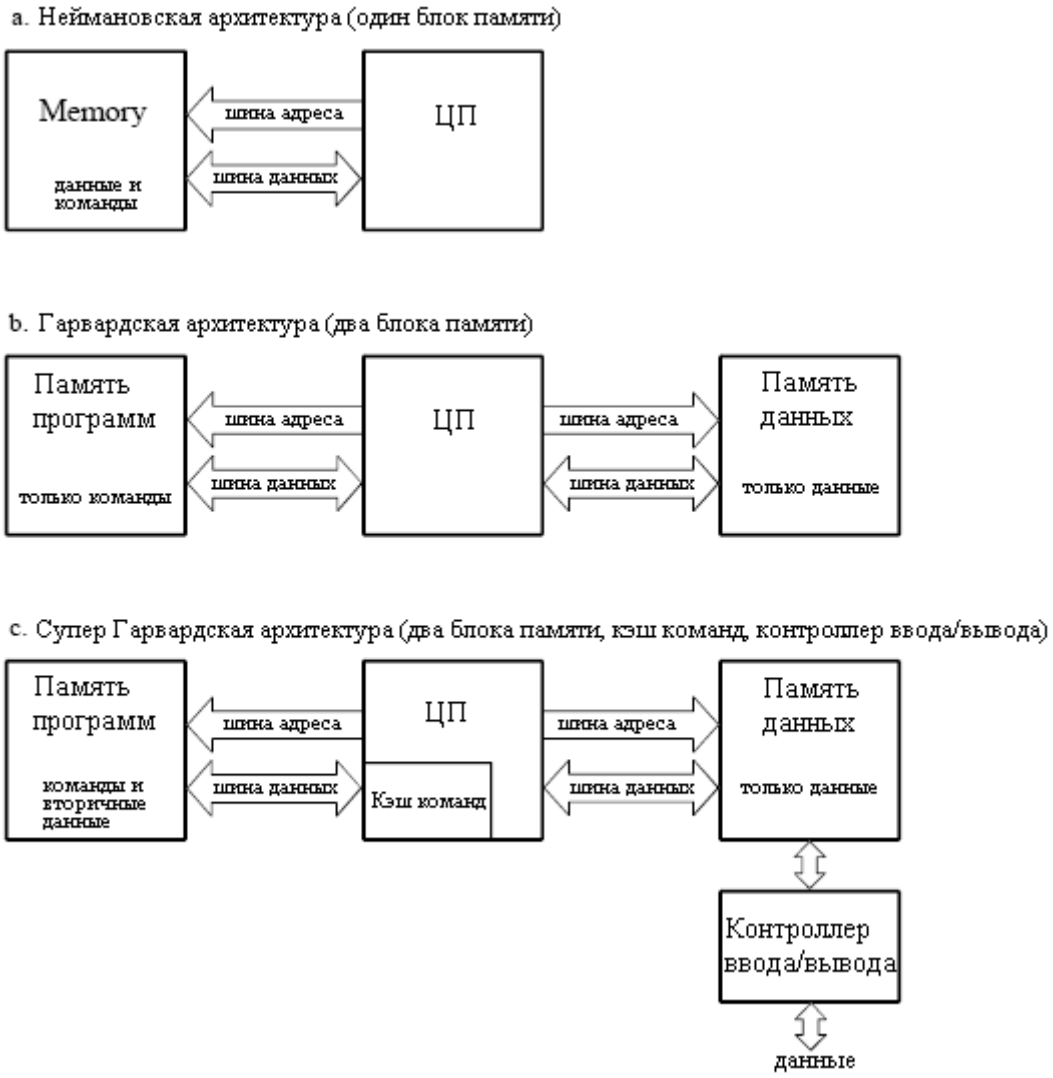


Рис. 28.4 Архитектура микропроцессора

Это достаточно быстро для того, чтобы передать весь текст этой книги всего за 2 миллисекунды! Более чем важно, что непосредственно подключенная к памяти аппаратура позволяет передавать такие потоки данных прямо в память (прямой доступ в память или ПДП), минуя прохождение через регистры ЦП. Другими словами, инструкции 1 и 14 из нашего получившегося списка независимы от других задач и одновременны с ними; ни одного тактового периода в ЦП не пропадает. Основные шины (шина памяти программ и шина памяти данных) доступны также со стороны внешних устройств, обеспечивая дополнительный интерфейс с внешней памятью и периферийными устройствами. Это позволяет ЦПОС SHARC использовать память объемом четыре

Гигаслова (16 Гбайт) со временем доступа 40 Мслов/сек (160 Мбайт/сек) при 32 разрядных данных. Вот это да!

Такой тип высокоскоростного ввода/вывода является ключевой характеристикой ЦПОС. Наиважнейшей целью является занесение данных, выполнение математических операций и вывод данных до того, как будет доступен следующий отсчет. Все остальное является вторичным. Некоторые ЦПОС содержат встроенные аналого-цифровые и цифро-аналоговые преобразователи, такая особенность называется **комбинированным сигналом**. Однако все ЦПОС могут быть подключены к внешним преобразователям через последовательный или параллельный порты.

Теперь давайте заглянем внутрь ЦП. В самом верху рисунка находятся два блока, по одному для каждых двух блоков памяти, отмеченные как **генератор адреса данных**. Они управляют адресами, посылаемыми в память программ и в память данных, определяя, откуда будет произведено считывание и куда будет произведена запись информации. В простых микропроцессорах с этой задачей обращались, как с частью весьма прозрачной для программиста и предназначенной программному секвенсеру. Однако ЦПОС спроектированы для работы с *кольцевыми буферами* и получения выгоды от дополнительного оборудования при эффективном его управлении. Это исключает необходимость использования драгоценных тактовых периодов ЦП на отслеживание того, как сохраняются данные. Например, в ЦПОС SHARC каждый из двух генераторов адреса данных может управлять *восьмью* кольцевыми буферами. Это означает, что каждый генератор адреса данных содержит 32 переменных (4 на буфер) плюс необходимая логика.

Почему так много кольцевых буферов? Некоторые алгоритмы ЦОС лучше всего выполняются поэтапно. Например, БИХ-фильтры более устойчивы, если реализуются в виде биквадратных каскадов (этап, содержащий два полюса и до двух нулей). Большое число каскадов требует, для более быстрого функционирования, большого числа кольцевых буферов. Генератор адреса данных ЦПОС SHARC спроектирован также для эффективного выполнения *быстрого преобразования Фурье*. В этом режиме генератор адреса данных конфигурируется для генерирования в кольцевом буфере необходимой части алгоритма БПФ, **адресов перевернутых битов**. Кроме того, обилие кольцевых буферов сильно упрощает генерацию кода ЦПОС, как для программиста, так и для компилятора языка высокого уровня, подобного ***C***.

Блок регистров данных ЦП используется таким же образом, как и в традиционных микропроцессорах. В ЦПОС SHARC типа ADSP-2106x - 16 регистров общего назначения по 40 битов каждый. Это позволяет проводить промежуточные вычисления, подготовку данных для математического сопроцессора, служить буфером передаваемых данных, хранить флаги для управления программой и т.д. Если необходимо эти регистры могут быть также использованы в качестве счетчиков для управления циклами; однако, для выполнения многих из этих функций в ЦПОС SHARC существуют дополнительные аппаратные регистры.

Обработка математики разбита на три секции **блок умножения**, **арифметико-логическое устройство** (**АЛУ**) и **кольцевой сдвиговый регистр**. Блок умножения берет из двух регистров два значения, перемножает их и результат помещает в другой регистр. АЛУ вычисляет сложение, вычитание, абсолютное значение, выполняет логические операции (И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, НЕ), осуществляет преобразование между форматами чисел с фиксированной и с плавающей запятой и подобные функции. Элементарные двоичные операции такие, как сдвиг, циклический сдвиг, извлечение и вставка сегмента, выполняются кольцевым сдвиговым регистром. Мощной характеристикой семейства SHARC является то, что к блоку умножения и АЛУ может осуществляться параллельный доступ. За один тактовый период данные из регистров 0-7 могут поступить на блок умножения, данные из регистров 8-15 могут поступить на АЛУ, а два результата вернуться в любой из 16 регистров.

Имеется также много важных характеристик семейства архитектуры SHARC, которые не показаны в этой упрощенной иллюстрации. Например, для снижения ошибки округления, ассоциирующейся с математической операцией умножения чисел с фиксированной запятой, в блок умножения встроен 80 битный аккумулятор. Другой интересной характеристикой является использование **теневых регистров** для всех ключевых регистров ЦП. Это дублирующие регистры, которые могут быть переключены на замену своих дублеров за один тактовый период. Они используются для *быстрого переключения контекста*, способности быстро обрабатывать прерывания. Когда происходит прерывание, то в традиционных микропроцессорах перед обслуживанием прерывания все внутренние данные должны быть сохранены. Это обычно заключается в занесении по очереди содержимого всех занятых регистров в стек. Для сравнения, прерывание в семействе SHARC обслуживается занесением внутренних данных в теневые регистры за *один тактовый период*. По завершении подпрограммы обслуживания прерывания, содержимое регистров восстанавливается также быстро. Эта характеристика позволяет очень быстро и эффективно обработать инструкции 0-4 (управление прерыванием отсчет готов) из нашего списка.

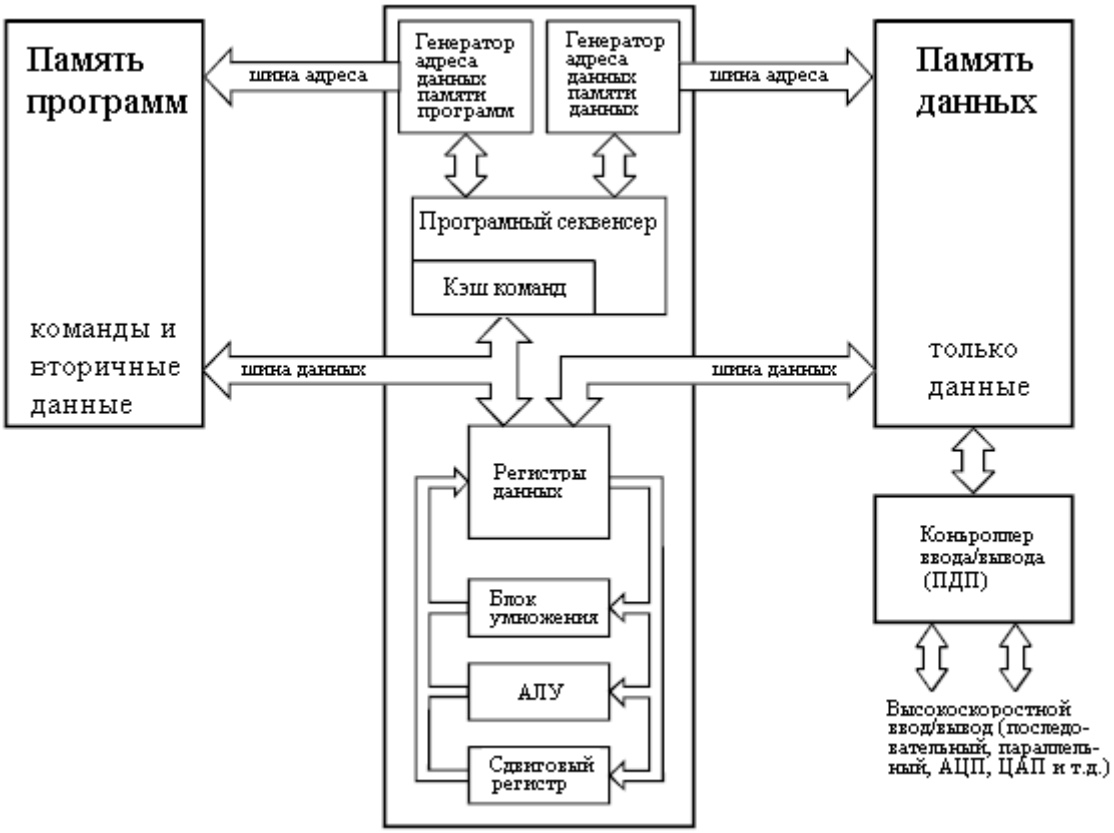


Рис. 28.5 Типичная архитектура ЦПОС

Теперь мы подходим к пределу работы архитектуры - какое количество операций в пределах цикла (шаги 6-12 в таблице 28.1) может быть выполнено одновременно. Благодаря своим высоким параллельным характеристикам ЦПОС SHARC может одновременно выполнять *все* эти операции. Определенно, за один тактовый период он может выполнить умножение (шаг 11), сложение (шаг 12), две пересылки данных (шаги 7 и 9), обновление двух указателей кольцевых буферов (шаги 8 и 10) и управление циклом (шаг 6). Понадобятся дополнительные тактовые периоды, связанные с началом и окончанием цикла (шаги 3, 4, 5 и 13, плюс загрузка на свои места начальных значений); однако, эти задачи также обрабатываются очень эффективно. Если цикл исполняется

более, чем несколько раз, такая добавка будет незначительной. В качестве примера, предположим, что Вы написали эффективную программу КИХ-фильтра, использующую 100 коэффициентов. Вы можете предположить, что она потребует приблизительно от 105 до 110 тактовых периодов на обработку отсчета (т.е. 100 циклов по коэффициентам плюс добавка). Это очень впечатляет; традиционному микропроцессору для такого алгоритма потребуется много тысяч тактовых периодов.

Фиксированная запятая по сравнению с плавающей

Цифровая обработка сигналов может быть разделена на две категории с **фиксированной запятой** и с **плавающей запятой**. Это относится к форматам, используемым для хранения и манипулирования числами внутри устройства. В ЦПОС с фиксированной запятой каждое число обычно представляется минимум с помощью 16 битов, хотя может использоваться и другая длина. Например, Motorola производит семейство ЦПОС с фиксированной запятой использующее 24 бита. Существует четыре обычных способа, которыми эти 2^{16}=65536 возможных комбинаций битов, могут представлять число. В **целом числе без знака** хранимое число может принимать любое целое значение от 0 до 65535. Подобно этому **целое число со знаком**, используя дополнение до двух, делает диапазон, включающий отрицательные числа, простирающимся от –32768 до 32767. При обозначении в виде **дроби без знака** 65536 уровней равномерно распределены между 0 и 1. Наконец, формат **дроби со знаком** позволяет отрицательным числам равномерно распределиться между -1 и 1.

Для сравнения, ЦПОС с плавающей запятой для хранения каждого значения используют минимум 32 бита. Это выражается в значительно большем числе комбинаций битов, чем для фиксированной запятой, если быть точным 2^{32}=4294967296. Ключевой чертой обозначения с плавающей запятой является то, что представляемые числа расположены *не* равномерно. В наиболее общем формате (ANSI/ стандарт IEEE 754-1985), самым большим и самым маленьким числами являются, соответственно, $\pm 3{,}4\times10^{38}$ и $\pm 1{,}2\times10^{-38}$. Представляемые значения распределены между этими двумя крайними значениями неравномерно, так что промежутки между любыми двумя числами приблизительно в десять миллионов раз меньше, чем значения чисел. Это важно, поскольку между большими числами устанавливаются большие промежутки, а малые промежутки устанавливаются между маленькими числами. Более подробно обозначение с плавающей запятой обсуждается в Главе 4.

Все ЦПОС с плавающей запятой могут обрабатывать также и числа с фиксированной запятой; настоятельная потребность при реализации счетчиков, циклов и сигналов, приходящих от АЦП и уходящих в ЦАП. Однако это не означает, что математика с фиксированной запятой будет выполняться также быстро, как математика с плавающей запятой; это зависит от внутренней архитектуры. Например, ЦПОС SHARC оптимизирован как для операций с плавающей запятой, так и для операций с фиксированной запятой и выполняет их с одинаковой эффективностью. По этой причине приборы SHARC часто относят к “32-битовым ЦПОС”, а не просто к “ЦПОС с плавающей запятой”.

На рис. 28.6 иллюстрируется основное соотношение между ЦПОС с фиксированной и плавающей запятой. В Главе 4 мы подчеркивали, что в компьютерах общего назначения арифметика с фиксированной запятой гораздо быстрее, чем арифметика с плавающей запятой. Однако для ЦПОС скорость приблизительно одинакова, что является результатом высоко оптимизированной под математические операции аппаратуры. Все регистры и шины данных вместо 16 разрядных должны быть 32 разрядными; блок умножения и АЛУ должны быть способны быстро выполнять арифметику с плавающей запятой, набор команд должен быть расширенным (таким, чтобы могли обрабатываться как числа с плавающей запятой, так и числа с

фиксированной запятой) и т.д. Плавающая запятая (32 бита) обладает лучшей точностью и более высоким динамическим диапазоном, чем фиксированная запятая (16 битов). Кроме того, программы с плавающей запятой часто обладают более коротким периодом разработки, поскольку программист вообще не должен беспокоиться о ситуациях подобных переполнению, машинному нулю и ошибке округления.

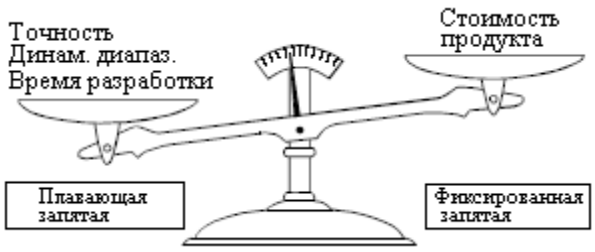


Рис. 28.6 Фиксированная запятая против плавающей запятой

С другой стороны ЦПОС с фиксированной запятой всегда были дешевле, чем устройства с плавающей запятой. Ничто не меняется так быстро как цена на электронику; все, что Вы найдете в книгах, будет уже устаревшим еще до того, как они будут напечатаны. Тем не менее, цена является ключевым фактором в понимании того, как развиваются ЦПОС, и мы должны довести до Вас общую идею. Когда эта книга была закончена в 1999 году, ЦПОС с фиксированной запятой продавались от 5 до 100 долларов за штуку, в то время, как устройства с плавающей запятой были в диапазоне от 10 до 300 долларов. Эта разница в цене может рассматриваться как мера относительной сложности между устройствами. Если Вы хотите выяснить, каковы цены *сегодня*, то Вам нужно посмотреть их *сегодня*.

Теперь, давайте, обратим внимание на функционирование; что может делать 32 разрядная система с плавающей запятой, чего не может делать 16 разрядная система с фиксированной запятой? Ответом на этот вопрос является **соотношение сигнал/шум**. Предположим, что мы отправляем на хранение 32 разрядное число в формате с плавающей запятой. Как упоминалось ранее, промежуток между этим числом и его смежными соседями приблизительно одна десятимиллионная от значения числа. Для хранения числа, оно должно быть округлено в сторону больших или меньших чисел, максимум на половину размера промежутка. Другими словами, каждый раз, когда мы отправляем число на хранение в обозначении с плавающей запятой, мы добавляем к сигналу *шум*.

То же самое происходит тогда, когда число отправляется на хранение в виде 16 битовой величины с фиксированной запятой, за исключением того, что добавленный шум значительно хуже. Это происходит потому, что промежуток между смежными числами значительно больше. Например, предположим, мы отправляем на хранение число 10000 в виде целого со знаком (изменяются от –32768 до 32767). Промежуток между числами одна десятитысячная от значения числа, которое мы собираемся хранить. Если мы хотим хранить число 1000, промежуток между числами всего одна тысячная от значения числа.

Шум в сигнале обычно представляется его *стандартным отклонением*. Это детально обсуждалось в Главе 2. Здесь же, важным фактом является то, что стандартное отклонение этого **шума квантования** приблизительно одна третья размера промежутка. Это означает, что отношение сигнал/шум при хранении числа с плавающей запятой приблизительно 30 миллионов к одному, в то время как для числа с фиксированной запятой всего только около десяти тысяч к одному. Другими словами, плавающая запятая имеет, грубо, в 3000 раз меньший шум квантования, чем фиксированная запятая.

Это дает нам важный способ понимания того, чем ЦПОС отличаются от традиционных микропроцессоров. Предположим, мы реализуем КИХ-фильтр с помощью

фиксированной запятой. Для того чтобы сделать это, мы в цикле берем каждый коэффициент, умножаем его на соответствующий отсчет из входного сигнала и прибавляем результат произведения к содержимому аккумулятора. Вот здесь есть проблема. В традиционных микропроцессорах этот аккумулятор просто еще одна 16 битовая переменная с фиксированной запятой. Для того чтобы избежать переполнения, мы должны смасштабировать складываемые величины, и соответственно, добавим шум квантования на каждом шаге. В наихудшем случае этот шум квантования будет просто суммироваться, сильно снижая отношение сигнал/шум системы. Например, в КИХ-фильтре с 500 коэффициентами, шум каждого выходного отсчета может быть в 500 раз больше, чем шум каждого входного отсчета. Соотношение сигнал/шум с *десяти тысяч к одному*, снизилось до ужасных *двадцать к одному*. Хотя это и экстремальный случай, он демонстрирует основной момент: когда над каждым отсчетом выполняется большое число операций, то это плохо, действительно плохо. Более подробно смотрите Главу 4.

ЦПОС решает эту проблему, используя **аккумулятор расширенной точности**. Это специальный регистр, который имеет разрядность в 2-3 раза больше, чем любая другая ячейка памяти. Например, в 16 разрядном ЦПОС он может иметь от 32 до 40 разрядов, в то время как в ЦПОС SHARC для использования с фиксированной запятой он содержит 80 бит. Этот расширенный диапазон фактически устраняет шум округления во время работы аккумулятора. Единственная ошибка округления допускается при масштабировании содержимого аккумулятора и загрузке его в 16 разрядную ячейку памяти. Такая стратегия работы очень хороша, хотя она ограничивает процесс выполнения некоторых алгоритмов. Для сравнения плавающая запятая имеет настолько низкий шум квантования, что данная техника обычно не требуется.

В дополнение к наличию более низкого шума квантования для систем с плавающей запятой более легко также разрабатывать алгоритмы. Большинство методов ЦОС основаны на повторении умножений и сложений. При фиксированной запятой после каждой операции должны проверяться возможности возникновения переполнения или машинного нуля. Программист должен постоянно понимать амплитуды чисел, то, как происходит накопление ошибки округления и, какое масштабирование следует провести. Для сравнения, с плавающей запятой эти вопросы не возникают; числа заботятся о себе сами (за исключением редких случаев).

Чтобы дать вам более лучшее понимание вопроса, на рис. 28.7 показана таблица из руководства пользователя SHARC. В ней описываются способы, которыми может быть выполнено умножение, как для форматов с фиксированной запятой, так и для форматов с плавающей запятой. Во-первых, посмотрим, как может быть выполнено умножение чисел с плавающей запятой; есть только один способ! То есть Fn = Fx * Fy, где Fn, Fx и Fy – любой из 16 регистров данных. Проще ничего быть не может. Для сравнения посмотрите на все возможные команды для умножения с фиксированной запятой. Они представляют собой множество параметров необходимых для эффективной обработки проблемы округления, масштабирования и форматирования чисел.

На рис. 28.7 Rn, Rx и Ry относятся к любому из 16 регистров данных, а MRF и MRB являются 80 разрядными аккумуляторами. Вертикальные линии отмечают *параметры*. Например, левая верхняя запись в этой таблице означает, что весе следующее имеет силу команды: Rn = Rx * Ry, MRF = Rx * Ry и MRB = Rx * Ry. Другими словами, значение любых двух регистров могут быть перемножены и размещены в другом регистре, или в одном из внешних аккумуляторов расширенной точности. В этой таблице также показано, что числа могут быть либо со знаком, либо без знака (S или U), и могут быть дробными или целыми (F или U). Параметры RND и SAT представляют способы управления округлением и переполнением регистра.

В таблице есть и другие параметры, но они не важны для нашей текущей дискуссии. Важная идея заключается в том, что программист, работающий с фиксированной запятой, должен понимать *дюжину* способов выполнения основной

операции умножения. Для контраста, программист, работающий с плавающей запятой, может потратить свое время, сконцентрировавшись на алгоритме.

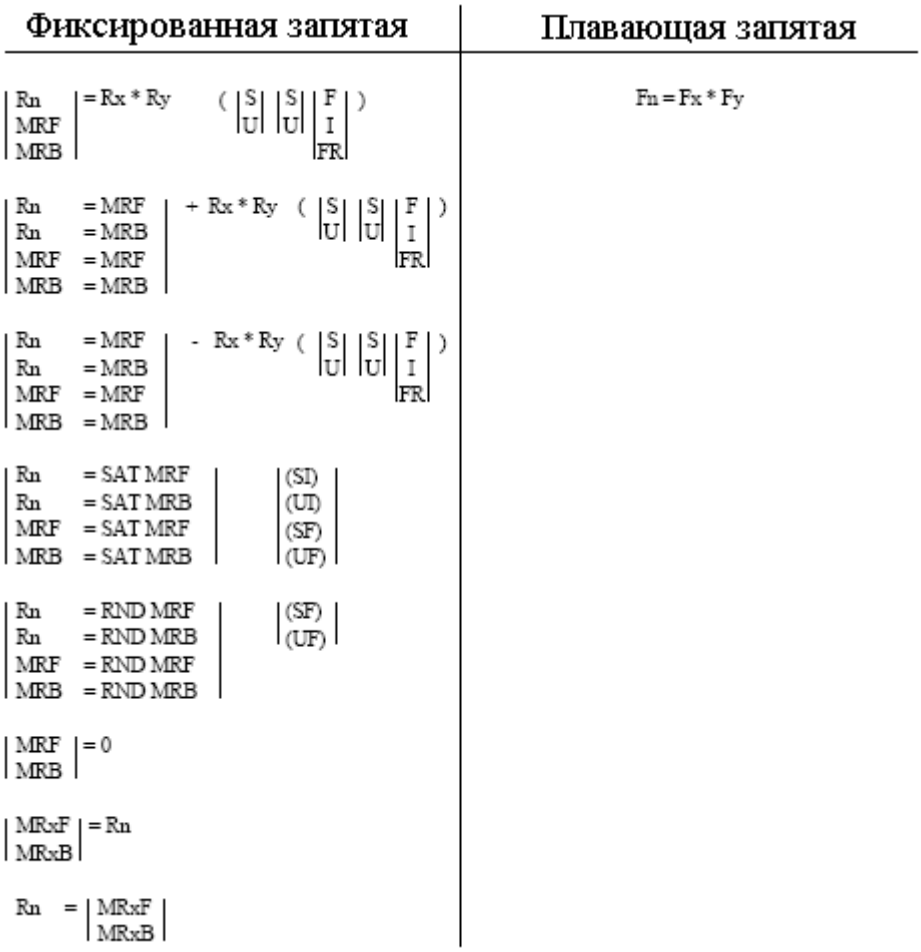


Рис. 28.7 Команды с фиксированной запятой против команд с плавающей запятой

Глядя на эти соотношения между фиксированной и плавающей запятыми, как Вы определитесь, что же использовать? Вот несколько вещей, которые следует рассмотреть. Во-первых, посмотрите, сколько битов используется в АЦП и ЦАП. Во многих приложениях 12-14 битов на отсчет является верхней границей использования фиксированной запятой против плавающей запятой. Например, телевизионные и другие видео сигналы обычно используют 8 битовые АЦП и ЦАП, и здесь приемлема точность фиксированной запятой. Для сравнения профессиональные аудио приложения могут снимать отсчеты с таким большим числом разрядов, как 20 или 24 бита, и почти всегда для охвата большого динамического диапазона, требуют плавающей запятой.

Следующее, что нужно смотреть, это сложность алгоритма, который будет запускаться. Если он относительно простой, думайте о фиксированной запятой, если он более сложный, думайте о плавающей запятой. Например, КИХ фильтрация и другие операции во временной области требуют всего несколько дюжин строк кода, что делает их подходящими для фиксированной запятой. Для контраста, алгоритмы частотной области такие, как спектральный анализ и БПФ свертка очень детализированы и могут быть намного более трудными для программирования. Хотя они могут быть написаны с использованием фиксированной запятой, время разработки будет значительно сокращено, если использовать плавающую запятую.

Наконец подумай те о деньгах: насколько важна стоимость продукта, и насколько важна стоимость разработки программы? При выборе фиксированной запятой стоимость

продукта будет снижена, но стоимость разработки будет вероятно выше из-за более сложных алгоритмов. И наоборот, плавающая запятая обычно обеспечивает более быстрый и более дешевый период разработки, но более дорогой конечный продукт.

На рис. 28.8 показаны некоторые из основных тенденций в ЦПОС. На рис. 28.8a иллюстрируется воздействие цифровых процессоров обработки сигналов на рынок *внедрений*. Ими являются приложения, использующие микропроцессор для непосредственной работы и управления некоторыми более большими системами такими, как сотовый телефон, микроволновая печь или автомобильная инструментальная панель.

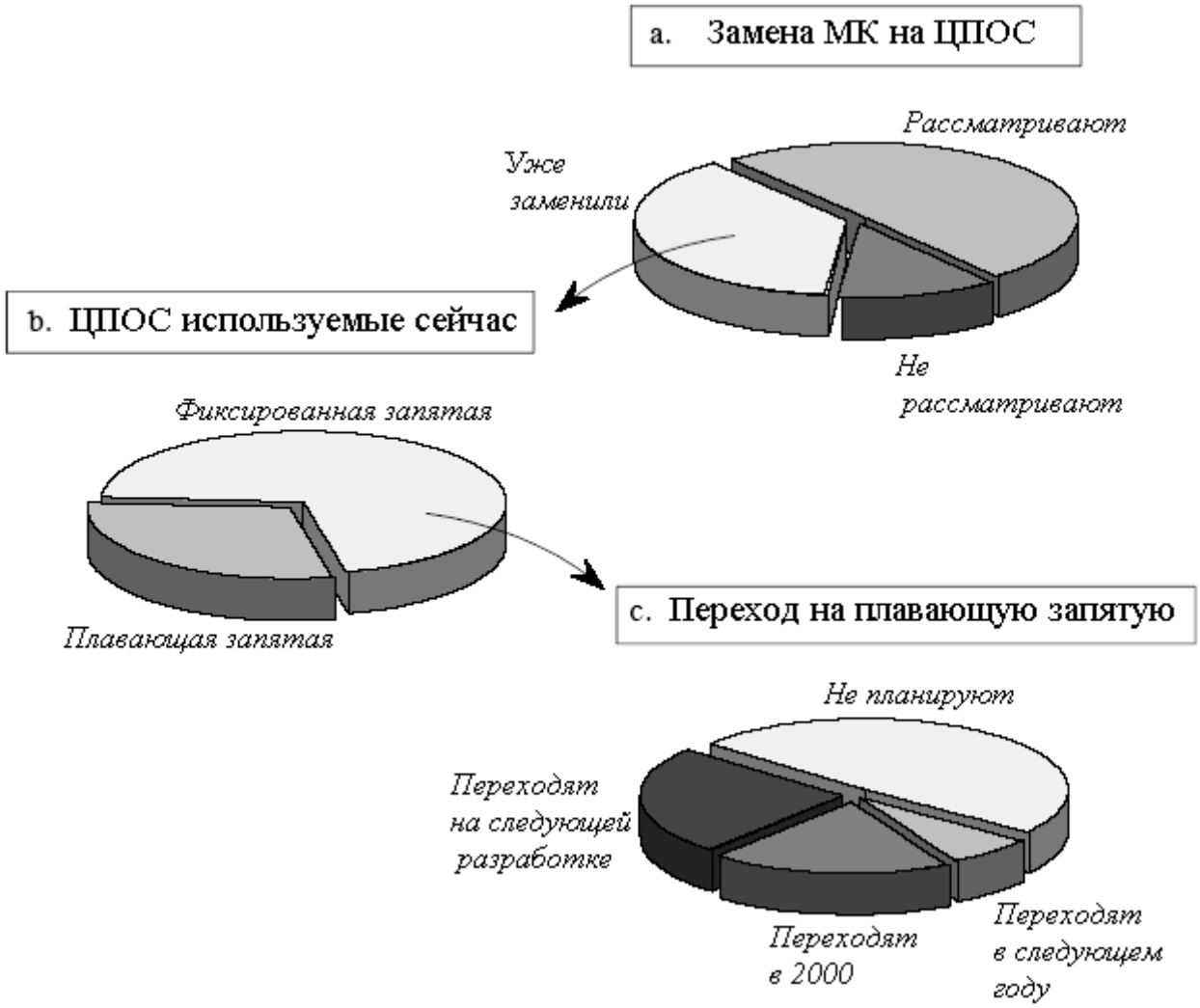


Рис. 28.8 Основные тенденции в ЦОС

При упоминании таких устройств, для того чтобы отличать их от микропроцессоров, используемых в персональных компьютерах, часто используется название “микроконтроллер”. Как показано на рис. 28.8a около 38% разработчиков, работающих на этом рынке, уже начали использовать ЦПОС, а 49% рассматривают переключение на них. Высокая производительность и вычислительная мощность ЦПОС часто делает их идеальным выбором для внедренческих проектов.

Как показано на рис. 28.8b, в настоящее время почти в два раза большее число инженеров вместо плавающей запятой используют фиксированную запятую. Однако это сильно зависит от приложения. Фиксированная запятая более популярна в конкурентоспособных потребительских продуктах, где стоимость электроники должна оставаться очень низкой. Хорошим примером этому являются сотовые телефоны. Когда Вы конкурируете за продажу миллионов Ваших изделий, разница в цене всего на

несколько долларов может быть разницей между успехом и неудачей. Для сравнения, плавающая запятая более обычна тогда, когда требуется более высокая производительность, а стоимость не является важной. Например, представьте, что Вы проектируете медицинскую систему отображения подобную сканеру компьютерного томографа. Их, когда-либо, будет продано всего несколько сотен моделей по цене нескольких сотен тысяч долларов каждая. Для этого приложения стоимость ЦПОС не имеет значения, а производительность является критической. Несмотря на большое число используемых ЦПОС с фиксированной запятой, рынок плавающей запятой является быстро растущим сегментом. Как показано на рис. 28.8c, свыше половины инженеров, использующих 16 битовые устройства, когда-нибудь в ближайшем будущем планируют перейти на плавающую запятую.

Прежде чем оставить эту тему, нам следует еще раз подчеркнуть, что плавающая запятая и фиксированная запятая обычно используют соответственно, 32 бита и 16 битов, *но не всегда*. Например, семейство SHARC может представлять числа в виде 32 битов с фиксированной запятой, режим типичный для цифровых аудио приложений. Это позволяет создать 2^{32} уровней квантования, равномерно расположенных в относительно небольшом диапазоне, скажем между –1 и 1. Для сравнения, в системе обозначений с плавающей запятой 2^{32} уровней квантования логарифмически распределены по большому диапазону, обычно $\pm 3{,}4\times10^{38}$. Это дает 32 битовой фиксированной запятой лучшую *точность*, то есть, ошибка квантования в любом из отсчетов будет ниже. Однако 32 битовая плавающая запятая имеет более высокий *динамический диапазон*, означающий существование большей разницы меду самым большим числом и самым маленьким числом из тех, которые могут быть представлены.

C по сравнению с ассемблером

ЦПОС программируются на тех же языках, что и другие научные и инженерные приложения, обычно на *ассемблере* или ***C***. Программы, написанные на ассемблере, могут исполняться быстрее, в то время как программы, написанные на ***C***, легче разрабатывать и сопровождать. В традиционных приложениях, таких как программы, запускаемые на персональных компьютерах и универсальных ЭВМ, предпочтение почти всегда отдается ***C***. Если вообще используется ассемблер, то это ограничивается короткими подпрограммами, которые должны исполняться с предельной скоростью. Это показано на рис. 28.9a; на каждого традиционного программиста, работающего на ассемблере, приходится приблизительно *десять* работающих на ***C***.

Однако программы ЦОС отличаются от традиционного программного обеспечения в двух важных аспектах. Во-первых, программы обычно значительно короче, скажем, одна сотня строк по сравнению с десятью тысячами строк. Во-вторых, время исполнения часто является критической частью приложения. В конце концов, вот почему, прежде всего, используют ЦПОС, за его ослепительную скорость. Эти два фактора побуждают многих инженеров-программистов при программировании цифровых процессоров обработки сигналов переключаться с ***C*** на ассемблер. Данное иллюстрируется на рис. 28.9b; почти столько же программистов ЦПОС пользуются ассемблером, сколько пользуются и языком ***C***.

На рис. 28.9c, рассматривая доход, полученный от продуктов ЦОС, это развивается дальше. На каждый доллар, полученный с ЦОС, запрограммированной на ***C*,** два доллара получены с ЦОС, запрограммированной на ассемблере. Причина этому проста: деньги делаются на преодолении конкуренции. С точки зрения чистой работы, такой как время исполнения и стоимость производства ассемблер почти всегда имеет преимущество над ***C***. Например, команды ***C*** обычно требуют большей памяти, чем ассемблер, что удорожает аппаратуру. Однако рынок ЦПОС постоянно изменяется. По мере роста рынка производители будут отвечать разработкой ЦПОС, которая *оптимизирована* для

программирования на ***C***. Например, при существовании набора больших регистров общего назначения, и унифицированном пространстве памяти, ***C*** значительно более эффективен. Эти будущие улучшения минимизируют разницу во времени выполнения между ***C*** и ассемблером и позволят использовать ***C*** в еще большем числе приложений.

Для того чтобы лучше понять причины принятия решения в пользу ***C*** или ассемблера, давайте рассмотрим типичную задачу ЦОС, запрограммированную на каждом из этих языков. Пример, который мы будем использовать, представляет собой *скалярное произведение* двух массивов *x*[ ] и *y*[ ]. Это простая математическая операция, мы умножаем каждый коэффициент из одного массива на соответствующий коэффициент в другом массиве и суммируем результаты произведений, т.е. $x[0]\times y[0] + x[1]\times y[1] + x[2]\times y[2] + \ldots$. Это должно выглядеть очень знакомым; это фундаментальная операция в КИХ-фильтрах. То есть, каждый отсчет в выходном сигнале находится умножением хранимых отсчетов из входного сигнала (в одном массиве) на коэффициенты фильтра (в другом массиве) и суммированием результатов произведений.

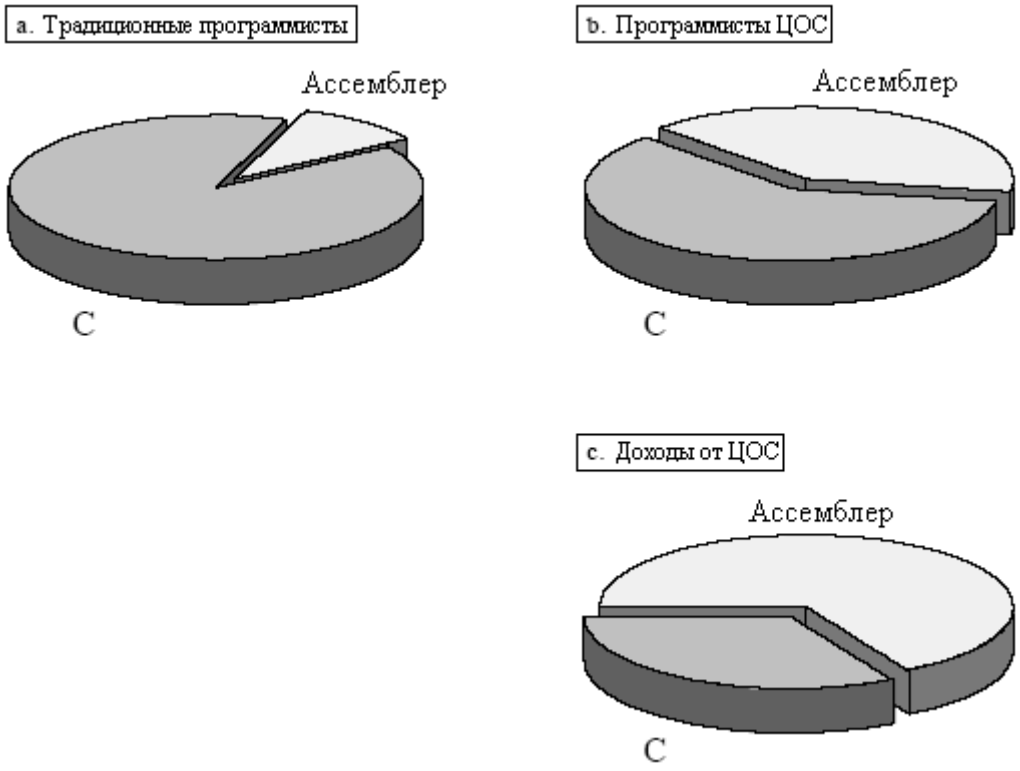


Рис. 28.9 Программирование на **C** против программирования на ассемблере

В таблице 28.2 показано как вычисляется скалярное произведение в программе написанной на ***C***. В строках 001-004 мы определяем длину двух массивов *x*[ ] и *y*[ ] равной 20 элементам. Мы также определяем переменную *result*, содержащую по завершении программы вычисленное скалярное произведение. Строка 011, используя переменную *n* в качестве счетчика циклов, управляет необходимыми для вычисления 20 циклами. Единственное выражение внутри цикла, перемножающее соответствующие коэффициенты из двух массивов и добавляющее результат произведения в аккумулирующую переменную *s*, находится в строке 012. (Если Вы незнакомы с ***C***, выражение: s += *x*[*n*] * *y*[*n*] означает то же самое, что: $s = s + x[n] * y[n]$.) После цикла в строке 013 значение в аккумуляторе *s* передается выходной переменной *result*.

Ключевым преимуществом в использовании языка высокого уровня (такого, как ***C***, Фортран или Бейсик) является то, что программисту не нужно понимать архитектуру используемого микропроцессора; знание архитектуры оставлено для компилятора.

Например, эта короткая программа на ***C*** использует несколько переменных: *n*, *s*, *result*, плюс массивы *x*[ ] и *y*[ ]. Каждой из этих переменных для отслеживания их значений должен быть назначен "дом" в аппаратных средствах. В зависимости от микропроцессора этими местами хранения могут быть регистры общего назначения, ячейки в основной памяти или специальные регистры, предназначенные для специфических функций. Однако, человек, пишущий высокоуровневую программу, знает немного или совсем ничего о таком распределении памяти; эта задача была делегирована инженерам по программному обеспечению, тем - кто написал компилятор. Проблема заключается в том, что эти два человека никогда не встречались; они общаются только через набор предопределенных правил. Языки высокого уровня легче, чем ассемблер потому, что Вы передаете половину работы кому-то другому. Однако, они менее эффективны потому, что Вы весьма не уверены в том, насколько хорошо будет выполнена делегированная работа.

Таблица 28.2

```
001   #define LEN 20
002    float dm x[LEN];
003    float pm y[LEN];
004    float result;
005
006   main()
007
008   {
009    int n;
010    float s;
011    for (n=0;n<LEN;n++)
012      s += x[n]*y[n];
013    result = s
014   }
```

Для сравнения в таблице 28.3 показана программа скалярного произведения, написанная на ассемблере для ЦПОС SHARC. Языки ассемблера для ЦПОС фирмы Analog Devices (оба, как для 16 битовых с фиксированной запятой, так и 32 битовых SHARC приборов) известны за их простой алгебраически подобный синтаксис. Хотя мы не хотим останавливаться на всех деталях, приведем здесь основные операции. Обратите внимание, что *всё* относится к аппаратуре; здесь в команде нет абстрактных переменных, только регистры данных и ячейки памяти.

Таблица 28.3

```
001   i12 = _y;                      /* i12 points to beginning of y[ ] */
002   i4 = _x;                       /* i4 points to beginning of x[ ] */
003
004   lcntr = 20, do (pc,4) until lce;   /* loop for the 20 array entries */
005     f2 = dm(i4,m6);              /* load the x[ ] value into register f2 */
006     f4 = pm(i12,m14);            /* load the y[ ] value into register f4 */
007     f8 = f2*f4;                  /* multiply the two values, store in f8 */
008     f12 = f8 + f12;              /* add the product to the accumulator in f12 */
009
010   dm(_result) = f12;             /* write the accumulator to memory */
```

Каждая точка с запятой представляет тактовый период. Массивы *x*[ ] и *y*[ ] содержатся в кольцевых буферах в основной памяти. В строках 001 и 002 регистры i4 и

i12 указывают на начальное местоположение этих массивов. Затем, под управлением строки 004, мы выполняем 20 циклов. Построение этого оператора использует способность ЦПОС SHARC *зацикливания до обнуления*. Другими словами, все переменные, необходимые для управления циклом, содержатся в выделенных аппаратных регистрах, работающих параллельно с другими, выполняемыми внутри микропроцессора операциями. В данном случае, регистр *lcntr* (счетчик циклов) загружается начальным значением, равным 20, и каждый раз по выполнении цикла декрементируется. Цикл прерывается тогда, когда содержимое *lcntr* достигает значения нуля (в команде это отмечается оператором *lce*, означающим, что "счетчик цикла кончился"). Цикл охватывает строки с 004 по 008, что определяется оператором (pc,4). То есть, цикл заканчивается через четыре строки от текущего содержимого счетчика команд.

Таблица 28.4

```
001     i12 = _y;                                   /* i12 points to beginning of y[ ] */
002     i4 = _x;                                    /* i4 points to beginning of x[ ] */
003
004     f2 = dm(i4,m6), f4 = pm(i12,m14)            /* prime the registers */
005     f8 = f2*f4, f2 = dm(i4,m6), f4 = pm(i12,m14);
006
007     lcntr = 18, do (pc,1) until lce;            /* highly efficient main loop */
008     f12 = f8 + f12, f8 = f2*f4, f2 = dm(i4,m6), f4 = pm(i12,m14);
009
010     f12 = f8 + f12, f8 = f2*f4;                 /* complete the last loop */
011     f12 = f8 + f12;
012
013     dm(_result) = f12;                          /* store the result in memory */
```

Внутри цикла строка 005 загружает значение из *x*[ ] в регистр данных f2, тогда как строка 006 загружает значение из *y*[ ] в регистр данных f4. Символы "dm" и "pm" показывают, что значения вызываются, соответственно, через шину “памяти данных” и шину “памяти программ”. Переменные i4, m6, i12 и m14 являются регистрами генераторов адреса данных, управляющих кольцевыми буферами, содержащими *x*[ ] и *y*[ ]. В строке 007 перемножаются два значения в f2 и f4, а результат сохраняется в регистре данных f8. В строке 008 результат произведения в f8 складывается с содержимым аккумулятора - регистром данных f12. После завершения цикла содержимое аккумулятора в f12 перемещается в память.

Данная программа правильно вычисляет скалярное произведение, но она не использует высоко параллельную архитектуру SHARC. В таблице 28.4 показана эта программа, переписанная в высоко оптимизированном виде с параллельным выполнением многих операций. Во-первых, заметьте, что строка007 выполняет только 18 циклов вместо 20. Заметьте также, что этот цикл содержит только одну строку (008), но что эта строка содержит составную команду. Стратегия здесь состоит в том, что для того чтобы сделать цикл настолько эффективным, насколько это возможно, в данном случае используется одна строка, которая может быть выполнена за один тактовый период. Для того чтобы реализовать эту стратегию полностью, мы должны иметь также небольшое количество команд для осуществления “загрузки” регистров на первом цикле (строки 004 и 005) и еще небольшую часть команд для завершения последнего цикла (строки 010 и 011).

Для того чтобы понять, как это все работает, изучите строку 008 - единственный оператор внутри цикла. В этом единственном операторе параллельно выполняются *четыре* операции: (1) из кольцевого буфера в памяти программ берется значение для *x*[ ] и размещается в f2; (2)) из кольцевого буфера в памяти данных берется значение для *y*[ ] и

размещается в f4; (3) перемножаются предыдущие значения из f2 и f4 и помещаются в f8; и (4) предыдущее значение в f8 прибавляется к содержимому аккумулятора в f12.

Например, когда в пятый раз исполняется строка 008, из памяти извлекаются $x[7]$ и $y[7]$ и сохраняются в f2 и f4. В это же время значения из $x[6]$ и $y[6]$ (которые были в f2 и f4 в начале этого цикла) перемножаются и помещаются в f8. Кроме того, значение из $x[5]\times y[5]$ (которые были в f8 в начале этого цикла) прибавляются к значению в f12.

Давайте, сравним число тактовых периодов, необходимых для выполнения не оптимизированной и оптимизированной программ. Имейте в виду, что в каждом цикле всего только 20 циклов с четырьмя необходимыми действиями. Для не оптимизированной программы необходимо 80 тактовых периодов для выполнения действий в пределах циклов, плюс 5 тактовых периодов на выполнение команд, расположенных перед циклом, в общем 85 тактовых периодов. Для сравнения, оптимизированная программа выполняет 18 циклов за 18 тактовых периодов, но требует 11 тактовых периодов на загрузку регистров и завершение последнего цикла. Это дает полное время выполнения в 29 тактовых периодов, или приблизительно в три раза быстрее, чем метод решения задачи “в лоб”.

А вот и большой вопрос: насколько быстро работает программа, написанная на ***C***, по сравнению с программой на ассемблере? После того, как программа из таблицы 28.2 скомпилирована, похож ли ее исполняемый код на наш *эффективный* или *неэффективный* ассемблеровский пример? Ответом является то, что компилятор генерирует *эффективный* код. Однако важно осознавать, что скалярное произведение является очень простым примером. Когда программа становится более сложной, с многократными вложенными циклами и беспорядочными переходами к подпрограммам, компилятору становится гораздо труднее создавать оптимизированный код. Если Вы делаете что-то простое, то ждите, что компилятор даст Вам почти оптимальное решение. Если же Вы делаете что-то странное или сложное, ждите, что программа на ассемблере будет работать значительно быстрее, чем она же, но написанная на ***C***. Думается, раза в 2-3 в наиболее худшем случае. Как отмечалось ранее, эффективность ***C*** по сравнению с ассемблером зависит в основном от специфики используемого ЦПОС. При использовании языков высокого уровня, подобных ***C***, архитектуры с плавающей запятой в основном могут программироваться более эффективно, чем устройства с фиксированной запятой. Конечно, для этого важно надлежащее программное обеспечение, такое как отладчик с профилирующими характеристиками, помогающими Вам понять, сколько времени будут исполняться различные участки команд.

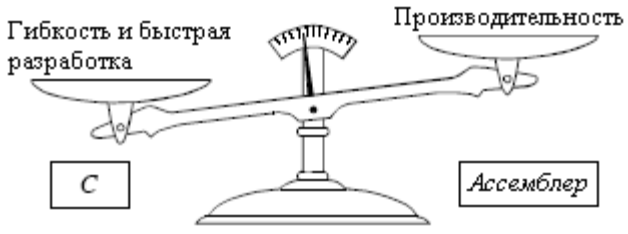


Рис. 28.10 Ассемблер против ***C***

Существует также способ, когда Вы можете получить все лучшее от обоих этих миров: написать программу на ***C***, но на критических участках, которые должны исполняться быстро, использовать ассемблер. Это одна из причин, по которой ***C*** так популярен в науке и технике. Он работает, как язык высокого уровня, но также позволяет Вам непосредственно манипулировать аппаратным обеспечением так, как Вы этого пожелаете. Даже если Вы намереваетесь программировать только на ***C***, Вам, вероятно, потребуются некоторые знания архитектуры ЦПОС и набора ассемблеровских команд. Например, посмотрите назад на строки 002 и 003 в таблице 28.2, программа скалярного

произведения на ***C***. "dm" означает, что *x*[ ] хранится в памяти данных, в то время, как "pm" показывает, что y[ ] располагается в памяти программ. Даже притом, что программа написана на языке высокого уровня, для получения от устройства лучшей производительности, все еще требуются элементарные знания аппаратных средств.

Какой же язык наиболее хорош для *Ваших* приложений? Все зависит от того, что для Вас наиболее важно. Если Вам нужна гибкость и быстрое время разработки программы, выбирайте ***C***. С другой стороны, если Вам нужна самая лучшая из возможных производительность, используйте ассемблер. Как показано на рис. 28.10, это компромисс на который Вы вынуждены пойти. Вот несколько вещей, которые Вам следует рассмотреть.

- Насколько сложной является программа? Если она большая и запутанная, Вы, вероятно, будете хотеть использовать ***C***. Если она маленькая и простая, хорошим выбором может быть ассемблер.
- Вы хотите выжать максимальную скорость из ЦПОС? Если так, ассемблер выжмет Вам последнюю каплю производительности устройства. В менее требовательных приложениях ассемблер имеет небольшие преимущества, и Вы должны рассматривать использование ***C***.
- Сколько программистов будут работать вместе? Если проект достаточно большой, для больше, чем одного программиста, полагайтесь на ***C***, и используйте ассемблерные вставки только на участках критических к времени исполнения.
- Что наиболее важно, *стоимость продукта* или *стоимость разработки*? Если стоимость продукта, выбирайте ассемблер; если стоимость разработки, выбирайте ***C***.
- Какова Ваша подготовка? Если у Вас есть опыт в ассемблировании (на других микропроцессорах) выбирайте для вашего ЦПОС ассемблер. Если Вы раньше работали с ***C***, выбирайте для Вашего ЦПОС ***C***.
- Что предлагает Вам использовать производитель?

Этот последний пункт очень важен. Предположим, Вы спрашиваете производителя ЦПОС, какой язык использовать, и он говорит Вам: “Можно использовать ***C*** или ассемблер, но мы рекомендуем Вам ***C***.” Вам лучше принять их совет. Что же они действительно сказали: *"Наш ЦПОС настолько трудно программировать на ассемблере, что прежде чем его использовать, Вам понадобится 6 месяцев обучения*". С другой стороны, некоторые ЦПОС программируются на ассемблере легко. Например, к этой категории относятся продукты фирмы Analog Devices. Просто спросите их инженеров; они этим очень гордятся.

Один из лучших способов принять решение относительно ЦПОС и программного обеспечения, поговорить с инженерами, которые их используют. Попросите у производителя отзывы компаний, использующих их продукты, или поищите в интернете людей, с которыми вы можете обмениваться электронными письмами. Не стесняйтесь; инженеры любят высказывать свое мнение о продуктах, которые они используют. Они будут польщены, что Вы их спросили.

Насколько быстры ЦПОС?

Первой причиной использования ЦПОС вместо традиционного микропроцессора является *скорость*, способность перемещать отсчеты в устройство, выполнять требуемые математические операции и выдавать обработанные данные. Отсюда возникает вопрос: Насколько быстры ЦПОС? Обычным способом ответа на эти вопросы являются **эталонные тесты**, методы выражающие скорость работы микропроцессора в виде числа. Например, скорость системы с фиксированной запятой часто указывается в **MIPS** (миллионах операций с целыми числами в секунду). Аналогично, скорость устройств с

плавающей запятой может быть выражена в **MFLOPS** (миллионах операций с плавающей запятой в секунду).

Сто пятьдесят лет назад Британский премьер министр Бенджамин Дизраэли заявил, что существует три типа лжи: *ложь*, *дьявольская ложь* и *статистика*. Если бы Дизраэли был сегодня жив и работал с микропроцессорами, он бы добавил *эталонные тесты*, как четвертую категорию. Идея, стоящая за эталонными тестами, обеспечить сравнение, один к одному, для того, чтобы показать какое из устройств лучше. К сожалению, часто на практике это невозможно, поскольку разные микропроцессоры хороши в разных областях. Представьте себе вопрос: Какой автомобиль лучше Кадилак или Феррари? Это зависит от того, для чего вы его хотите использовать!

Неразбериха с эталонными тестами осложняется конкурентным характером электронной промышленности. Производители хотят показать свой продукт в лучшем свете, и они будут использовать любую двусмысленность в испытательной процедуре в свою пользу. В электронике существует старая поговорка: "*Тот, кто пишет техническое описание, может добиться от устройства в два раза большей производительности, чем инженер*". Эти люди не являются обманщиками, просто им платят за то, чтобы у них было хорошее воображение. Эталонные тесты следует рассматривать как *инструмент* для усложненной задачи. Если у Вас нет опыта использования этого инструмента, Вы можете прийти к неправильному заключению. Лучшим подходом является поиск конкретной информации по скорости работы алгоритмов, которые Вы планируете выполнять. Например, если Ваше приложение вызывает подпрограмму КИХ-фильтра, то ищите точное число тактовых периодов, которые потребуются устройству для выполнения этой конкретной задачи.

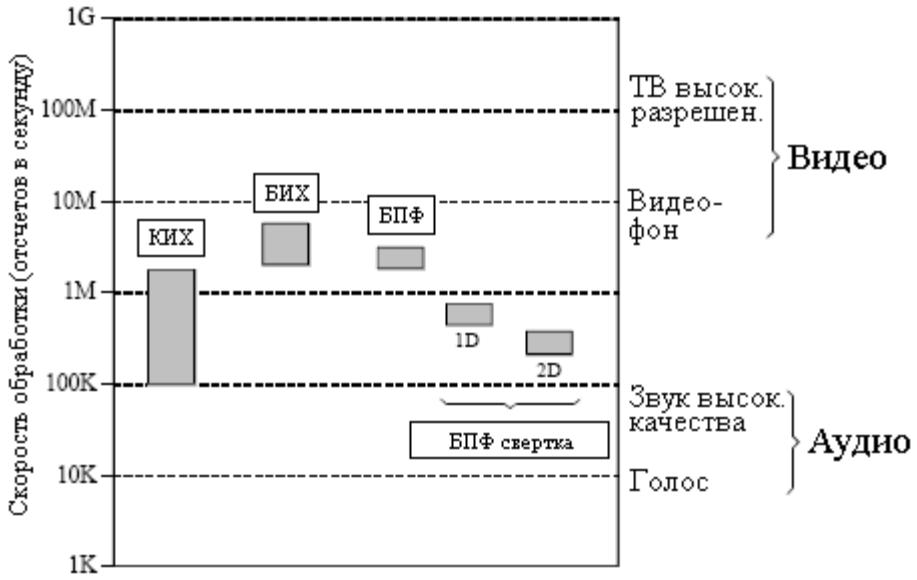


Рис. 28.11 Скорость ЦОС

Пользуясь такой стратегией, давайте посмотрим на время, требующееся для выполнения различных алгоритмов на нашем типичном ЦПОС, семейства SHARC компании Analog Devices. Помните о том, что скорость микропроцессора удваивается приблизительно каждые три года. Это означает, что Вы должны уделять специальное внимание *методам*, которые мы используем в этом примере. Реальные цифры всегда меняются и Вам, следует повторять вычисления каждый раз, когда Вы начинаете новый проект. В мире технологий двадцать первого века только моргните, и Вы уже отстали!

Когда дело доходит до понимания времени выполнения, ЦПОС семейства SHARC, одни из наиболее простых для работы. Это связано с тем, что они могут выполнять

операцию умножение-накопление за один тактовый период. Поскольку большинство КИХ-фильтров используют от 25 до 400 коэффициентов, соответственно, для обработки каждого отсчета требуется от 25 до 400 тактовых периодов. Как описывалось ранее, для достижения такой эффективности цикла понадобится некоторое количество вспомогательных команд (загрузка первого цикла и завершение последнего цикла), но это незначительно, когда количество циклов настолько велико. Для получения пропускной способности фильтра мы должны разделить тактовую частоту SHARC (на сегодня 40 МГц) на число тактовых периодов, необходимых на один отсчет. Это дает максимальную скорость данных для КИХ-фильтра, равную приблизительно от 100K до 1,6M отсчетов в секунду. Более простых вычислений, чем эти быть не может! Эти значения пропускной способности для КИХ-фильтра показаны на рис. 28.11.

Для рекурсивных фильтров вычисления являются настолько же простыми. Типичные БИХ-фильтры используют приблизительно от 5 до 17 коэффициентов. Поскольку эти циклы являются относительно короткими, мы добавим небольшое количество дополнительных периодов, скажем по 3 на каждый отсчет. Это даст от 8 до 20 тактовых периодов необходимых на один отсчет обрабатываемых данных. Для тактовой частоты 40 МГц это дает максимальную пропускную способность БИХ-фильтра от 1,8M до 3,1M отсчетов в секунду. Эти значения для БИХ-фильтра также показаны на рис. 28.11.

Далее мы подходим к технологиям частотной области, основанным на быстром преобразовании Фурье. Производители ЦПОС почти всегда обеспечивают подпрограммами БПФ. Это высоко оптимизированные программы, написанные на ассемблере. В листе спецификации на ADSP-21062 ЦПОС SHARC отмечается, что комплексное БПФ 1024 отсчетов требует 18221 тактовых периодов, или приблизительно 0,46 миллисекунд при 40 МГц. Для вычисления пропускной способности более проще рассматривать это, как 17,8 тактовых периодов на один отсчет. Эта величина “на один отсчет” при более длительных или менее длительных БПФ меняется незначительно. Например, БПФ 256 отсчетов требует около 14,2 тактовых периодов на один отсчет, БПФ 4096 отсчетов требует 21,4 тактовых периодов на один отсчет. Действительное БПФ может быть вычислено на 40% быстрее, чем эти значения, приведенные для комплексного БПФ. Это делает весь диапазон для всех программ БПФ, простирающимся приблизительно от 10 до 22 тактовых периодов на один отсчет, что соответствует пропускной способности приблизительно от 1,8M до 3,3M отсчетов в секунду.

Наиболее быстрым способом реализации КИХ-фильтров является БПФ свертка. В типичном случае из входного сигнала берется сегмент в 512 отсчетов, дополняется 512 добавочными нулями и преобразуется в его частотный спектр при помощи 1024 точечного БПФ. После перемножения этого спектра с желаемым частотным откликом для перехода назад к временной области, используется обратное БПФ. Полученные 1024 точки объединяются со смежными обрабатываемыми сегментами при помощи метода наложения. Это дает 512 точек выходного сигнала.

Сколько это занимает тактовых циклов? Каждый сегмент в 512 отсчетов требует двух 1024 точечных БПФ, плюс небольшого количества вспомогательных операций. Грубо говоря, это приблизительно раз в пять больше, чем для одного 512 точечного БПФ. Поскольку действительное БПФ требует около 12 тактовых периодов на отсчет, БПФ свертка может быть выполнена примерно за 60 тактовых периодов на один отсчет. Для 2106x ЦПОС SHARC при 40 МГц это соответствует пропускной способности данных приблизительно 660K отсчетов в секунду.

Заметим, что это почти тоже самое, что и КИХ-фильтр с 60 коэффициентами, реализованный при помощи обычной свертки. Другими словами, если КИХ-фильтр имеет меньше чем 60 коэффициентов, с помощью обычной свертки он реализуется быстрее. Если он имеет больше чем 60 коэффициентов, быстрее оказывается БПФ свертка. Ключевым превосходством БПФ свертки является то, что с ростом количества коэффициентов время выполнения растет, как логарифм от их числа. Например, 4096

точечное ядро фильтра требует всего, приблизительно, на 30% более длительного исполнения, чем ядро с 512 точками.

БПФ свертка может применяться также и в двухмерных (2D – прим. перев.) задачах, таких как обработка изображения. Например, предположим, что мы хотим обработать в частотной области изображение размером 800×600 пикселей. Во-первых, дополняем изображение нулями для того, чтобы сделать его размером 1024×1024. Затем, с помощью взятия БПФ от каждой из строк, сопровождаемого взятием БПФ от каждого из столбцов результата, вычисляется двухмерный частотный спектр. После умножения этого 1024×1024 спектра на желаемый частотный отклик, берется двухмерное обратное БПФ. Это выполняется взятием обратного БПФ от каждой из строк, а затем каждого из результирующих столбцов. Суммируя число тактовых периодов и деля на число отсчетов, мы находим, что вся эта процедура занимает, грубо, 150 тактовых периодов на пиксель. Для 40 МГц ADSP-2106 это соответствует пропускной способности данных около 260K отсчетов в секунду.

Сравнивая на рис. 28.11 эти разные методы мы можем сделать важное наблюдение. *Почти все методы ЦОС требуют от 4 до 400 команд* (*тактовых периодов*, *для семейства SHARC*) *на исполнение*. Для ЦПОС SHARC, работающих на частоте 40 МГц, мы можем немедленно сделать заключение, что их пропускная способность данных будет между 100K и 10M отсчетов в секунду в зависимости от того, насколько сложный используется алгоритм.

Теперь, когда мы понимаем, насколько быстро ЦПОС *могут* обрабатывать сигналы, давайте, обратим наше внимание в другую сторону; насколько быстро нам *нужно* обрабатывать данные? Конечно же, это зависит от приложения. Мы посмотрим на два наиболее типичных, аудио и видео обработку.

Скорость данных, необходимая для аудио сигнала, зависит от требуемого качества воспроизводимого звука. С нижнего конца находится речь телефонного качества, требующая охвата частот между приблизительно 100 Гц и 3,2 кГц, определяя этим частоту дискретизации, равную приблизительно 8K отсчетов в секунду. Для сравнения, музыка высокого качества должна содержать весь диапазон слуха человека от 20 Гц до 20 кГц. Для каждого из каналов, левого и правого (здесь имеется в виду стереозвучание – прим. перев.), часто используется частота дискретизации равная 44, 1 кГц, что дает полный Hi-Fi (высокого качества – прим. перев.) сигнал с 82,2K отсчетов в секунду. Как соотносится семейство SHARC с этими требованиями? Как показано на рис. 28.11, оно легко может справиться с высококачественным звучанием, или обработать в одно и тоже время несколько дюжин голосовых сигналов.

Видеосигналы - совсем другая история, они требуют приблизительно в тысячу раз большей скорости данных, чем аудио сигналы. Хорошим примером видео низкого качества является CIF (формат общего интерфейса) стандарт для видеофонов. Он использует 352×288 пикселей, по три цвета на пиксель, при 30 кадрах в секунду, с общей скоростью передачи данных 9,1 миллионов отсчетов в секунду. С верхнего конца качества находится HDTV (телевидение с высоким разрешением), использующее 1920×1080 пикселей, по 3 цвета на пиксель, при 30 кадрах в секунду. Это требует скорости передачи данных свыше 186 миллионов отсчетов в секунду. Как показано на рис. 28.11, такие скорости передачи данных за пределами возможностей одного ЦПОС SHARC. Существуют и другие приложения, которые также требуют таких очень высоких скоростей передачи данных, например радиолокаторы, гидролокаторы, а также военные приложения, подобные наведению ракет.

Для того чтобы обрабатывать такие мощные задачи, несколько ЦПОС должны быть объединены в единую систему. Это называется **мультиобработкой** или **параллельной обработкой**. ЦПОС SHARC были спроектированы подобным образом, подразумевая мультиобработку и включают в себя специальные характеристики, чтобы сделать ее настолько легкой, насколько это возможно. Например, для соединения

внешних шин нескольких ЦПОС SHARC не требуется никакой внешней аппаратной логики; вся логика арбитража шины уже содержится внутри каждого устройства. В качестве альтернативы, для объединения нескольких процессоров в различные конфигурации могут использоваться порты связи (параллельный - 4 бита). На рис. 28.12 показаны типичные способы, которыми ЦПОС SHARC могут быть организованы в мультипроцессорную систему. На рис. 28.12a показана одна из схем объединения, в этой схеме алгоритм разбивается на последовательные этапы, причем в стратегии “конвейера” каждый процессор выполняет только один из этапов. На рис. 28.12b процессоры взаимодействуют через одну совместно используемую глобальную память, осуществляя доступ через параллельную шину (т.е. внешний порт). На рис. 28.13 показан другой способ, с помощью которого в единую систему может быть объединено большое число процессоров, 2D или 3D “сеть”. Каждая из этих конфигураций для конкретной задачи будет иметь относительные достоинства и недостатки.

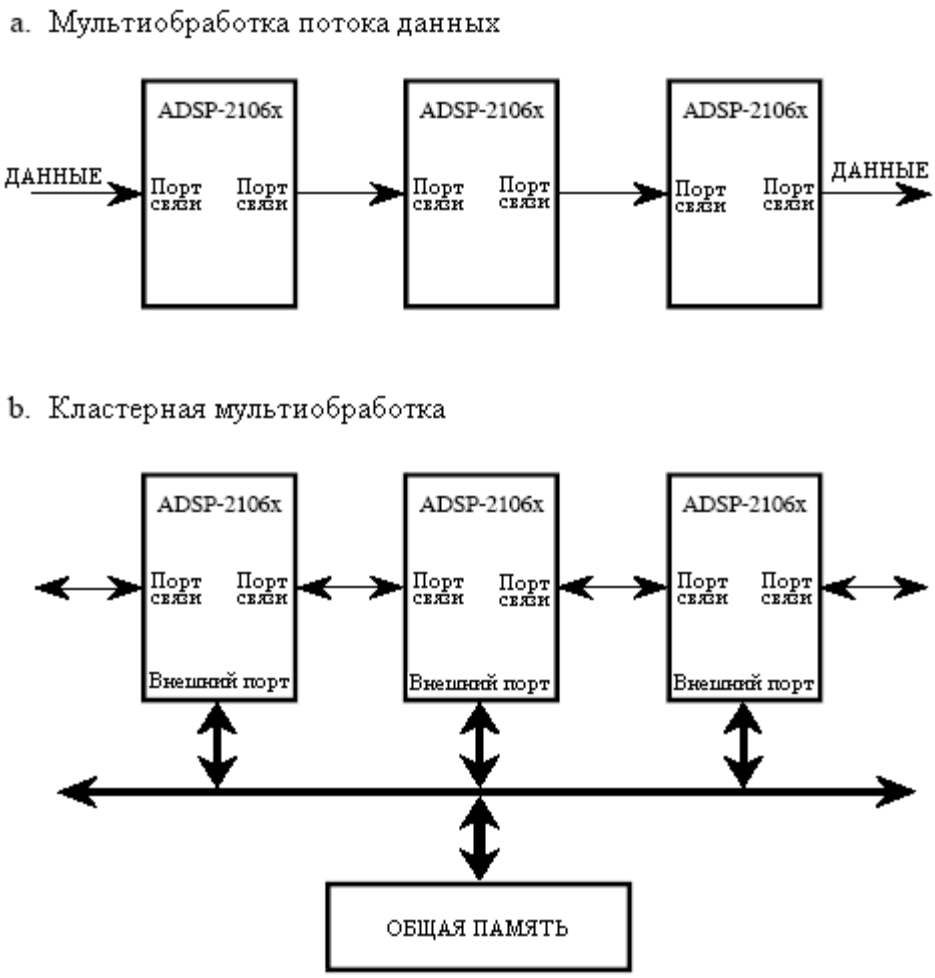


Рис. 28.12 Мультипроцессорная конфигурация

Для того, чтобы облегчить жизнь программисту, семейство SHARC использует *единое адресное пространство*. Это означает, что адресное пространство в 4 Гигаслова, доступ к которому осуществляется через 32 битовую шину адреса, делится между разными работающими вместе процессорами. Для передачи данных от одного процессора к другому просто запишите или прочтите их из соответствующего места памяти. Об остальном позаботится внутренняя логика SHARC, передавая данные между процессорами с такой высокой скоростью, как 240 Мегабайт в секунду (при тактовой частоте 40 МГц).

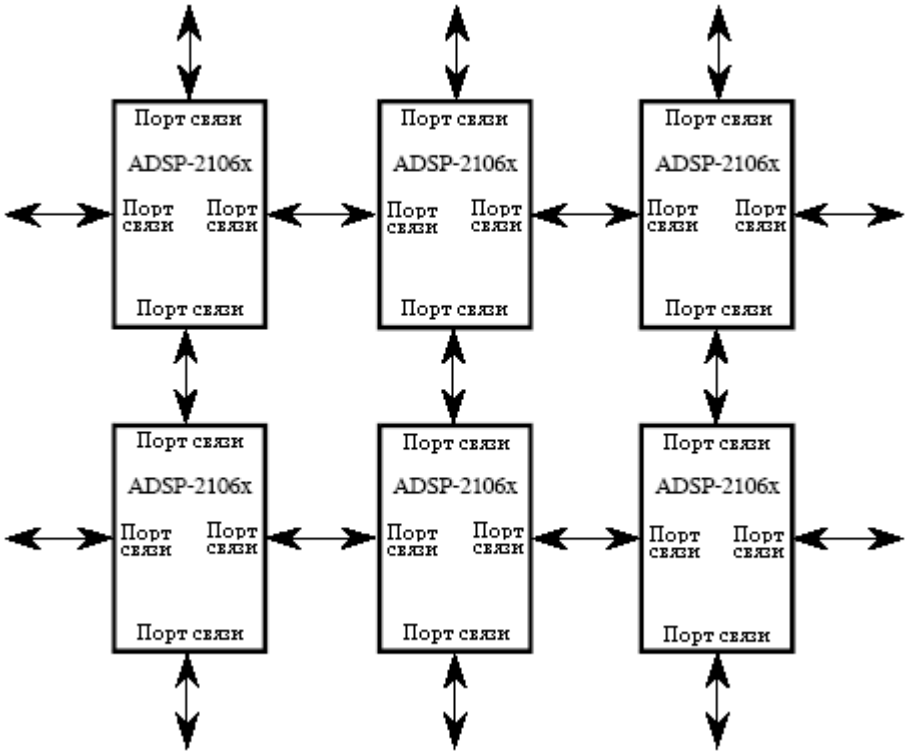


Рис. 28.13 Конфигурация мультипроцессорной “сети”

Рынок цифровых процессоров обработки сигналов

Рынок ЦПОС очень большой и быстрорастущий. Как показано на рис. 28.14, на смене столетий он будет составлять приблизительно 8-10 миллиардов долларов в год, и ежегодно расти со скоростью 30-40 %. Это будет обеспечиваться непрерывным спросом на лучшие и более дешевые потребительские товары, такие как: сотовые телефоны, мультимедийные компьютеры, высококачественное воспроизведение звука. Эти высокодоходные приложения формируют область, в то время как менее выгодные сферы, такие как научная аппаратура, просто находятся на гребне технологии.

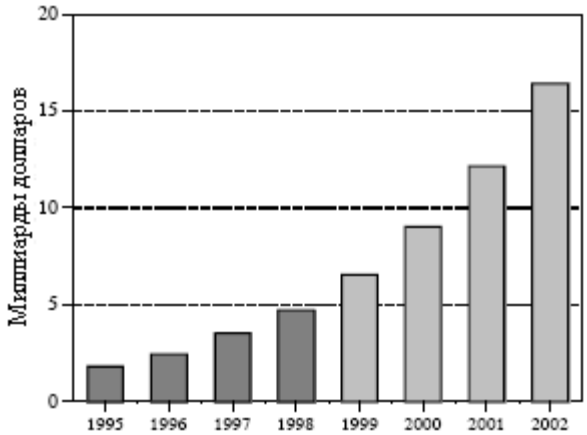


Рис. 28.14 Рынок ЦОС

ЦПОС может быть куплен в трех формах, как **ядро**, как **процессор**, и как продукт **уровня платы**. В терминах ЦПОС, “ядро” относится к части процессора, где выполняются ключевые задачи, включающей: регистры данных, умножитель, АЛУ, генератор адреса и программный секвенсер. Полный **процессор** требует объединения ядра с памятью и интерфейсом для связи с внешним миром. Несмотря на то, что ядро и эти внешние части разработаны отдельно, они будут изготовлены на том же самом кусочке кремния, что сделает *процессор* единой интегральной схемой.

Предположим, Вы создаете сотовые телефоны и хотите включить в конструкцию ЦПОС. Вы, вероятно, захотите купить ЦПОС в виде *процессора*, то есть интегральную микросхему (“чип”), содержащую ядро, память и другие внутренние детали. Например, ADSP-21060 семейства SHARC поставляется в упаковке “240 свинцовая метрическая PQFP” размером всего 35×35×4 миллиметра. Для того чтобы встроить эту ИС в Ваш продукт, Вы разрабатываете печатную плату, на которую она будет впаяна рядом с другой Вашей электроникой. Это наиболее обычный способ использования ЦПОС.

Теперь, предположим, что компания, в которой Вы работаете, производит свои собственные интегральные схемы. В данном случае Вы можете не захотеть весь *процессор*, а только конструкцию *ядра*. После заключения соответствующего лицензионного соглашения, Вы можете начинать делать чипы, которые высоко адаптированы к вашему конкретному приложению. Это дает Вам гибкость выбора в количестве подключенной памяти, в способах приема и передачи данных, в выборе типа корпуса и так далее. Заказные устройства такого типа становятся все более и более важной долей рынка ЦПОС.

Наконец, существует несколько дюжин компаний, которые продадут Вам ЦПОС уже установленные на печатные платы. Они обладают такими особенностями как дополнительная память, аналого-цифровые и цифро-аналоговые преобразователи, гнезда перепрограммируемого ПЗУ, мультипроцессор на той же самой плате, и так далее. Хотя некоторые из этих плат предназначены для использования в виде отдельного компьютера, большинство их сконфигурированы так, чтобы быть встроенными в главную ЭВМ, наподобие персонального компьютера. Компании, производящие такой вид плат называются **разработчиками третьей части**. Лучший способ их найти – это спросить у производителя ЦПОС, который Вы хотите использовать. Зайдите на вебсайт производителя ЦПОС; если Вы не нашли списка там, свяжитесь с ними по электронной почте. Они будут больше чем счастливы, рассказать Вам, кто использует их продукты и как с ними связаться.

На сегодняшний день (1998) на рынке цифровых процессоров обработки сигналов доминируют четыре компании. Здесь приводится их перечень и основная схема, которую они используют для нумерования своих товаров:

Analog Devices (*www.analog.com/dsp*)
ADSP-21xx 16 бит, фиксированная запятая
ADSP-21xxx 32 бита, плавающая и фиксированная запятая

Lucent Technologies (*www.lucent.com*)
DSP16xxx 16 бит, фиксированная запятая
DSP32xx 32 бита, плавающая запятая

Motorola (*www.mot.com*)
DSP561xx 16 бит, фиксированная запятая
DSP560xx 24 бита, фиксированная запятая
DSP96002 32 бита, плавающая запятая

Texas Instruments (*www.ti.com*)
TMS320Cxx 16 бит, фиксированная запятая
TMS320Cxx 32 бита, плавающая запятая

Не забывайте, что различия между ЦПОС и традиционными микропроцессорами не всегда очевидны. Например, посмотрите, как Intel описывает добавление технологии MMX к своему процессору Pentium:

> "*Инженеры Intel добавили 57 мощных новых команд, специально разработанных для эффективного манипулирования и обработки видео, аудио и графическими данными. Эти команды ориентированы на высоко параллельные, повторяющиеся последовательности, часто обнаруживаемые в мультимедийных операциях*".

В будущем мы, несомненно, увидим еще больше ЦПОС - подобных функций, встроенных в традиционные микропроцессоры и микроконтроллеры. Интернет и другие мультимедийные приложения являются сильной движущей силой для этих изменений. Эти приложения расширяются так быстро, что через двадцать лет, вполне вероятно, что цифровой процессор обработки сигналов может стать “традиционным” микропроцессором.

Как же Вам не отстать от этой быстро изменяющейся области? Лучший способ - читать каталоги, охватывающие рынок ЦПОС, такие, как EDN (Новости электронных разработок, *www.ednmag.com*), и ECN (Новости электронных компонентов, *www.ecnmag.com*). Они распространяются бесплатно, и содержат последнюю информацию относительно того, что является доступным и куда движется промышленность. Каталоги являются “обязательным” чтивом для всех, серьезно занимающихся этой областью. Вы можете также захотеть попасть в список адресатов некоторых производителей ЦПОС. Это позволит Вам получать объявления о новых продуктах, информацию о ценах и специальные предложения (такие, как бесплатное программное обеспечение и оценочные комплекты по низкой цене). Некоторые производители распространяют также периодические информационные бюллетени. Например, Analog Devices публикует четыре раза в год “*Аналоговый диалог*”, содержащий статьи и информацию по текущим вопросам в обработке сигналов. Со всеми этими ресурсами, и гораздо большими, можно вступить в контакт по интернету. Начните с исследования вебсайта производителя, а затем пошлите ему электронное письмо с запросом конкретной информации.

ГЛАВА
29

НАЧНЕМ С ЦПОС

Однажды Вы решите, что для Ваших приложений неплох цифровой процессор обработки сигналов, Вам только нужно с чего-то начать. Большое число производителей продадут Вам недорогие оценочные комплекты, позволяющие Вам собственноручно испытать их продукты. Это грандиозное учебное пособие; не имеет значения новичок Вы или профи, оно является наилучшим способом познакомиться с конкретным ЦПОС. Например, Analog Devices для обучения потенциальных клиентов работе с цифровыми процессорами обработки сигналов семейства SHARC® предоставляет EZ-KIT® Lite. Всего за 179 долларов Вы получите все аппаратное и программное обеспечение необходимое для того, чтобы увидеть ЦПОС в действии. Сюда входят “стандартные” программы, поставляемые с комплектом, а также приложения, которые Вы можете написать и самостоятельно - на ассемблере или ***C***. Предположим, что Вы купили один из этих комплектов от Analog Devices и несколько дней балуетесь с ним. Эта глава является обзором того, что Вы можете предположительно найти и чему научиться.

Семейство ADSP-2106x

В предыдущей главе мы рассмотрели основные операции ADSP-2106x - семейства цифровых процессоров обработки сигналов "SHARC". В таблице 29.1 показаны различные члены этого семейства. Все устройства используют одну и туже архитектуру, но имеют разный объем встроенной памяти, что является ключевым фактором в принятии решения, какое из этих устройств использовать. Доступная память - общее узкое место в системах с ЦПОС. ЦПОС SHARC борется с этим посредством предоставления полного удовлетворения встроенной двух портовой SRAM (статическое оперативное запоминающее устройство – прим. перев.). Однако платить за большую память, чем Вам нужно, это последняя вещь, которую Вы захотите сделать. ЦПОС часто входит в состав чувствительных к цене изделий, таких как сотовые телефоны и CD плееры. Другими словами, построение этого семейства определяется *рынком* точно так же, как и *технологией*.

Старейшим членом этой семьи является ADSP-21020. Этот чип содержит ядро архитектуры, но не включает встроенную память и управление вводом/выводом. Это означает, что он не может функционировать как автономный компьютер; для того чтобы стать функциональной системой, он требует внешних компонент. Другие же устройства являются полными однокристальными ЭВМ. Для работы всем им требуется источник питания и некоторый способ загрузки программы в память, подобный внешнему PROM (программируемое постоянное запоминающее устройство – прим. перев.) или каналу связи.

Заметьте, что даже находящиеся в нижней части таблицы 29.1 продукты, имеют очень значительный объем памяти. Например, ADSP-21065L имеет 544Кбит внутренней SRAM. Этого достаточно для того, чтобы сохранить 6-8 секунд цифровой речи (8K отсчетов в секунду, 8 битов на отсчет). В верхней части семейства находится ADSP-21060, имеющий 4Мбит памяти. Этого больше чем достаточно для сохранения полностью оцифрованного изображения (512×512 пикселей, 8 битов на пиксель). Если Вам требуется еще больше памяти, Вы легко добавите к любому из этих устройств внешнее SRAM (или более медленную память).

Таблица 29.1

ПРОДУКТ	Память	Примечание
AD1460	4 Mbit ×4	Quad-SHARC, четыре ADSP-21060 в одном модуле, обеспечивает невероятные 480 MFLOPS, при размере всего 2,05"×2,05"×0,16".
ADSP-21160M	4 Mbit	Новинка! Архитектура ядра типа одна команда множественные данные (SIMD); оптимизирован для работы в мультипроцессорном режиме с портом связи; внешняя шина 64 бита, 14 каналов ПДП
ADSP-21060	4 Mbit	Мощный дом семейства; максимальная память; порты связи для высокоскоростной передачи данных и мультипроцессорной обработки
ADSP-21062	2 Mbit	Те же характеристики, что и у ADSP-21060, но с меньшей внутренней памятью (SRAM), за меньшую цену
ADSP-21061	1 Mbit	Недорогая версия используемая в EZ-KIT Lite; минимум памяти и отсутствие портов связи; дополнительная характеристика - ПДП для последовательного порта
ADSP-21065L	544 kbit	Недавнее дополнение к семейству; быстрый и очень недорогой ($10). Привлечет большое число приложений с фиксированной запятой к семейству SHARC
ADSP-21020	-0-	Старейший член семейства. Содержит ядро процессора без встроенной памяти или без интерфейса ввода/вывода. Не совсем ЦПОС SHARC

Кроме памяти, члены этого семейства различаются также их секциями ввода/вывода. ADSP-21060 и ADSP-21062 (верхняя часть семейства) имеют шесть *портов связи* каждый. Последние представляют собой 4 битовые параллельные связи для объединения ЦПОС в мультипроцессорных системах и других приложениях, требующих гибкого высокоскоростного ввода/вывода. ADSP-21061 и ADSP-21065L (нижняя часть семейства) не имеют портов связи, но характеризуются большим числом каналов ПДП, помогающих работе с *последовательными портами*. Позади некоторых номеров в обозначении устройств Вы будете также встречать буквы "L" или "M", наподобие "ADSP-21060L". Это означает, что устройство работает от напряжения ниже, чем традиционные 5 вольт. Например, ADSP-21060L работает от 3,3 вольт, в то время как ADSP-21160M использует всего 2,5 вольта.

В июне 1998 года Analog Devices обнародовала второе поколение своего семейства архитектуры SHARC с анонсированием ADSP-21160. Он характеризуется архитектурой ядра типа одна команда - множественные данные (SIMD или “сим-ди”), работающей на 100 МГц, с ускоренной шиной памяти с диапазоном до 1600 мегабайт в секунду, двумя 64-битовыми шинами данных и четырьмя 80-битовыми аккумуляторами для вычислений с фиксированной запятой. В целом, новый ADSP-21160M выполняет 1024 точечное БПФ всего за 46 микросекунд. ЦПОС SIMD содержит второй набор вычислительных блоков

(арифметико-логическое устройство, многорегистровое циклическое сдвиговое устройство, файл регистров данных и умножитель), позволяющий Analog Devices, Inc. поддерживать обратную совместимость кодов с семейством ADSP-2106x при обеспечении прогона программ с более высокой, доходящей до десяти раз производительностью.

Комплект EZ-KIT Lite семейства SHARC

Комплект EZ-KIT Lite дает Вам все, что Вам нужно для изучения ЦПОС SHARC, включая аппаратное обеспечение, программное обеспечение и руководство пользователя. На рис. 29.1 показана блок-схема аппаратного обеспечения, входящего в комплект EZ-KIT Lite, построенного на базе цифрового процессора обработки сигналов ADSP-21061. Оно поставляется в виде 4½ × 6½ дюймовой печатной платы, установленной на пластмассовой подставке, позволяющей располагать ее на Вашем столе. (Существует также версия, называемая EZ-LAB, использующая ADSP-21062, которая вставляется в разъем в вашем компьютере.) Вам нужно побеспокоиться всего о четырех соединениях: с источником питания постоянного тока, с Вашим компьютером, через последовательный интерфейс (RS-232), с входным и выходным сигналами. В комплекте даже есть источник питания постоянного тока и кабель для последовательного соединения. Входные и выходные сигналы - уровня звука, с амплитудой около 1 вольта. В качестве альтернативы, перемычка на плате позволяет непосредственно к входу присоединить микрофон. Идея состоит в том, чтобы подключить микрофон к входу, а к выходу присоединить установку из громкоговорителей с усилителем (наподобие используемой с персональными компьютерами). Это позволит Вам *услышать* эффект различных алгоритмов ЦОС.

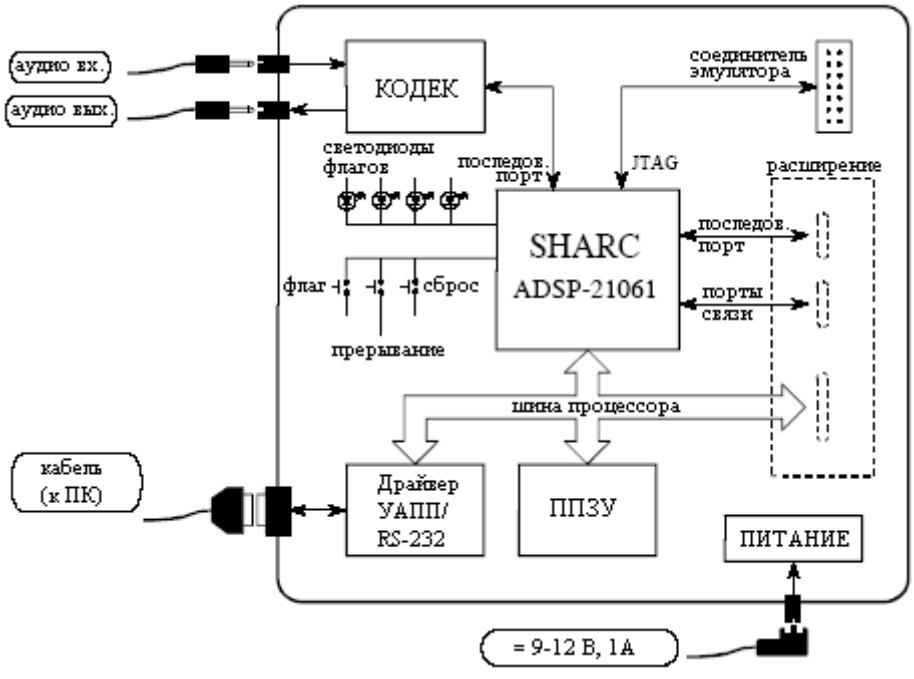


Рис. 29.1 Блок-схема комплекта EZ-KIT Lite

Аналого-цифровое и цифро-аналоговое преобразования осуществляются с помощью кодека (кодер-декодер) Analog Devices AD1847. Это 16 битовый сигма-дельта преобразователь, способный оцифровывать два канала (стерео) со скоростью до 48K отсчетов в секунду и одновременно с той же скоростью осуществлять вывод двух каналов. Поскольку основным назначением этой платы является обработка аудио сигналов, входы и выходы имеют развязку по постоянному току с частотой среза около 20 Гц.

Три кнопки на плате позволяют пользователю генерировать прерывание, сброс процессора и переключать бит флага, который может быть считан системой. Четыре установленных на плате светодиода могут быть включены или выключены переключением битов. Если Вы честолюбивы, то на плате имеется секция, которая позволяет Вам осуществить доступ к последовательному порту, портам связи (только на EZ-LAB с ADSP-21062) и шине процессора. Однако они *непопулярны*, и Вам придется установить соединители и другие компоненты самостоятельно.

Вот как это работает. Когда питание подано, процессор осуществляет загрузку из расположенного на плате ППЗУ (512 килобайт), загружая программу, устанавливающую связь через последовательный интерфейс с Вашим персональным компьютером. Затем Вы запускаете на Вашем ПК *основную* программу *EZ-Lite*, позволяющую Вам загружать программы и пересылать данные из ЦПОС. В составе EZ-KIT Lite поставляются несколько предварительно написанных программ; они могут быть запущены простым щелчком мыши по иконке. Например, полосно-пропускающая программа позволяет Вам говорить в микрофон и слышать результат после прохождения через полосно-пропускающий фильтр. Эти программы полезны по двум причинам: (1) они позволяют Вам быстро получить систему выполняющую что-то интересное, давая Вам уверенность в том, что это работает, и (2) они предоставляют Вам шаблон для создания Ваших собственных программ. Что и подводит нас к нашей следующей теме, примеру разработки с использованием EZ-KIT Lite.

Пример разработки: аудио КИХ-фильтр

После того, как некоторое время Вы поэкспериментируете с предварительно написанными программами, Вы захотите изменить их для получения опыта в программировании. Программы могут быть написаны либо на ассемблере, либо на ***C***; EZ-KIT Lite обеспечивает программным инструментарием, поддерживающим оба языка. Позже в этой главе мы рассмотрим современные методы программирования, такие как *симулирование* (имитирование – прим. перев.), *отладка* и работа в *интегрированной среде разработки*. Сейчас мы сфокусируемся на наиболее простом способе заставить программу работать. Маленьким ножкам - маленькие шаги.

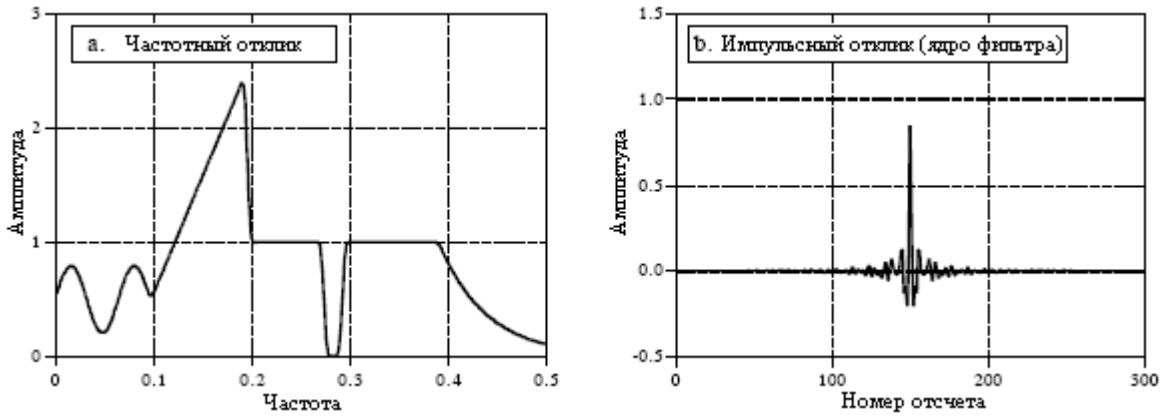


Рис. 29.2 Пример КИХ фильтра

Поскольку исходные коды представлены в ASCII, стандартный текстовый редактор - это все, что нужно для осуществления изменений в существующих файлах или создания полностью новой программы. В таблице 29.2 показан пример программы КИХ-фильтра написанной на ассемблере. Хотя это единственный код, о котором Вам сейчас следует беспокоиться, имейте в виду, что существуют и другие файлы необходимые для того, чтобы сделать эту программу завершенной. Сюда входят “файл описания архитектуры”

(который определяет конфигурацию аппаратуры и распределение памяти), таблица распределения векторов прерываний и подпрограмма инициализации кодека. В конечном итоге, Вам нужно будет понять, что происходит и в этих разделах, но сейчас Вам следует просто копировать их из предварительно написанных программ.

Как показано вверху таблицы 29.2, есть три переменные, которые следует определить перед переходом к основному разделу кодов. Ими являются: **NR_COEF** – число точек в ядре фильтра; **dline[ ]** – кольцевой буфер удерживающий последние поступившие отсчеты из входного сигнала; **coef[ ]** – кольцевой буфер в котором содержится ядро фильтра. Нам также нужно предоставить программе две других части информации: частоту дискретизации кодека и имя файла, содержащего ядро фильтра, чтобы он мог быть считан в coef[ ]. Все эти шаги достаточно простые; не более чем по одной строке кода на каждый. В этом примере мы их не показываем потому, что они содержатся на участках кода, которые мы для простоты игнорируем.

На рис. 29.2 показано ядро фильтра, на примере которого мы будем тестировать программу - это тот же самый обычный фильтр, который мы разрабатывали в Главе 17. Как Вы помните, этот фильтр был выбран потому, что у него очень нерегулярный частотный отклик, усиливающий представление о том, что цифровые КИХ-фильтры могут обеспечить фактически любой желаемый частотный отклик. На рис. 29.2a показан частотный отклик нашего тестового фильтра, в то время как на рис. 29.2b показан соответствующий импульсный отклик (т.е. ядро фильтра). Это ядро фильтра, состоящее из 301 точки, загружается в ASCII файл и для формирования единой исполняемой программы объединяется с другими участками кода во время линкования.

Основная часть программы выполняет две функции. В строках с 6 по 13 конфигурируются генераторы адреса данных для управления кольцевыми буферами: dline[ ] и coef[ ]. Как описывалось в предыдущей главе, для каждого буфера требуется три параметра: начало буфера в памяти (b0 и b8), длина буфера (l0 и l8), размер шага данных, хранящихся в буфере (m0 и m8). Эти параметры, управляющие кольцевыми буферами, хранятся в регистрах аппаратного обеспечения в генераторах адреса данных, позволяя им очень эффективно осуществлять доступ и управление данными.

Вторым действием основной программы является цикл “битья баклуш” реализуемый в строках с 15 по 19. Он ничего не совершает, а ожидает прерывания, показывающего, что входной отсчет уже получен. Вся обработка в этой программе происходит по принципу **отсчет за отсчетом**. Каждый раз, как только с входа происходит считывание отсчета, вычисляется отсчет в выходном сигнале и направляется в кодек. В эту категорию попадает большинство алгоритмов временной области, таких как КИХ и БИХ-фильтры. Альтернативной является обработка **кадр за кадром**, которая требуется для методов частотной области. В методах кадр за кадром с входа считывается *группа* отсчетов, проводятся вычисления, и *группа* отсчетов переписывается на выход.

Подпрограмма, обслуживающая прерывание отсчет – готов, разбита на три части. Первая часть (строки с 27 по 33) считывает отсчет с кодека в виде числа с фиксированной запятой и преобразует его в число с плавающей запятой. В языке ассемблера SHARC на регистр данных, в котором хранится число с фиксированной запятой, ссылаются как на “r” (например, r0, r8, r15 и т.д.), а если в нем хранится число с плавающей запятой, ссылаются как на “f” (т.е. f0, f8 или f15). Например, в строке 32 число с фиксированной запятой в регистре данных 0 (т.е. r0) преобразуется в число с плавающей запятой и переписывается в регистр данных 0 (т.е. f0). Такое преобразование осуществляется в соответствии с масштабом, определенным числом с фиксированной запятой в регистре данных 1 (т.е. r1). В третьей части (строки с 47 по 53) имеют место обратные шаги; число с плавающей запятой для выходного отсчета преобразовывается к числу с фиксированной запятой и отправляется в кодек.

КИХ-фильтр, преобразующий входные отсчеты в выходные отсчеты, содержится в строках с 35 по 45. Все вычисления выполняются над числами с плавающей запятой, что

позволяет избежать необходимости беспокоиться о масштабировании и переполнении. Как описывалось в предыдущей главе, для того чтобы использовать преимущества в способности ЦПОС семейства SHARC параллельно выполнять много команд за один тактовый период, этот участок кодов оптимизирован.

Таблица 29.2

```
001  /************************************************************************
002  ******************        MAIN PROGRAM        **********************
003  ************************************************************************/
004  main:
005
006  /* INITIALIZE THE DAGS TO CONTROL THE CIRCULAR BUFFERS */
007
008  b0 = dline;          /* set up dline[ ], the buffer holding the past input samples */
009  l0 = @dline;
010  m0 = 1;
011  b8 = coef;           /* set up coef[ ], the buffer holding the filter coefficients */
012  l8 = @coef;
013  m8 = 1;
014
015  /* ENTER A LOOP, WAITING FOR THE SAMPLE-READY INTERRUPT */
016
017  wait:
018  idle;
019  jump wait;
020
021
022  /************************************************************************
023  *********     SUBROUTINE TO PROCESS ONE SAMPLE        ***********
024  ************************************************************************/
025  sample_ready:
026
027  /* ACQUIRE THE INPUT SAMPLE, CONVERT TO FLOATING POINT */
028
029  r0 = dm(rx_buf + 1);          /* move the input sample into r0 */
030  r0 = lshift r0 by 16;         /* shift to the highest 16 bits to preserve the sign */
031  r1 = -31;                     /* set the scaling for the conversion */
032  f0 = float r0 by r1;          /* convert from fixed to floating point */
033  dm(i0,m0) = f0;               /* store the new sample in dline[ ], and zero f12 */
034
035  /* CALCULATE THE OUTPUT SAMPLE FROM THE FIR FILTER */
036
037  f12 = 0;                                          /* prime the registers */
038  f2 = dm(i0,m0), f4 = pm(i8,m8);
039  f8 = f2*f4, f2 = dm(i0,m0), f4 = pm(i8,m8);
040                                                    /* efficient main loop */
041  lcntr = NR_COEF-2, do (pc,1) until lce;
042  f8 = f2*f4, f12 = f8+f12, f2 = dm(i0,m0), f4 = pm(i8,m8);
043
044  f8 = f2*f4, f12 = f8+f12;                         /* complete the last loop */
045  f12 = f8+f12;
046
047  /* CONVERT THE OUTPUT SAMPLE TO FIXED POINT & OUTPUT */
048
049  r1 = 31;                      /* set the scaling for the conversion */
050  r8 = fix f12 by r1;           /* convert from floating to fixed point */
051  rti(db);                      /* return from interrupt, but execute next 2 lines */
052  r8 = lshift r8 by -16;        /* shift to the lowest 16 bits  */
053  dm(tx_buf + 1) = r8;          /* move the sample to the output */
```

После того как написана ассемблерная программа и спроектировано ядро фильтра, мы готовы создать программу, которая может быть исполнена на ЦПОС SHARC. Это осуществляется запуском *компилятора*, *ассемблера* и затем *линкера*; три программы

поставляемые с EZ-KIT Lite. Компилятор преобразует программу, написанную на ***C***, в язык ассемблера SHARC. Если Вы непосредственно пишите программу на ассемблере так, как в этом примере, Вы опускаете этот шаг. Ассемблер и линкер преобразуют программу и внешние файлы (такие как файл архитектуры, подпрограмма инициализации кодека, ядро фильтра и т.д.) в конечный исполняемый файл. Все это занимает приблизительно 30 секунд, причем конечным результатом будет программа для SHARC, расположенная на жестком диске Вашего ПК. Затем для запуска полученной исполняемой программы на EZ-KIT Lite, используется основная программа *EZ-Lite*. Просто щелкните мышкой по файлу, который Вы хотите запустить на ЦПОС, а основная программа *EZ-Lite* позаботится об остальном, загрузке программы и ее запуске.

Это вызывает у нас два вопроса. Во-первых, как мы проверим наш аудио фильтр, чтобы убедиться, что он функционирует так, как мы его проектировали; и, во-вторых, что в этом мире, производящем *цифровые процессоры обработки сигналов*, делает *компания*, называемая *Аналоговые устройства* (Analog Devices Inc.)?

Аналоговые измерения в системах ЦОС

Просто на некоторое время, забудьте, что Вы изучаете *цифровые* методы. Давайте посмотрим на них с точки зрения инженера, специализирующегося на аналоговой электронике. Ему все равно, что находится внутри EZ-KIT Lite, единственное, что у него есть - это аналоговый вход и аналоговый выход. Как показано на рис. 29.3 он будет взывать к традиционным аналоговым методам исследования “черного ящика”, присоединит к входу генератор сигналов и при помощи осциллографа посмотрит на выход.

Что же обнаружит наш аналоговый гуру? Во-первых, система является *линейной* (настолько, насколько может сказать этот простой метод). Если на вход подается синусоидальная волна, синусоидальная волна наблюдается и на выходе. Если амплитуда или частота на входе изменяется, соответствующие изменения наблюдаются и на выходе. Когда частота на входе медленно нарастает, то наступает момент, когда амплитуда синусоидальной волны на выходе быстро уменьшается до нуля. Это происходит чуть ниже одной второй от частоты дискретизации из-за действия фильтра антисовмещения перед АЦП.

Сейчас наш инженер заметил что-то неизвестное в аналоговом мире: система имеет безупречную *линейную фазу*. Другими словами между событием происходящем во входном сигнале и реакцией на это событие в выходном сигнале существует постоянная задержка. В частности, рассмотрим наш образец ядра фильтра на рис. 29.2. Поскольку центр симметрии располагается при отсчете с номером 150, выходной сигнал будет задержан на 150 отсчетов относительно входного сигнала. Если, например, дискретизация в системе осуществляется на 8 кГц, эта задержка будет составлять 18,75 миллисекунд. Кроме этого, сигма-дельта преобразователь будет тоже давать небольшую дополнительную фиксированную задержку.

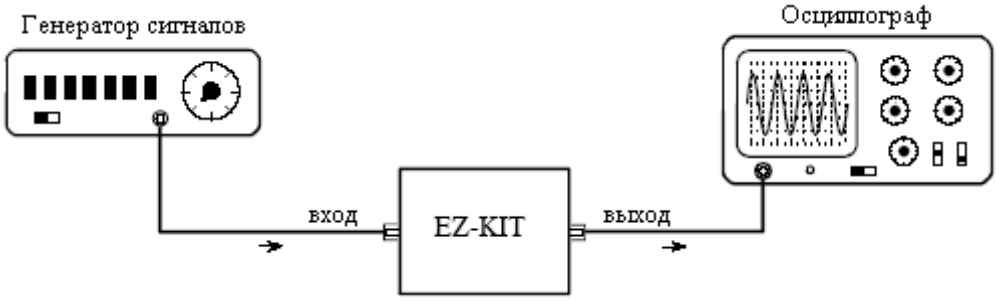


Рис. 29.3 Тестирование комплекта EZ-KIT Lite

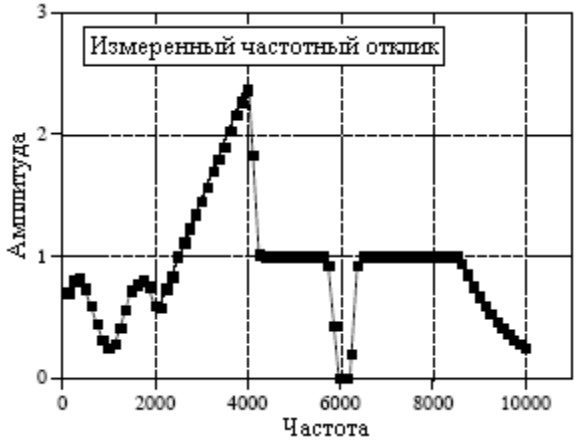


Рис. 29.4 Измеренная частотная характеристика

Наш аналоговый инженер будет очень взволнован, когда он увидит такую линейную фазу. Сигналы не будут появляться таким способом, каким он думал, они должны появляться, и он начнет крутить ручки с молниеносной скоростью. Он будет жаловаться, что плохо работает синхронизация, и бормотать такие слова как: "не понимаю", "что же здесь происходит?" и "кто поиграл с моим осциллографом?". Работа системы ЦОС так хороша, что пройдет несколько минут до того, как он поймет, что же он видит.

Для того чтобы произвести на него еще большее впечатление, мы попросили нашего инженера вручную измерить частотный отклик системы. Чтобы сделать это, он прогнал генератор сигналов через все частоты между 125 Гц и 10 кГц с шагом 125 Гц. На каждой частоте он измерял амплитуду выходного сигнала и делил ее на амплитуду входного сигнала. (Конечно же, наиболее простым способом сделать это является поддерживание амплитуды входного сигнала постоянной.) Для этого теста мы установили частоту дискретизации для EZ-KIT Lite равной 22 кГц. Другими словами, диапазон от 0 до 0,5 цифровой частоты на рис. 29.2а, что в измерении нашего реального мира соответствует диапазону от постоянной составляющей до 11 кГц.

На рис. 29.4 показаны действительные измерения, сделанные на EZ-KIT Lite; лучше и быть ничего не может! Данные измеренных точек согласуются с теоретической кривой в пределах погрешности измерений. Это что-то, чего наш аналоговый инженер *никогда* не видел у фильтров, сделанных из резисторов, конденсаторов и катушек индуктивностей.

Однако даже это не дает ЦОС форы, которой она заслуживает. Аналоговые измерения с использованием осциллографа и цифрового вольтметра имеют типичную достоверность и точность приблизительно от 0,1 до 1%. Для сравнения, поскольку внутренние вычисления выполняются с использованием плавающей запятой, данная система ЦОС ограничена только – 0,001% ошибкой округления 16 битового кодека. Другими словами, оцениваемое устройство является в *тысячу раз* более точным, чем используемое измерительное оборудование. Надлежащая оценка частотной характеристики потребовала бы специализированного инструмента такого, как компьютеризированная система сбора данных с 20 битовым АЦП. Располагая такими фактами, неудивительно, что ЦПОС часто используются в измерительных инструментах для достижения высокой точности.

Теперь мы можем ответить на вопрос: Почему *Аналоговые устройства* (Analog Devices) продают *цифровые процессоры обработки сигнала*? Всего десятилетие назад пышное искусство обработки сигналов воплощалось с помощью прецизионных операционных усилителей и подобных транзисторных схем. Сегодня высококачественная

аналоговая обработка заканчивается *цифровыми* методами. Analog Devices является грандиозным примером для подражания у отдельных лиц и у других компаний; придерживайтесь своих взглядов и целей, но не бойтесь адаптироваться к изменяющейся технологии!

Другой взгляд на фиксированную запятую по сравнению с плавающей

В этом последнем примере, мы использовали преимущество одной из ключевых характеристик ЦПОС SHARC, его способность управлять вычислениями с плавающей запятой. Даже притом, что при поступлении из кодека и обратно отсчеты находятся в формате с фиксированной запятой, для выполнения алгоритма промежуточной КИХ-фильтрации нам приходится преобразовывать их к формату с плавающей запятой. Как обсуждалось в предыдущей главе, существует две причины для того, чтобы желать обрабатывать данные с помощью математики с плавающей запятой: *простота программирования* и *рабочие характеристики*. Неужели это действительно имеет значение?

Для программиста да, это имеет большое значение. Коды с плавающей запятой легче записывать. Посмотрите назад, на ассемблеровскую программу в таблице 29.2. В основном КИХ-фильтре всего две строки (41 и 42). Напротив, программист, работающий с фиксированной запятой, должен добавить код управления данными в каждом математическом вычислении. Для того чтобы избежать переполнения и машинного нуля, должен быть проверен и если нужно соответственно смасштабирован размер значений. Во избежание разрушительных эффектов повторной ошибки округления, промежуточные результаты нужно будет также сохранить в аккумуляторе расширенной точности.

Вопросы рабочих характеристик намного более тонкие. Например, на рис. 29.5a показан низкочастотный КИХ-фильтр с умеренно крутым срезом, как описывалось в Главе16. Эта кривая "большого масштаба" выглядела бы точно также, использовалась ли бы в вычислении фиксированная или плавающая запятая. Для того чтобы увидеть разницу между этими двумя методами, мы должны изменить масштаб изображения по амплитуде в несколько сот раз так, как показано на рис. 29.5b, рис. 29.5 c, и рис. 29.5d. Здесь мы можем видеть явную разницу. Реализация с плавающей точкой, рис. 29.5b, имеет такой низкий шум округления, что ее характеристики ограничиваются только способом, которым мы сконструировали ядро фильтра. Выбросы в 0,02% вблизи перехода являются характеристикой используемого в этом фильтре окна Блэкмена. Дело в том, что если мы хотим улучшить характеристики, мы должны работать над *алгоритмом*, а не над его *исполнением*. Кривые на рис. 29.5c и рис. 29.5d показывают вносимый шум округления при представлении каждой точки с помощью 16 и 14 битов (речь идет о фиксированной запятой - прим ред. перев.), соответственно. Более хороший алгоритм не сделал бы ничего для того, чтобы эти кривые стали лучше; форма фактической частотной характеристики поглощается шумом.

На рис. 29.6 показана разница между фиксированной и плавающей запятой во *временной области*. На рис. 29.6a показан сигнал экспоненциально затухающих по амплитуде колебаний. Он может представлять, например, звуковые колебания от задетой струны или колебания земли от произведенного на расстоянии взрыва. Как и раньше, данная волна "большого масштаба" будет выглядеть точно так же, использовалась ли для представления отсчетов фиксированная или плавающая запятая. Для того чтобы увидеть разницу, мы должны изменить масштаб изображения по амплитуде так, как это показано на рис. 29.6b, рис. 29.6c и 29.6d. Как обсуждалось в Главе 3, появившееся квантование значительно больше, чем аддитивный случайный шум, ограничивающий распознаваемость небольших компонент в сигнале.

Такие различия характеристик между фиксированной и плавающей запятой часто не являются важными; например, они даже не могут быть замечены в сигналах "большого

масштаба" на рис. 29.5a и рис. 29.6a. Однако существуют некоторые приложения, где экстраординарная характеристика плавающей запятой полезна и может быть даже критической. Например, высококачественные бытовые аудио системы такие, как CD плееры, представляют сигнал с помощью 16 битов с фиксированной запятой. В большинстве случаев это превышает способности слуха человека. Однако лучшие профессиональные аудио системы оцифровывают сигналы от 20 до 24 битов, абсолютно не оставляя места ложным сигналам, которые могли бы загрязнять музыку. Плавающая запятая является почти идеальной для алгоритмов обрабатывающих высокоточные цифровые сигналы.

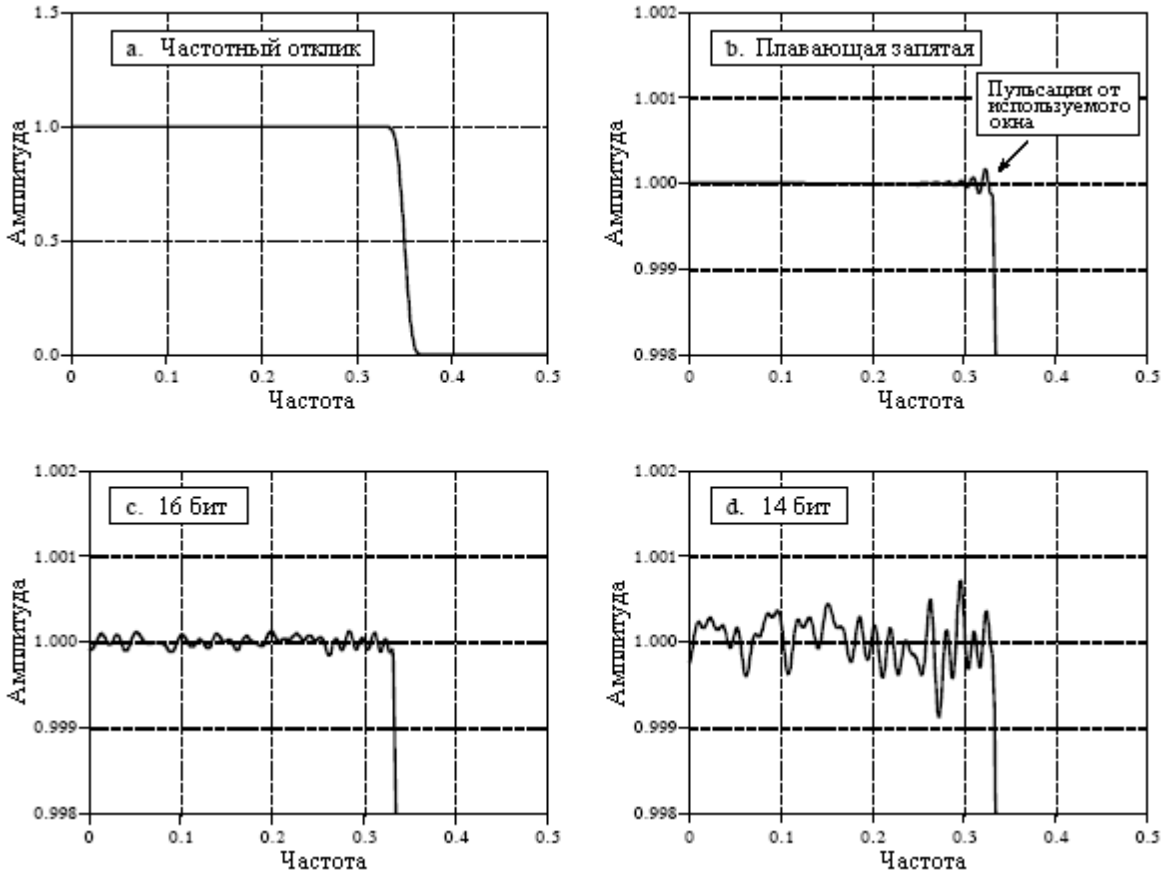


Рис. 29.5 Шум округления в частотной характеристике

Другим случаем, где требуются превосходные характеристики плавающей запятой, является случай, когда *алгоритм* особенно чувствителен к шуму. Например, КИХ-фильтры совсем нечувствительны к эффектам округления. Как показано на рис. 29.5, шум округления не изменяет общую форму частотного отклика; вся кривая становится только более зашумленной. БИХ фильтры - это совершенно другая история; здесь округление может стать причиной разрушений всех видов вплоть до приведения их к неустойчивости. Плавающая запятая позволяет этим алгоритмам достигать лучших параметров крутизны частоты среза, ослабления в полосе подавления и перерегулирования переходной характеристики.

Инструментарий передового программного обеспечения

Наш привычный пример фильтра показывает наиболее *простой* способ получить программу, исполняющуюся на ЦПОС SHARC: редактирование, ассемблирование, линкование и загрузка, выполненные с помощью индивидуальных программ. Этот метод

хорош для простых задач, но для программистов, находящихся в авангарде, существует доступный, более хороший инструментарий программного обеспечения. Давайте посмотрим на то, что доступно в тех случаях, когда Вы серьезно настроены на работу с ЦПОС.

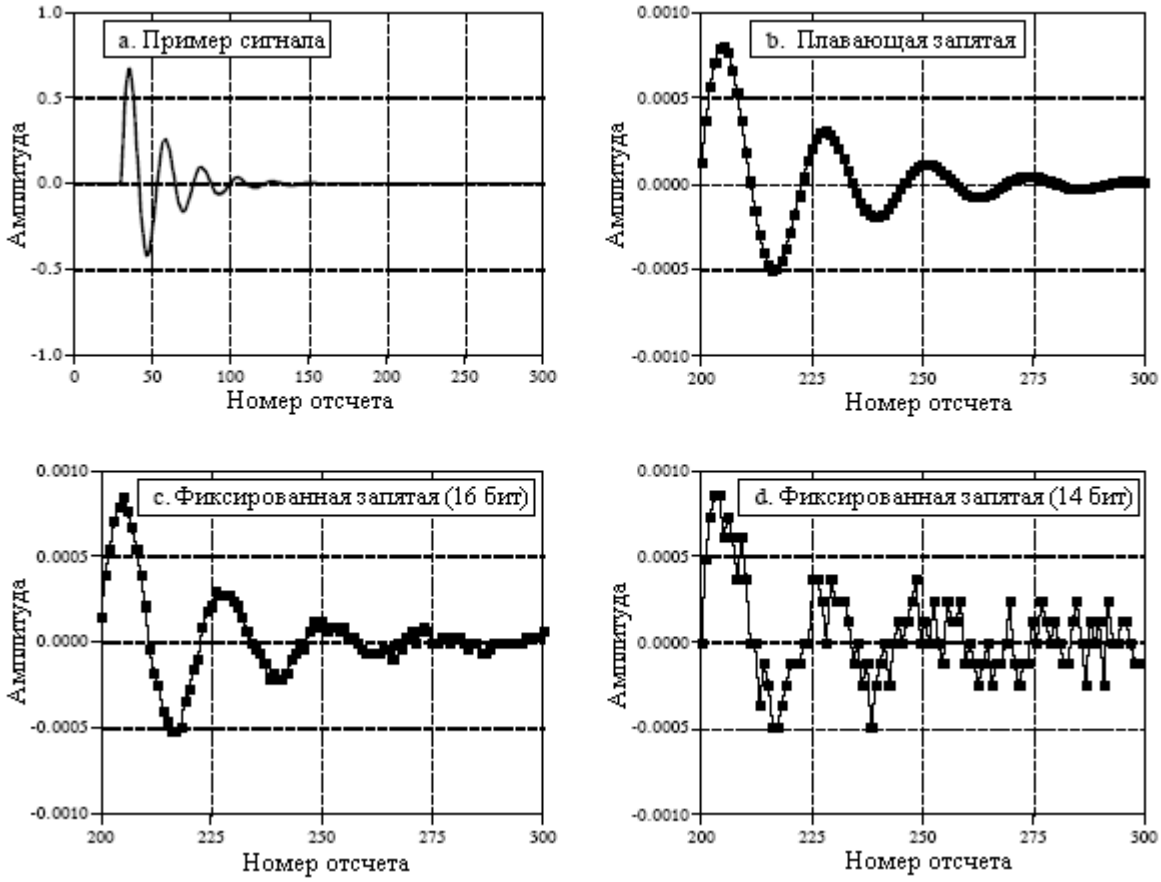


Рис. 29.6 Шум округления во временной области

Первым инструментом, который мы хотим пристально рассмотреть, является компилятор ***C***. Как обсуждалось в предыдущей главе, оба и ассемблер, и ***C*** обычно используются для программирования ЦПОС. Огромным преимуществом использования ***C*** является наличие библиотеки функций, как стандартных операций ***C***, так и алгоритмов ЦОС. В таблице 29.3 показан частичный список библиотеки функций ***C*** для ЦПОС SHARC. Математическая группа включает множество функций типичных для ЦОС таких, как тригонометрические (sin, cos, tan и т.д.), логарифмические и экспоненциальные. Если Вам в Ваших программах понадобятся функции этого вида, это само по себе может быть достаточной мотивацией для использования ***C*** вместо ассемблера. Обратите особое внимание на программы “обработки сигнала” в таблице 29.3. Здесь Вы найдете ключевые алгоритмы ЦОС, включая: действительное и комплексное БПФ, КИХ и БИХ фильтры, и статистические функции, такие как среднее значение, среднеквадратичное значение и выборочная дисперсия. Конечно, все эти программы написаны на ассемблере, что делает их очень эффективными, как по скорости, так и по использованию памяти.

В таблице 29.4 показан пример программы на ***C***, взятый из “Руководства по компилятору ***C*** и справочника рекомендаций” компании Analog Devices. Эта программа генерирует эхо за счет добавления задержанной версии сигнала к самому себе. Самые последние 4000 отсчетов из входного сигнала сохраняются в кольцевом буфере. По мере получения каждого отсчета кольцевой буфер обновляется, самый новый отсчет

добавляется к смасштабированной версии самого старого отсчета, и результирующий отсчет направляется на выход.

Таблица 29.3

МАТЕМАТИЧЕСКИЕ ОПЕРАЦИИ

abs	абсолютное значение
acos	арккосинус
asin	арксинус
atan	арктангенс
atan2	арктангенс частного
cabsf	комплексное абсолютное значение
cexpf	комплексная экспонента
cos	косинус
cosh	косинус гиперболический
cot	котангенс
div	деление
exp	экспонента
fmod	модуль
log	натуральный логарифм
log10	логарифм по основанию 10
matadd	суммирование матриц
matmul	умножение матриц
pow	вычислить мощность
rand	генератор случайных чисел
sin	синус
sinh	синус гиперболический
sqrt	корень квадратный
srand	затравка случайного числа
tan	тангенс
tanh	тангенс гиперболический

УПРАВЛЕНИЕ ПРОГРАММОЙ

abort	ненормальное завершение программы
calloc	распределение/инициализация памяти
free	освобождение памяти
idle	холостая команда процессора
interrupt	определение обработки прерывания
poll_flag_in	проверить входной флаг
set_flag	установить флаги процессора
timer_off	блокировать таймер процессора
timer_on	деблокировать таймер процессора
timer_set	инициализировать таймер процессора

МАНИПУЛЯЦИЯ СИМВОЛАМИ И СТРОКМИ

atoi	преобразование строки в целое
bsearch	двоичный поиск массива
isalnum	обнаружение буквенно-цифрового символа
isalpha	обнаружение буквенного символа
iscntrl	обнаружение управляющего символа
isdigit	обнаружение десятичной цифры
isgraph	обнаружение графического символа
islower	обнаружение символа нижнего регистра
isprint	обнаружение печатного символа
ispunct	обнаружение символа пунктуации
isspace	обнаружение символа пробела
isupper	обнаружение символа верхнего регистра
isxdigit	обнаружение шестнадцатеричной цифры
memchr	найти первый встретившийся символ
memcpy	копировать символ
strcat	конкатенация строк
strcmp	сравнить строки
strerror	получить сообщение об ошибке
strlen	длина строки
strncmp	сравнить символы
strrchr	найти последний встретившийся символ
strstr	найти строку в строке
strtok	преобразовать строку в лексемы
system	отправить строку в операционную систему
tolower	заменить верхний регистр нижним
toupper	заменить нижний регистр верхним

ОБРАБОТКА СИГНАЛА

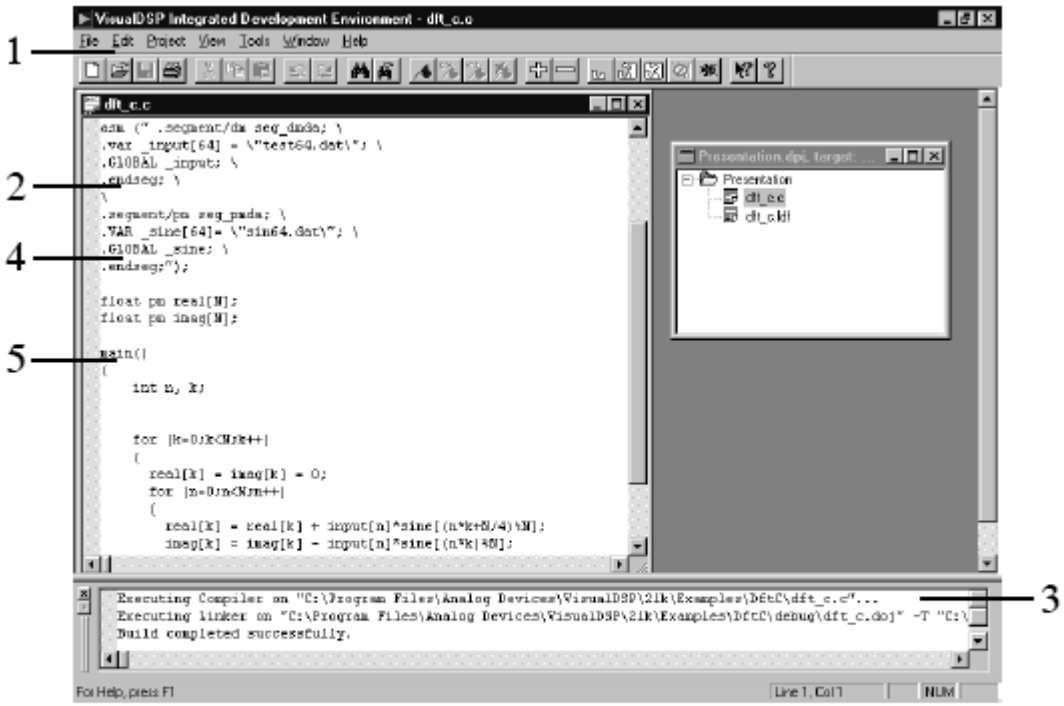
a_compress	А-закон сжатия
a_expand	А-закон растяжения
autocorr	автокорреляция
biquad	биквадратная секция фильтра
cfftN	комплексное БПФ
crosscorr	взаимная корреляция
fir	КИХ-фильтр
histogram	гистограмма
iffN	обратное комплексное БПФ
iir	БИХ-фильтр
mean	среднее значение массива
mu_compress	мю-закон сжатия
mu_expand	мю-закон растяжения
rfftN	действительное БПФ
rms	среднеквадратичное значение массива

Таблица 29.4

```
/*     CIRCBUF.C                                                          */
/*     This is an echo program written in C for the ADSP-21061 EZ-KIT Lite.  The  */
/*     echo program takes the talkthru program and adds a circular buffering scheme.      */
/*     The circular buffer is defined by the functions CIRCULAR_BUFFER, BASE,             */
/*     and LENGTH.  The echo is performed by adding the current input to the oldest       */
/*     input. The delay in the echo can be modified by changing BUFF_LENGTH.    */
/*                                                                                        */
/*                                                                                        */
#include <21020.h>          /*  for the idle() command  */
#include <signal.h>         /*  for the interrupt command  */
#include <macros.h>         /*  for the CIRCULAR_BUFFER and segment functions  */

#define BUFF_LENGTH 4000
                                        /*  define echo as 21k DAG1 reg i1  */
CIRCULAR_BUFFER (float,1,echo)          /*  a DM pointer to a circular buffer  */

volatile float in_port segment (hip_reg0);      /*  hip_reg0 and hip-reg2 are  */
volatile float out_port segment (hip_reg2);     /*  used in the architecture file  */

void process_input (int);
void main (void)

{       /*  Make this a variable length array.  If emulator is stopped at main        */
        /*  and _BUFF_LENGTH in dm window is modified, the echo delay                  */
        /*  is modified.  Do not make BUFF_LENGTH greater than stack size!             */

float data_buff [BUFF_LENGTH];
interrupt (SIG_IRQ3, process_input);

BASE (echo) = data_buff;                /*  Loads b1 and i1 with buff start adr  */
LENGTH (echo) = BUFF_LENGTH;            /*  Loads L1 with the length of the buffer */

        /*  as the array is filled, the nth location contains the newest value, while  */
        /*  the nth + 1 location contains the oldest value.  */

while (1)
  {                                     /*  the echo sends the sum of the most  */
  float oldest, newest;                 /*  recent value and the oldest value  */
  idle();

        /*  Echo is pointing to the nth location after the interrupt routine.     */
        /*  Place the new value in variable 'newest'.  After the access, update   */
        /*  the pointer by one to point at location n+1.                          */
  CIRC_READ (echo, 1, newest, dm);

        /*  Now echo is pointing to n+1.  Read the location and place value in    */
        /*  variable 'oldest'.  Do not update the pointer, since it is now        */
        /*  pointing to the new location for the interrupt handler.               */
  CIR_READ (echo, 0, oldest, dm);

        /*  add the oldest and most recent and send out on port                   */
  out_port=oldest+newest;
  }
}

void process_input (int int_number)
{
        /* The newest input value is written over the oldest value in the nth  */
        /*  location and the pointer is not updated.                           */
  CIRC_WRITE (echo, 0, in_port, dm);
}
```

Следующим продвинутым продуктом, который вам следует искать, является **интегрированная среда разработки**. Этот причудливый термин означает, что все

необходимое для программирования и тестирования ЦПОС объединено в один слаженно функционирующий пакет. Analog Devices снабжает интегрированной средой разработки продукт под названием VisualDSP®, работающий под управлением Windows® 95 и Windows NT™. На рис. 29.7 показан пример основного окна обеспечивающего непрерывный способ редактирования, создания и отладки программ.



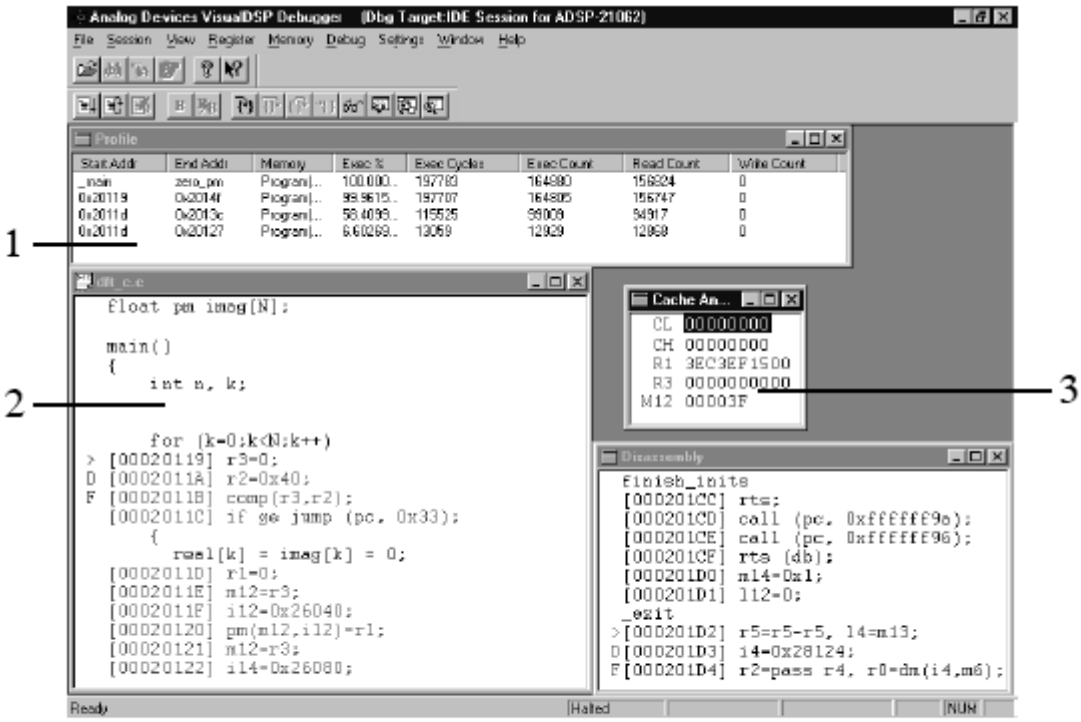
1. Легкий переход между редактированием, созданием и отладкой
2. Смешивание ассемблера и C в общем исходном файле
3. Наблюдаемые "результаты" создания
4. Мощный редактор понимающий синтаксис
5. Свободный доступ через закладки

Рис. 29.7 Пример окна VisualDSP

Вот некоторые из ключевых особенностей VisualDSP, показывающие, почему интегрированная среда разработки так важна для быстрой разработки программного обеспечения. Редактор специализирован для создания программ на ***C***, ассемблере, и смеси из них обоих. Например, он понимает *синтаксис* языков, что позволяет выводить на дисплей различные виды утверждений различными цветами. Вы можете также редактировать более одного файла одновременно. Это очень удобно, так как заключительная программа создается линкованием нескольких файлов вместе.

На рис. 29.8 показан пример окна отладчика VisualDSP. Это интерфейс для инструментов двух разных видов: симуляторов и эмуляторов. **Симуляторы** тестируют код для ЦПОС на *персональном компьютере* без необходимости наличия самого ЦПОС. Это обычно является первым шагом отладки после того, как написана программа. Симулятор копирует архитектуру и операции ЦПОС, включая: потоки входных данных, прерывания и другой ввод/вывод. **Эмуляторы** (такие как Analog Devices EZ-ICE®) проверяют работу программы на *действительном оборудовании*. Это требует наличия программного обеспечения эмулятора (на Вашем персональном компьютере) для того, чтобы иметь возможность контролировать электрические сигналы *внутри* ЦПОС. Поддерживающей данное особенностью ЦПОС SHARC является тестовый порт доступа IEEE 1140.1 JTAG, позволяющий внешним приборам отслеживать внутренние функции процессора.

После того, как Вы поработали с оценочным комплектом и подумали о покупке продвинутого программного инструментария, Вам следует также рассмотреть вопрос и о посещении подготовительных курсов. Они проводятся многими производителями ЦПОС. Например, Analog Devices предлагает 3 дневных курса, которые проводятся несколько раз в год в нескольких разных местах. Это огромная возможность узнать о ЦПОС непосредственно от экспертов. Для уточнения деталей загляните на вебсайты производителей.



1. Код профиля для идентификации узких мест
2. Смешанный вид распечатки С и ассемблера
3. Созданное окно обычных регистров

Рис. 29.8 Окно отладчика VisualDSP

ГЛАВА
30

КОМПЛЕКСНЫЕ ЧИСЛА

Комплексные числа являются расширением обычных чисел, используемых в ежедневной математике. Они обладают уникальным свойством представления и манипулирования *двумя* переменными, как *единой* количественной величиной. Это очень естественно подходит к Фурье анализу, где частотная область складывается из двух сигналов, действительной и мнимой частей. Комплексные числа делают уравнения, используемые в ЦОС, более короткими, а также делают доступными методы, которые трудны или вообще не возможны при использовании только действительных чисел. Например, Быстрое Преобразование Фурье базируется на комплексных числах. К сожалению, комплексные методы чересчур насыщены математикой, и для их эффективного использования требуется значительное обучение и практика. Многие ученые и инженеры рассматривают, в связи с этим, комплексные методы как барьер между ЦОС как *инструментом*, и ЦОС, как успехом в *карьере*. В этой главе мы рассмотрим математику комплексных чисел и элементарные приемы использования их в науке и инженерном деле. В трех последующих главах будут обсуждаться важные методы, базирующиеся на комплексных числах: *комплексное преобразование Фурье*, *преобразование Лапласа* и *z-преобразование*. Эти комплексные преобразования – сердце теории ЦОС. Готовьтесь, сюда идет математика!

Система комплексного числа

Для иллюстрации комплексных чисел посмотрим на ребенка, подбрасывающего в воздух мяч. Например, предположим, что мяч подброшен вертикально с начальной скоростью 9,8 метра в секунду. Секундой позже, после того как он покинул руки ребенка, мяч достигнет высоты 4,9 метра, а ускорение свободного падения (9,8 метра на секунду^2) снизит его скорость до нуля. Затем мяч ускоряется к земле и спустя две секунды, после того как он был брошен, ловится ребенком. Из элементарных уравнений физики высота мяча в любой момент времени определяется как:

$$h = \frac{-gt^2}{2} + vt,$$

где h – высота над землей (в метрах), g – ускорение свободного падения (9,8 метра на секунду^2), v – начальная скорость (9,8 метра в секунду) и t – время (в секундах).

А сейчас, предположим, что мы хотим узнать, *когда* мяч пройдет определенную высоту. Подставляя известные значения и решая это уравнение относительно t, получим:

$$t = 1 \pm \sqrt{1 - h/4,9}.$$

Например, мяч находится на высоте трех метров *дважды*: *t=0,38* (движется вверх) и *t=1,62* секунды (движется вниз).

Пока мы задаем разумные вопросы, эти уравнения дают разумные ответы. Но что происходит, когда мы задаем неблагоразумные вопросы? Например: В какой момент времени мяч достигнет высоты *10* метров? В действительности на этот вопрос нет ответа, поскольку мяч *никогда* не достигает такой высоты. Тем не менее, подстановка значения *h=10* в вышеуказанное уравнение дает два ответа: $t = 1 + \sqrt{-1,041}$ и $t = 1 - \sqrt{-1,041}$. Оба эти ответа содержат корень квадратный из отрицательного числа, содержат что-то, чего, как нам известно, в мире не существует. Это необычное свойство полиномиальных уравнений впервые было использовано итальянским математиком Гироламо Кардано (1501-1576). Двумя веками позже великий немецкий математик Карл Фридрих Гаусс (1777-1855) ввел термин **комплексные числа** и проложил путь для современного понимания этой области.

Каждое комплексное число представляет собой сумму двух компонент: **действительной части** и **мнимой части**. Действительная часть – это **действительное число**, одно из обычных чисел, которые мы учили в детстве. Мнимая часть – это мнимое число, то есть *корень квадратный из отрицательного числа*. Чтобы привести все к стандартному виду, обычно мнимая часть приводится к обыкновенному числу, умноженному на корень квадратный из минус единицы. В качестве примера, комплексное число $t = 1 + \sqrt{-1,041}$ первоначально приводится к $t = 1 + \sqrt{1,041}\sqrt{-1}$, а затем к окончательному виду $t = 1 + 1,02\sqrt{-1}$. Действительная часть этого комплексного числа - *1*, тогда как мнимая часть - $1,02\sqrt{-1}$. Такое обозначение позволяет присвоить абстрактному члену $\sqrt{-1}$ специальный символ. Математики для обозначения $\sqrt{-1}$ уже давно используют символ *i*. Для сравнения, инженеры-электрики используют символ *j*, потому что *i* используется для представления электрического тока. Для ЦОС типичны оба символа. В этой книге будет использоваться *j* – соглашение инженеров-электриков.

Например, все следующее - это законные комплексные числа: *1+2j*, *1-2j*, *-1+2j*, *3,14159+2,7183j*, *(4/3)+(19/2)j* и т.д. Все обычные числа, такие как *2*, *6,34* и *–1,414* могут рассматриваться как комплексные числа с *нулевой* мнимой частью, т.е. *2+0j*, *6,34+0j* и *–1,414+0j*.

Подобно тому, как действительные числа описываются позицией на числовой оси, комплексные числа представляются местом в двумерном пространстве, называемом **комплексной плоскостью**. Как показано на рис. 30.1, горизонтальная ось комплексной плоскости является действительной частью комплексного числа, в то время как вертикальная ось является мнимой частью. Поскольку действительные числа - это те комплексные числа, мнимая часть которых равна нулю, *действительная числовая ось* - это то же самое, что и *ось x* комплексной плоскости.

Хотя комплексное число состоит из двух частей, в математических уравнениях оно представляется одной переменной. Например, три комплексных переменных на рис. 30.1 могут быть записаны как:

$$A = 2 + 6\,j;$$
$$B = -4 - 1,5\,j;$$
$$C = 3 - 7\,j,$$

где *A*, *B* и *C* – комплексные переменные. Это иллюстрирует сильное *преимущество* и сильный *недостаток* использования комплексных чисел. Преимущество - это присущая стенографичность представления двух вещей одним символом. Недостаток – это

необходимость помнить, какие переменные являются комплексными, а какие переменные обычными числами.

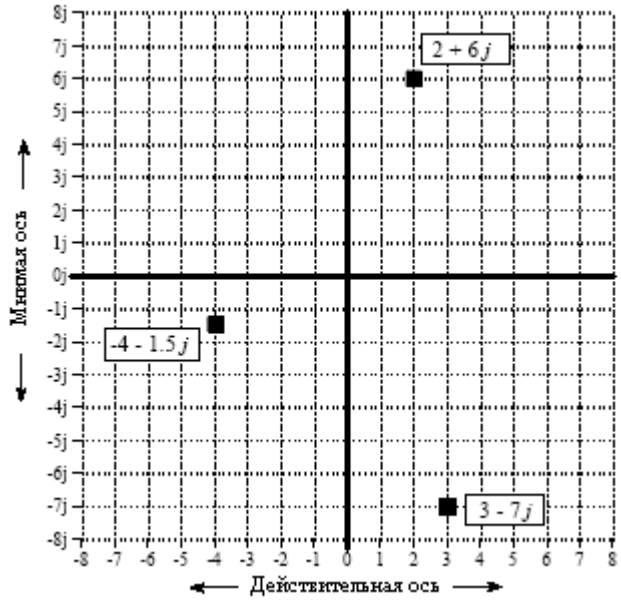


Рис. 30.1 Комплексная плоскость

Математическая система обозначений для разделения комплексной переменной на её действительную и мнимую части использует операторы: *Re*( ) и *Im*( ). Например, используя вышеупомянутые комплексные числа, получим:

$$\mathrm{Re}\,A = 2 \qquad \mathrm{Im}\,A = 6\,;$$

$$\mathrm{Re}\,B = -4 \qquad \mathrm{Im}\,B = -1{,}5\,;$$

$$\mathrm{Re}\,C = 3 \qquad \mathrm{Im}\,C = -7\,.$$

Обратите внимание, что значение, возвращаемое математическим оператором *Im*( ) не содержит *j*. Например, *Im*(*3+4j*) эквивалентно *4*, а не *4j*.

Комплексные числа придерживаются той же алгебры, что и обычные числа, рассматривая величину *j* как константу. Например, сложение, вычитание, умножение и деление определяются следующим образом:

$$(a+bj)+(c+dj)=(a+c)+j(b+d) \tag{30.1}$$

$$(a+bj)-(c+dj)=(a-c)+j(b-d) \tag{30.2}$$

$$(a+bj)\cdot(c+dj)=(ac-bd)+j(bc+ad) \tag{30.3}$$

$$\frac{(a+bj)}{(c+dj)}=\left(\frac{ac+bd}{c^2+d^2}\right)+j\left(\frac{bc-ad}{c^2+d^2}\right). \tag{30.4}$$

При манипулировании уравнениями, подобными этим, пользуются двумя уловками. Первая, всякий раз, когда сталкиваются с членом j^2, он заменяется на -1. Это следует из определения j, согласно которому: $j^2 = \left(\sqrt{-1}\right)^2 = -1$. Вторая уловка – это способ избавиться от j в знаменателе дроби. Например, знаменателем левой части уравнения 30.4 является $c+dj$. Устранение всех мнимых членов из знаменателя реализуется умножением числителя и знаменателя на член $c-dj$. На жаргоне данной области изменение знака мнимой части комплексного числа называется получением **комплексно сопряженного** числа. Такое действие обозначается звездочкой в верхнем правом углу переменной. Например, если $Z=a+bj$, тогда $Z^*=a-bj$. Другими словами, уравнение 30.4 получено умножением и числителя и знаменателя на число, комплексно сопряженное знаменателю.

Даже тогда, когда переменные A, B и C комплексные, сохраняются следующие свойства.

$$AB = BA \qquad \text{(свойство коммутативности)} \qquad (30.5)$$

$$(A+B)+C = A+(B+C) \qquad \text{(свойство ассоциативности)} \qquad (30.6)$$

$$A(B+C) = AB + AC \qquad \text{(свойство дистрибутивности)} \qquad (30.7)$$

Данные соотношения могут быть доказаны разделением каждой переменной на ее действительную и мнимую части и выполнением алгебраических преобразований.

Полярная система обозначений

Помимо только что описанной *прямоугольной системы обозначений*, комплексные числа могут быть также выражены в *полярной системе обозначений*. Например, на рис. 30.2 в полярной форме показаны три комплексных числа, те же самые, что первоначально были представлены на рис. 30.1. Длина вектора, начинающегося в начале координат и заканчивающегося в комплексной точке, это **модуль**, тогда как **фазовый угол** измеряется между этим вектором и положительным направлением оси x. В соответствии со следующими уравнениями, можно осуществлять преобразования комплексных чисел из прямоугольной системы в полярную и обратно (обратите внимание на *неудобства* полярной системы которые обсуждались в Главе 8):

$$M = \sqrt{(\mathrm{Re}\,A)^2 + (\mathrm{Im}\,A)^2}$$

$$\Theta = \arctan\left[\frac{\mathrm{Im}\,A}{\mathrm{Re}\,A}\right]$$

(преобразование из прямоугольной системы в полярную) (30.8)

$$\mathrm{Re}\,A = M\cos\Theta$$

$$\mathrm{Im}\,A = M\sin\Theta$$

(преобразование из полярной системы в прямоугольную) (30.9)

Эти преобразования приводят к гигантскому скачку в математике. (Да-да, это означает, что Вы должны уделить им исключительно особое внимание.) Комплексное число, записанное в прямоугольной системе обозначений, – это число в форме $a+bj$.

Информацию несут переменные *a* и *b*, но собственно комплексное число - это все выражение *a+bj*. В полярной форме ключевая информация содержится в *M* и *Θ*, а что представляет собой все выражение для собственно комплексного числа?

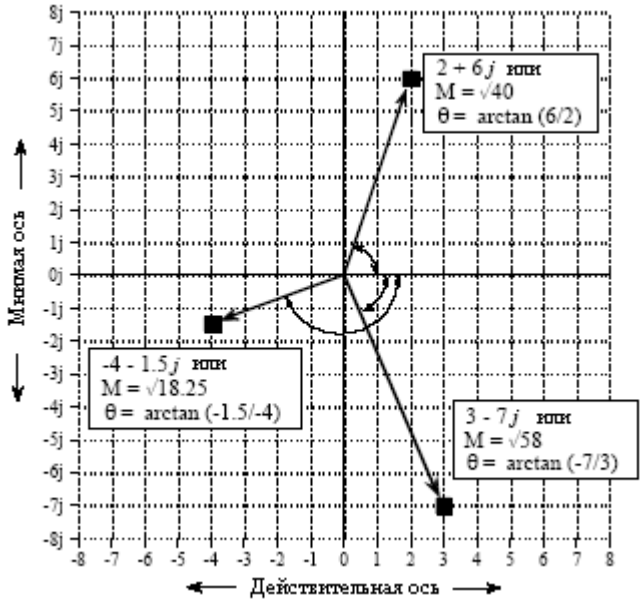


Рис. 30.2 Комплексные числа в полярной форме

Ключом к ответу является уравнение 30.9, преобразование из полярной системы в прямоугольную. Если мы начнем с собственно комплексного числа *a+bj* и применим выражение 30.9, мы получим:

$$a + jb = M\left(\cos\Theta + j\sin\Theta\right). \qquad (30.10)$$

Выражение слева - собственно *прямоугольное* описание комплексного числа, в то время как выражение справа - собственно *полярное* описание.

Прежде чем продолжить и сделать следующий шаг, давайте посмотрим, как мы добрались до этой точки. Первое, мы дали прямоугольной форме комплексного числа графическое представление, т.е. место на двумерной плоскости. Второе, мы определили термины *M* и *Θ* в соответствии с нашими предыдущими представлениями относительно взаимосвязи между полярными и прямоугольными координатами (уравнения 30.8 и 30.9). Третье, мы следовали за математическими результатами этих действий, подходя к тому, какой должна быть корректная полярная форма комплексного числа, т.е. *M(cosΘ+jsinΘ)*. Даже притом, что эта логика строга, "интуитивно" результат трудно понять. К сожалению, понимание стало хуже.

Одно из наиболее важных уравнений в комплексной математике это **формула Эйлера**, названная в честь умнейшего и очень плодотворного швейцарского математика Леонарда Эйлера (1707-1783) (В 1726 году Леонард Эйлер был приглашен в Петербургскую АН и в 1727 году переехал в Россию. Он начал работу в звании адъюнкта, а в 1731 году стал действительным членом Петербургской Академии Наук. С 1741 по 1766 годы Эйлер работал в Берлине, член Берлинской АН. В 1766 году Эйлер возвратился в Россию, где вокруг него сформировалась знаменитая Петербургская математическая школа. – прим. перев.):

$$e^{jx} = \cos x + j\sin x\,. \qquad (30.11)$$

Это соотношение, если Вам нравятся подобные вещи, может быть доказано разложением экспоненциального члена в ряд Тейлора:

$$e^{jx} = \sum_{n=0}^{\infty} \frac{(jx)^n}{n!} = \left[\sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} \right] + j \left[\sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} \right].$$

Два члена этого выражения справа, взятые в квадратные скобки - это ряды Тейлора для *cos(x)* и *sin(x)*. Не тратьте на это доказательство слишком много времени, мы не собираемся его для чего-нибудь использовать.

Переписывание уравнения 30.10 с использованием результата формулы Эйлера приводит в наиболее общем случае к выражению комплексного числа в полярной системе обозначений, **комплексной экспоненте**:

$$a + jb = Me^{j\Theta}. \qquad (30.12)$$

Комплексные числа в такой экспоненциальной форме – основа математики ЦОС. Начните свое понимание этой математики с запоминания уравнений от 30.8 до 30.12. Сильное преимущество использования такой экспоненциальной полярной формы заключается в том, что она очень проста для выполнения умножения и деления комплексных чисел:

$$M_1 e^{j\Theta_1} M_2 e^{j\Theta_2} = M_1 M_2 e^{j(\Theta_1 + \Theta_2)} \quad \text{(умножение комплексных чисел)} \quad (30.13)$$

$$\frac{M_1 e^{j\Theta_1}}{M_2 e^{j\Theta_2}} = \left[\frac{M_1}{M_2} \right] e^{j(\Theta_1 - \Theta_2)} \quad \text{(деление комплексных чисел)} \quad (30.14)$$

То есть, комплексные числа в полярной форме умножаются перемножением их модулей и сложением их углов. Наиболее простой путь выполнить сложение и вычитание в полярной форме - преобразовать числа в прямоугольную форму, выполнить операцию и произвести обратное преобразование в полярную форму. В прямоугольной форме комплексные числа обычно представляются в компьютерных программах, а в полярной при написании и манипулировании уравнениями. Так же, как *Re*( ) и *Im*( ) используются для извлечения прямоугольных компонент из комплексной переменной, операторы *Mag*( ) и *Phase* ( ) используются для извлечения полярных компонент. Например, если $A=5e^{j\pi/7}$, тогда *Mag(A)=5* и *Phase(A)=π/7*.

Использование комплексных чисел методом подстановки

Давайте подведем итог, к чему мы пришли. Решения общих алгебраических уравнений часто содержат корень квадратный из отрицательного числа. Они называются *комплексными числами* и представляют решения, которые, как мы это знаем, в мире существовать не могут. Комплексные числа выражаются в одной из двух форм: *a+bj* (прямоугольная) или $Me^{j\Theta}$ (полярная), где *j* это символ представляющий собой $\sqrt{-1}$. В любой системе обозначений отдельное комплексное число содержит две разных части информации: или *a* и *b*, или *M* и *Θ*. Несмотря на их иллюзорную природу, комплексные числа подчиняются математическим законам, которые подобны (или идентичны) тем, что управляют обычными числами.

Здесь же следует описание того, что такое комплексные числа и как они вписываются в мир чистой математики. Наша следующая задача состоит в том, чтобы описать пути, которые делают комплексные числа полезными в науке и при решении технических проблем. Как это возможно, использовать математику, которая не имеет никакой связи с нашим ежедневным жизненным опытом? Ответ будет следующим: *Если инструмент*, *который у нас есть – молоток*, *сделайте задачу похожей на гвоздь*. Другими словами, мы *изменяем* физическую задачу до формы комплексного числа, манипулируем комплексными числами, а затем делаем *обратные изменения*, назад к физическому ответу.

Существует два способа представления физической задачи с помощью комплексных чисел: простой метод **подстановки** и более элегантный метод, который мы будем называть **математической эквивалентностью**. Математическая эквивалентность будет обсуждаться в следующей главе о *комплексном преобразовании Фурье*. Остаток этой главы посвящен методу подстановки.

В методе подстановки берутся два действительных физических параметра и помещаются: один в действительную часть комплексного числа, а другой в мнимую часть. Это позволяет манипулировать двумя величинами как одним объектом, т.е. одним комплексным числом. После выполнения желаемых математических операций комплексное число разделяется на его действительную и мнимую части, которые снова соответствуют, интересовавшим нас, физическим параметрам.

Как все это работает, покажет простой пример. Как Вы помните из элементарной физики, *векторы* могут представлять такие вещи, как: силу, скорость, ускорение и т.д. Например, представьте парусную шлюпку, которую в одном направлении несет ветром, а в другом направлении океанским течением. Результирующая сила, действующая на шлюпку, является векторной суммой двух индивидуальных векторов сил. Этот пример показан на рис. 30.3, где два вектора $\boldsymbol{A}$ и $\boldsymbol{B}$ суммируются по правилу параллелограмма, давая в результате вектор $\boldsymbol{C}$.

Мы можем представить эту задачу с помощью комплексных чисел, размещая координату восток/запад в действительную часть комплексного числа, а координату север/юг в мнимую часть. Это позволяет нам обращаться с каждым вектором, как с одним комплексным числом, даже притом, что оно состоит из двух частей. Например, сила ветра, вектор $\boldsymbol{A}$, могла бы иметь направление 2 части на восток и 6 частей на север, и быть представлена, как комплексное число: $2+6j$. Аналогично, сила океанского течения, вектор $\boldsymbol{B}$, могла бы быть в направлении 4 части на восток и 3 части на юг, и быть представлена, как комплексное число: $4-3j$. Эти два вектора могут быть сложены посредством уравнения 30.1, дающего в результате комплексное число, представляющее вектор $\boldsymbol{C}$: $6+3j$. Выполняя обратные преобразования к физическим величинам, найдем, что общая сила, действующая на шлюпку, имеет направление 6 частей на север и три части на восток.

Могла ли эта задача быть решена без комплексных чисел? Конечно! Комплексные числа просто обеспечивают формализованный способ сразу отслеживать *две* компоненты, формирующие *один* вектор. Необходимо всегда помнить, что некоторые физические задачи могут быть преобразованы в комплексную форму простым добавлением j к одной из компонент. Обратное преобразование к физическим величинам не более чем удаление j. В этом суть метода *подстановки*.

А вот и камень преткновения. Как мы определим, что правила и законы, применяемые в комплексной математике, точно такие же, как правила и законы, применяемые в исходной физической задаче? Например, мы использовали уравнение 30.1 для сложения векторов сил в задаче со шлюпкой. Как мы узнали, что сложение комплексных чисел обеспечивает тот же самый результат, что и сложение векторов сил? В большинстве случаев то, что комплексная математика может быть использована для конкретных приложений, мы знаем потому, что *кто-то другой* сказал, что для этих конкретных приложений она может быть применима. Какой-нибудь блестящий математик

или пользующийся авторитетом инженер разработал детали метода и опубликовал результаты. Следует помнить, что мы не можем просто подставлять *любую* задачу в комплексную форму и ожидать того, что ответ будет иметь смысл. Мы должны придерживаться приложений, для которых применимость комплексного анализа уже была показана.

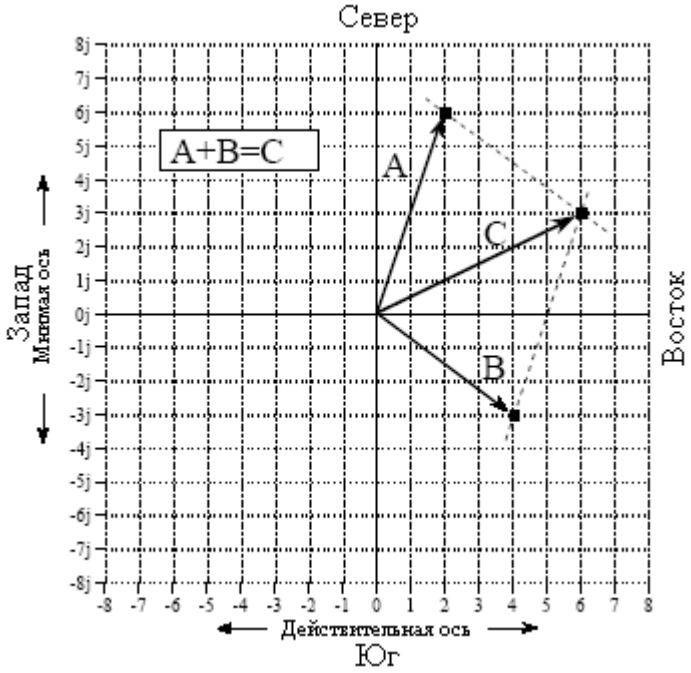


Рис. 30.3 Суммирование векторов с помощью комплексных чисел

Давайте посмотрим на пример, где метод подстановки комплексного числа *не работает*. Представьте, что Вы покупаете яблоки по 5 долларов за ящик и апельсины по 10 долларов за ящик. Вы представляете это с помощью комплексного числа $5+10j$. В течение определенной недели Вы покупаете 6 ящиков яблок и два ящика апельсин, которые Вы представляете комплексным числом $6+2j$. Полная стоимость, которую Вы должны заплатить за продукты, равна количеству единиц товара умноженному на стоимость одной единицы, т.е. $(5+10j)\times(6+2j)=10+70j$. Другими словами, комплексная математика показывает, что Вы, в общем, должны заплатить 10 долларов за яблоки и 70 долларов за апельсины. Дело в том, что ответ полностью неверный! Правила комплексной математики не соответствуют правилам этой конкретной физической задачи.

Комплексное представление синусоид

Комплексные числа находят нишу в электронике и обработке сигналов, поскольку они являются компактным способом представления и манипулирования наиболее используемыми из всех волн - синусоидальной и косинусоидальной волнами. Традиционный способ представления синусоиды следующий: $A\cos(\omega t)+B\sin(\omega t)$ или $M\cos(\omega t+\varphi)$, соответственно, в прямоугольной и полярной системах обозначения. Обратите внимание, что мы представляем частоту символом ω – *циклическая частота* в *радианах в секунду*. Если это будет для Вас более удобным, Вы можете заменить каждую ω на $2\pi f$, чтобы выражение было в герцах. Однако большая часть математики в ЦОС написана с использованием более коротких обозначений, и Вам с ними, следует стать более фамильярными. Использование комплексных чисел для представления этих важных форм волн естественно, поскольку для представления одной синусоиды требуется два параметра (т.е. A и B или M и φ). Переход от традиционного представления синусоиды к

комплексному числу осуществляется непосредственно, используя метод подстановки. Так, в прямоугольной форме:

$$A\cos(\omega t) + B\sin(\omega t) \quad \overset{\rightarrow}{\leftarrow} \quad a + jb,$$

(традиционное представление) (комплексное число)

где $A \overset{\rightarrow}{\leftarrow} a$ и $B \overset{\rightarrow}{\leftarrow} -b$. Выражаясь словами, амплитуда косинусной волны становится действительной частью комплексного числа, в то время как амплитуда синусной волны, взятая с *обратным знаком*, становится мнимой частью. Важно понимать, что это *не* уравнение, а просто путь, позволяющий комплексному числу *представлять* синусоиду. Такая подстановка может также быть выражена и в полярной форме:

$$M\cos(\omega t + \varphi) \quad \overset{\rightarrow}{\leftarrow} \quad Me^{j\Theta},$$

(традиционное представление) (комплексное число)

где $M \overset{\rightarrow}{\leftarrow} M$ и $\Theta \overset{\rightarrow}{\leftarrow} -\varphi$. Выражаясь словами, метод подстановки в полярной системе обозначений оставляет модуль без изменений, но меняет знак фазового угла.

Почему меняют знак мнимой части и фазового угла? Это делается для того, чтобы подстановка приобрела ту же форму, что и у *комплексного преобразования Фурье*, описанного в следующей главе. От такого изменения знака, методы *подстановки* этой главы ничего не приобретают, и это почти всегда делается для того, чтобы обеспечивать совместимость с другими более превосходящими методами (знак меняют потому, что изменение знака происходит при разложении $\cos(\alpha + \beta)$ – прим. перев.).

Использование комплексных чисел для представления синусоидальных и косинусоидальных волн является обычной техникой анализа электрических цепей и ЦОС. Это связано с тем, что большое число (но не все) правил и законов, которым подчиняются комплексные числа точно такие же, как и те, которым подчиняются синусоиды. Другими словами мы можем представить синусную и косинусную волны комплексными числами, различным образом манипулировать числами и получить окончательный ответ, соответствующий поведению синусоид.

Однако мы должны быть осторожными и использовать только те математические операции, которые имитируют представляемую физическую задачу (в данном случае синусоиды). Например, предположим, что мы используем комплексные переменные A и B для представления двух синусоид с одной и той же частотой, но с отличными амплитудами и фазовыми сдвигами. Когда происходит *сложение* двух комплексных чисел, получается третье комплексное число. Аналогично, когда складываются две синусоиды, создается третья синусоида. Как Вы вероятно и надеялись, третье комплексное число представляет третью синусоиду. Комплексное сложение соответствует физической системе.

А сейчас представим умножение комплексных чисел A и B, дающее в результате другое комплексное число. Соответствует ли оно тому, что происходит, когда умножаются две синусоиды? Нет! Умножение двух синусоид *не* производит другой

синусоиды. Комплексное умножение не соответствует физической системе и поэтому не может использоваться.

К счастью правомерные операции четко определяются. Для правомерности должны удовлетворяться два условия. Первое, все синусоиды должны иметь *одну и туже* частоту. Например, если комплексные числа *1+1j* и *2+2j* представляют синусоиды одной и той же частоты, тогда сумма двух синусоид представляется комплексным числом *3+3j*. Однако, если *1+1j* и *2+2j* представляют синусоиды с различными частотами, то не существует ни одной операции, которую можно было бы проделать с комплексным представлением. В таком случае сумма комплексных чисел *3+3j* бессмысленна.

Несмотря на это, при использовании комплексных чисел частота может быть оставлена, как переменная, но это должна быть всюду *та же самая* частота. Например, совершенно справедливо сложение *2+3ωj* и *3ω+1j*, дающее в результате *5ω+(3ω+1)j*. Так представляют синусоиды, у которых амплитуда и фаза меняются с изменением частоты. Хотя мы не знаем, каково конкретное значение частоты, мы знаем, что она везде *одна и та же*, т.е. ω.

Вторым требованием является то, что представляемые операции должны быть *линейными*, как обсуждалось в Главе 5. Например, синусоиды могут объединяться с помощью сложения и вычитания, а не с помощью умножения и деления. Аналогично, системы могут выполнять роль усилителя, аттенюатора, высоко- или низкочастотного фильтра и т.д., но не такие функции, как пороговые, возведение в квадрат и ограничение. Помните, что свертка и Фурье анализ справедливы только для линейных систем.

Комплексное представление систем

Рис.30.4 показывает пример использования комплексных чисел для представления синусоиды, проходящей через линейную систему. Будем использовать для этого примера непрерывный сигнал, хотя дискретный сигнал обрабатывается таким же образом. Поскольку входной сигнал является синусоидой, а система линейна, выходной сигнал также будет синусоидой, имеющей ту же частоту, что и входной сигнал. Как показано на рисунке, входной сигнал в этом примере условно представлен, как $3\cos(\omega t+\pi/4)$ или эквивалентным выражением $2{,}1213\cos(\omega t)-2{,}1213\sin(\omega t)$. При представлении с помощью комплексного числа оно выглядит, как $3e^{-j\pi/4}$ или *2,1213+j2,1213*. Аналогично, выходной сигнал условно представлен, как $1{,}5\cos(\omega t-\pi/8)$ или в альтернативной форме $1{,}3858\cos(\omega t)+0{,}5740\sin(\omega t)$. Это представляется комплексным числом $1{,}5e^{j\pi/8}$ или *1,3858-j0,5740*.

Характеристики системы также могут быть представлены с помощью комплексного числа. Модуль комплексного числа - это отношение между модулями выходного и входного сигналов (т.е. $M_{вых}/M_{вх}$). Аналогично, угол комплексного числа это разница между выходным и входным углами взятая с *обратным знаком* (т. е. $-(\varphi_{вых}-\varphi_{вх})$). В приведенном здесь примере система описывается комплексным числом $0{,}5e^{j3\pi/8}$. Другими словами амплитуда синусоиды уменьшается в 0,5 раза, в то время как фазовый угол изменяется на $-3\pi/8$.

Комплексное число, представляющее систему, может быть преобразовано в прямоугольную форму как *0,1913-j0,4619*, но при интерпретации того, что это означает, мы должны быть весьма осторожными. Это *не означает* ни то, что синусоидальная волна, проходящая через систему, изменяется по амплитуде до *0,1913*, ни то, что косинусоидальная волна изменяется до *–0,4619*. Вообще чистая синусоидальная или косинусоидальная волна, входящая в линейную систему, преобразуется в *смесь* синусоидальной и косинусоидальной волн.

К счастью, комплексная математика автоматически отслеживает эти пересекающиеся члены. Когда синусоида проходит через линейную систему, комплексные числа, представляющие входной сигнал, и система *перемножаются*,

производя комплексное число представляющее выходной сигнал. Если любые два комплексных числа известны, может быть найдено третье комплексное число. Вычисления, как это показано на рис. 30.4, могут быть произведены либо в полярной, либо в прямоугольной форме.

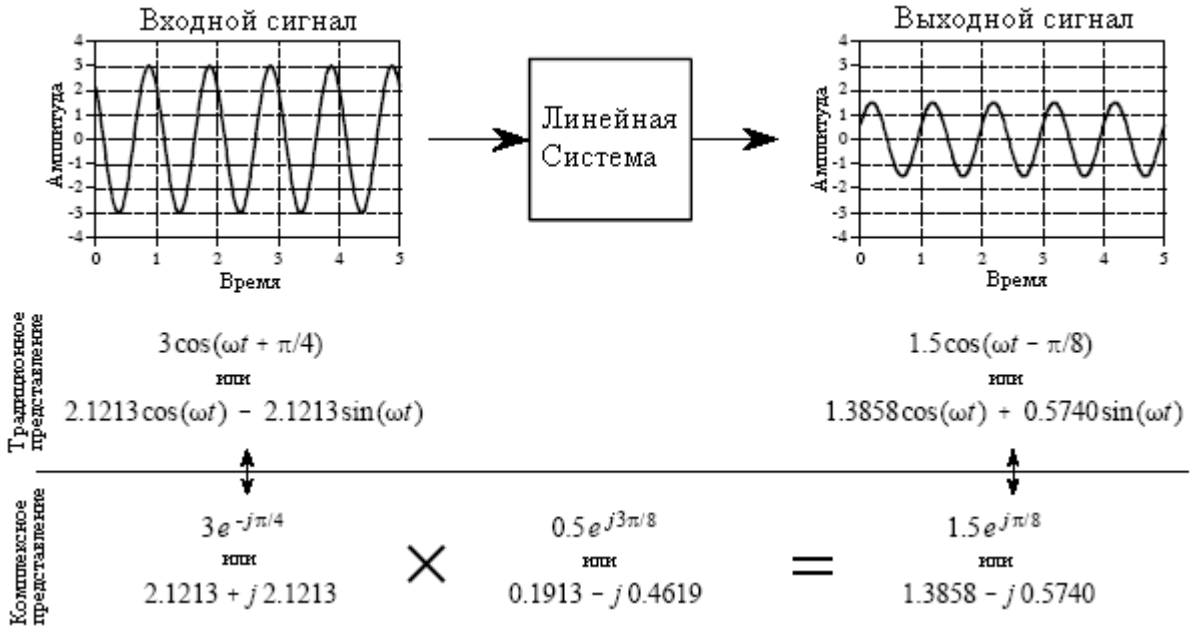


Рис. 30.4 Синусоиды представленные комплексными числами

В предыдущих главах мы описали, как преобразование Фурье разлагает сигнал на косинусные и синусные волны. Амплитуды косинусных волн были названы *действительной частью*, в то время как амплитуды синусных волн назывались *мнимой частью*. Мы подчеркнули, что эти амплитуды - обычные числа, а члены, действительные и мнимые - только названия, используемые для того, чтобы их не путать. Теперь, когда были введены комплексные числа, должно быть вполне очевидно, откуда пришли эти названия. Например, представьте *1024* точки сигнала, разложенного на *513* косинусных волн и *513* синусных волн. Используя метод подстановки, мы можем представить спектр с помощью *513* комплексных чисел. Однако не следует вводить себя в заблуждение, думая, что это и есть *комплексное преобразование Фурье*, тема Главы 31. Это все еще *действительное преобразование Фурье*, разве только что спектр с помощью метода подстановки был размещен в комплексном формате.

Анализ электрических схем

Данный метод подстановки комплексных чисел вместо косинусных и синусных волн называется **фазорным преобразованием**. Это основной инструмент, используемый при анализе электрических цепей состоящих из резисторов, конденсаторов и катушек индуктивностей. (Более формально инженеры-электрики определяют фазорное преобразование, как умножение на комплексную величину $e^{j\omega}$ и нахождение действительной части. Такой подход позволяет записывать процедуру в виде уравнения, делая ее более простой, при математической обработке. Метод “подстановки” приводит к тому же результату, но он менее элегантен.)

Прежде всего, нужно понять взаимосвязь между током и напряжением для каждого из этих элементов. Для резисторов она выражается в законе Ома: $v=iR$, где i – мгновенный ток, протекающий через элемент, v – мгновенное напряжение, приложенное к элементу и R - сопротивление. Конденсатор и катушка индуктивности, напротив, подчиняются

дифференциальным уравнениям: $i=Cdv/dt$ и $v=Ldi/dt$, где C – емкость и L – индуктивность. В большинстве основных методов анализа электрических цепей сначала, в соответствии с конфигурацией схемы, составляются эти противные дифференциальные уравнения, которые решаются затем относительно интересующих параметров. Несмотря на то, что это даст ответы практически на *все*, что касается электрической цепи, математика здесь, может стать настоящим кошмаром.

Все может быть сильно упрощено наложением ограничения, чтобы сигналы были синусоидальными. Представляя эти синусоиды комплексными числами, трудные *дифференциальные* уравнения могут быть непосредственно заменены намного более простыми *алгебраическими* уравнениями. Рис. 30.5 иллюстрирует, как все это работает. Каждую из этих трех компонент (резистор, конденсатор и катушку индуктивности), показанных на рисунке, мы рассматриваем как *систему*. Входным сигналом системы является синусоидальный ток через элемент, в то время как выходной сигнал – синусоидальное напряжение, падающее на двух его выводах. Это означает, что мы можем представить входной и выходной сигналы системы двумя комплексными переменными: I (для тока) и V (для напряжения), соответственно. Отношение выходного сигнала к входному сигналу может быть также выражено с помощью комплексного числа (комплексное число, являющееся результатом деления двух других комплексных чисел, обычно называют фазором – прим. перев.). Это комплексное число называется **импедансом** (полным сопротивлением – прим. перев.) и ему присваивается символ Z. Это означает, что

$$I \times Z = V .$$

Выражаясь словами, комплексное число, представляющее синусоидальное напряжение, равно комплексному числу, представляющему синусоидальный ток, умноженному на импеданс (другое комплексное число). Если любые из этих двух чисел заданы, то может быть найдено третье. В полярной форме модуль импеданса - это отношение амплитуды напряжения V к амплитуде тока I. Аналогично, фаза импеданса - это разница фаз между V и I.

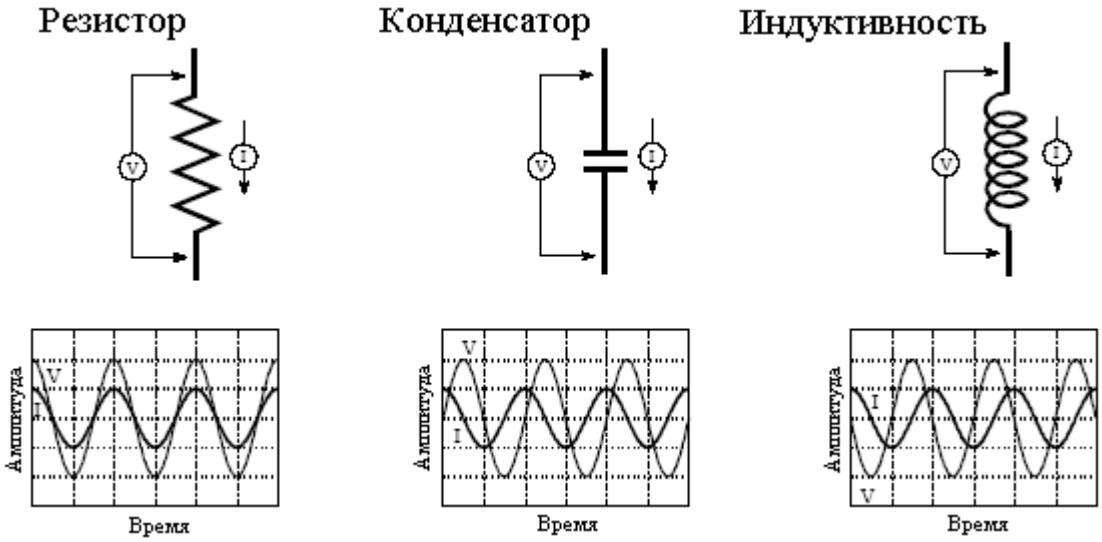


Рис. 30.5 Определение импеданса

Указанные соотношения можно рассматривать, как *закон Ома для синусоид*. Закон Ома ($v=iR$) описывает, каким образом *сопротивление* связывает мгновенные ток и напряжение в резисторе. Когда синусоидальный сигнал представлен комплексным числом, соотношение становится: $V=IZ$. То есть *импеданс* связывает ток и напряжение.

Сопротивление - обычное число, так как оно имеет дело с двумя обычными числами. Импеданс - комплексное число, так как оно связывает два комплексных числа. Импеданс содержит информации больше чем сопротивление, потому что он диктует и амплитуды *и* фазовые углы.

Из дифференциальных уравнений, подчиняющихся своим собственным правилам, можно показать, что импедансом резистора, конденсатора и катушки индуктивности являются R, $-j/\omega C$ и $j\omega L$, соответственно. В качестве примера представьте, что ток в каждой из этих компонент - косинусоидальная волна единичной амплитуды, как показано на рис. 30.5. С помощью метода подстановки, он может быть представлен комплексным числом $1+0j$. Напряжение, падающее на резисторе, будет: $V=IZ=(1+0j)R=R+0j$. Другими словами, косинусоидальная волна с амплитудой R. Напряжение на конденсаторе найдется как: $V=IZ=(1+0j)(-j/\omega C)$. Это упрощается до $0 - j/\omega C$ - синусоидальная волна с амплитудой $1/\omega C$. Аналогично, напряжение на катушке индуктивности может быть вычислено, как: $V=IZ=(1+0j)(j\omega L)$. Это упрощается до $0 + j\omega L$ - отрицательная синусоидальная волна с амплитудой ωL.

Красота этого метода состоит в том, что *R-L-C* цепи могут анализироваться без того чтобы обращаться к дифференциальным уравнениям. С *импедансом* резисторов, конденсаторов и катушек индуктивностей обращаются таким же образом, как с *сопротивлением* в цепях постоянного тока. Это относится ко всем основным схемам соединения, таким как: последовательное соединение резисторов, параллельное соединение резисторов, делители напряжения и т.д.

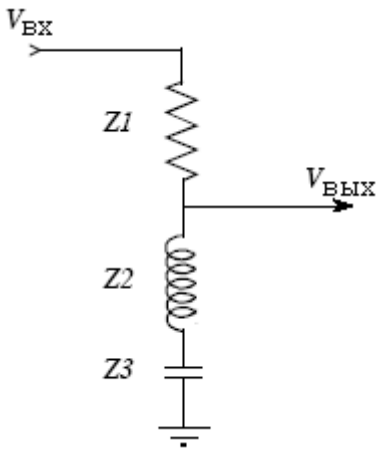


Рис. 30.6 *R-L-C* фильтр-пробка

В качестве примера на рис. 30.6 показана *R-L-C* цепь, называемая **фильтр - “пробка”**, используемая для удаления узкой полосы частот. Например, она может устранять фон переменного тока промышленной частоты в аудио или измерительном сигнале. Если бы эта цепь состояла из трех резисторов (вместо резистора, конденсатора и катушки индуктивности), отношение между выходным и входным сигналами определялось бы с помощью формулы для делителя напряжения: $v_{вых}/v_{вх}=(R_2+R_3)/(R_1+R_2+R_3)$. Поскольку цепь содержит конденсатор и катушку индуктивности уравнение переписывается с использованием импедансов:

$$\frac{v_{вых}}{v_{вх}} = \frac{Z_2 + Z_3}{Z_1 + Z_2 + Z_3},$$

где $v_{вых}$, $v_{вх}$, Z_1, Z_2 и Z_3 - все являются комплексными переменными. Подставляя значения импеданса для каждой переменной, получим:

$$\frac{v_{вых}}{v_{вх}} = \frac{j\omega L - \dfrac{j}{\omega C}}{R + j\omega L - \dfrac{j}{\omega C}}.$$

Затем мы прокручиваем это через алгебру для того, чтобы отделить все содержащее *j* от всего, что *j* не содержит. Другими словами мы разделяем уравнение на его действительную и мнимую части. Эта алгебра может быть утомительна и длинна, но альтернативой является только запись и решение дифференциальных уравнений, еще более противная задача. Когда действительная и мнимая части разделены, появляется комплексное представление фильтра – “пробки”:

$$\frac{v_{вых}}{v_{вх}} = \frac{k^2}{R^2 + k^2} + j\frac{Rk}{R^2 + k^2},$$

где $k=\omega L-1/\omega C$.

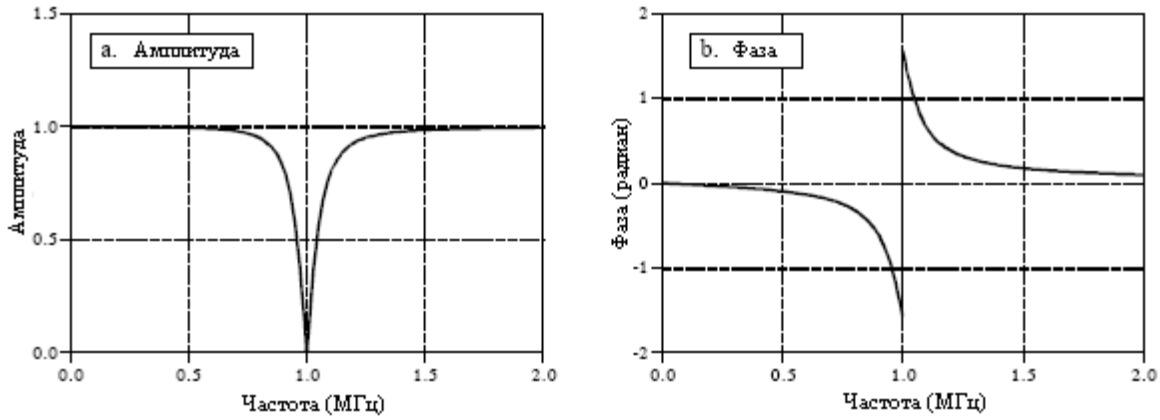


Рис. 30.7 Частотная характеристика фильтра-пробки

Наконец, соотношение преобразуется к полярному обозначению и отображается графически как на рис. 30.7 (кривые на рисунке приведены для следующих значений элементов: *R=50* Ом, *C=470* пФ и *L=54* мкГн):

$$Mag = \frac{\omega L - 1/\omega C}{\sqrt{R^2 + (\omega L - 1/\omega C)^2}} \qquad Phase = \arctan\left(\frac{R}{\omega L - 1/\omega C}\right).$$

Ключевой момент, который следует запомнить из этих примеров - то, как *метод подстановки* позволяет комплексным числам представлять задачи реального мира. В следующей главе мы познакомимся с более превосходным способом использования комплексных чисел в науке и инженерном деле, с *математической эквивалентностью*.

ГЛАВА
31

КОМПЛЕКСНОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

Несмотря на то, что комплексные числа главным образом оторваны от нашей реальности, они могут быть использованы для решения научных и инженерных задач двумя способами. Первый способ заключается в том, что параметры из задачи реального мира могут быть подставлены внутрь комплексной формы так, как это показано в предыдущей главе. Второй способ гораздо более элегантный и мощный, способ создания комплексных чисел математически эквивалентных физической задаче. Этот подход приводит к *комплексному преобразованию Фурье* - более утонченной версии *действительного преобразования Фурье*, обсужденной в Главе 8. Комплексное преобразование Фурье важно не только само по себе, но также и как ступень к более мощным комплексным методам, таким как *преобразование Лапласа* и *z-преобразование*. Эти комплексные преобразования являются базисом теории ЦОС.

Действительное ДПФ

Все четыре члена семейства преобразования Фурье (ДПФ, ДВПФ, ряды Фурье и преобразование Фурье) могут быть выполнены, как с действительными, так и с комплексными числами. Поскольку ЦОС в основном сосредоточивается на ДПФ, мы будем использовать его в качестве примера.

Перед тем как погрузиться в комплексную математику, давайте еще раз просмотрим действительное ДПФ, сделав специальный акцент на тех вещах, которые в математике кажутся неуклюжими. В Главе 8 мы определили версию *действительного* дискретного преобразования Фурье в соответствии с уравнениями:

$$\mathrm{Re}\,X[k] = \frac{2}{N}\sum_{n=0}^{N-1} x[n]\cos(2\pi kn/N)$$

$$\mathrm{Im}\,X[k] = \frac{-2}{N}\sum_{n=0}^{N-1} x[n]\sin(2\pi kn/N). \qquad (31.1)$$

Выражаясь словами, состоящий из N отсчетов сигнал во временной области $x[n]$, раскладывается на набор из $N/2+1$ косинусных волн и $N/2+1$ синусных волн с частотами, определяемыми индексом k. Амплитуды косинусных волн содержатся в *Re* X[*k*], амплитуды синусных волн содержатся в *Im* X[*k*]. Работа этих уравнений заключается в *коррелировании* соответствующих косинусных или синусных волн с сигналом во временной области. Несмотря на использование названий: *действительная часть* и *комплексная часть*, в этих уравнениях комплексных чисел нет. Здесь нигде в поле зрения нет *j*! Мы также включили в эти уравнения нормализующий фактор *2/N*. Помните, он может быть помещен перед любым уравнением синтеза или анализа или обрабатываться

отдельно (как описано в уравнении 8.3). Такие уравнения должны быть Вам хорошо знакомы из предыдущих глав. Если нет, вернитесь назад и перед продолжением освежите эти концепции. Ведь если Вы не понимаете *действительного* ДПФ, Вы никогда не сможете понять *комплексного* ДПФ.

Даже притом, что действительное ДПФ использует только действительные числа, *подстановка* позволяет *представить* частотную область комплексными числами. Как следует из названия множеств, *Re* X*[k]* становится действительной частью спектра комплексной частоты, а *Im* X*[k]* становится мнимой частью. Другими словами мы ставим *j* перед каждым значением в мнимой части и складываем результат с действительной частью. Однако не делайте ошибки, думая, что это и есть "комплексное ДПФ". Здесь нет ничего кроме *действительного ДПФ с комплексной подстановкой*.

В то время как действительное ДПФ адекватно многим приложениям в науке и инженерном деле, математически оно неуклюже в трех аспектах. Первый, оно может использовать преимущества комплексных чисел только через применение **подстановки**. Это создает для математиков дискомфорт; они желают говорить: "это *равно* тому", а не просто: "это *представляет* то". Например, представьте, что мы делаем следующее математическое утверждение: *A* равно *B*. Мы моментально знаем бесчисленные следствия: *5A = 5B*, *1+A = 1+B*, *A/x = B/x* и т.д. Теперь предположим, что мы делаем утверждение: *A* представляет *B*. Без дополнительной информации, мы абсолютно ничего не знаем! Когда вещи равны, мы получаем доступ к четырем тысячелетиям математики. Когда вещи только представляют друг друга, мы должны начать на пустом месте с новых определений. Например, когда синусоиды представляются с помощью комплексных чисел, мы допускаем сложение и вычитание, но запрещаем умножение и деление.

Второй аспект, плохо поддающийся действительному преобразованию Фурье, это часть спектра с **отрицательными частотами**. Как Вы помните из Главы 10, синусные и косинусные волны могут быть описаны, как имеющие *положительную* частоту и *отрицательную* частоту. Поскольку оба взгляда идентичны, действительное преобразование Фурье игнорирует отрицательные частоты. Однако существуют приложения, где отрицательные частоты важны. Это бывает тогда, когда компоненты с отрицательными частотами принудительно перемещаются в часть спектра положительных частот. Как говорят, приведения принимают человеческую форму. Например, это то, что случается при совмещении имен, круговой свертке и амплитудной модуляции. Поскольку действительное преобразование Фурье не использует отрицательные частоты, его возможности, связанные с этими ситуациями, очень ограничены.

Третье чем мы недовольны, это **специальная обработка *Re* X*[0]*** и ***Re* X*[N/2]***, первой и последней точки в частотном спектре. Предположим, что мы начнем с *N* точечного сигнала *x[n]*. Взятие ДПФ дает частотный спектр, содержащийся в *Re* X*[k]* и *Im* X*[k]*, где *k* изменяется от *0* до *N/2*. Однако это не соответствует амплитудам, требуемым для восстановления формы волны во временной области; прежде всего, отсчеты *Re* X*[0]* и *Re* X*[N/2]* должны быть разделены на два. (Посмотрите на уравнение 8.3 чтобы освежить свою память.) Последнее легко выполняется в компьютерных программах, но иметь с этим дело в уравнениях неудобно.

Элегантным решением этих проблем является комплексное преобразование Фурье. Для комплексных чисел и отрицательных частот идти рука об руку совершенно естественно. Давайте посмотрим, как все это работает.

Математическая эквивалентность

Наш первый шаг показать, как синусоидальные и косинусоидальные волны могут быть вписаны в *уравнение* с комплексными числами. Ключом к этому служит Формула Эйлера, представленная в предыдущей главе:

$$e^{jx} = \cos(x) + j\sin(x). \qquad (31.2)$$

На первый взгляд она не оказывает большой помощи; одно комплексное выражение равно другому комплексному выражению. Тем не менее, небольшие алгебраические преобразования могут привести к двум другим формам:

$$\cos(x) = \frac{e^{jx} + e^{-jx}}{2} \qquad \sin(x) = \frac{e^{jx} - e^{-jx}}{2j}. \qquad (31.3)$$

Данный результат чрезвычайно важен, мы выработали способ записи *уравнений*, связывающих комплексные числа и обычные синусоиды. Хотя уравнение 31.3 является стандартной формой тождества, для данного обсуждения будет более полезным, если мы поменяем некоторые члены друг с другом местами:

$$\begin{aligned} \cos(\omega t) &= \frac{1}{2}e^{j(-\omega)t} + \frac{1}{2}e^{j\omega t} \\ \sin(\omega t) &= \frac{1}{2}je^{j(-\omega)t} - j\frac{1}{2}e^{j\omega t}. \end{aligned} \qquad (31.4)$$

Каждое выражение является суммой двух экспонент: одна содержит *положительную* частоту (ω), а другая содержит отрицательную частоту ($-\omega$). Другими словами, когда синусные и косинусные волны записываются комплексными числами, автоматически появляется часть спектра с отрицательными частотами. С положительными и отрицательными частотами обращаются в равной мере одинаково; для формирования законченной формы волны требуется ровно половина каждой из них.

Комплексное ДПФ

Прямое комплексное ДПФ, записанное в полярной форме, определяется как:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}. \qquad (31.5)$$

Для получения альтернативной записи прямого преобразования в прямоугольной форме может использоваться Формула Эйлера:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]\left(\cos(2\pi kn/N) - j\sin(2\pi kn/N)\right). \qquad (31.6)$$

Для начала сравним уравнение *комплексного преобразования Фурье* с уравнением *действительного преобразования Фурье*, уравнение 31.1. На первый взгляд они кажутся идентичными, если не считать небольших алгебраических преобразований необходимых для преобразования уравнения 31.6 в уравнение 31.1. Однако, это очень большое заблуждение; различия между этими двумя уравнениями неуловимы и их очень легко пропустить, а они чрезвычайно важны. Давайте подробно пройдемся по этим различиям.

Во-первых, действительное преобразование Фурье преобразует сигнал *x[n]* во временной области в два сигнала *Re* X*[k]* и *Im* X*[k]* в действительной частотной области. Используя комплексную подстановку, частотная область может быть *представлена* простым комплексным множеством X*[k]*. В комплексном преобразовании Фурье оба и *x[n]* и X*[k]* - это множества комплексных чисел. Практическое замечание: Даже притом, что временная область здесь является комплексной, не существует ничего, что *потребует* от нас использования ее мнимой части. Предположим, что мы хотим обработать действительный сигнал такой, как последовательность измерений напряжения, сделанных в течение некоторого времени. Такая группа данных становится действительной частью сигнала временной области, в то время как мнимая часть состоит из нулей.

Во-вторых, действительное преобразование Фурье связано только с *положительными* частотами. То есть индекс *k* частотной области изменяется только от *0* до *N/2*. Для сравнения, комплексное преобразование Фурье включает как *положительные*, так и *отрицательные* частоты. Это означает, что *k* изменяется от *0* до *N-1*. Частоты лежащие между *0* и *N/2* положительные, в то время как частоты лежащие между *N/2* и *N-1* отрицательные. Вспоминаете, частотный спектр дискретного сигнала является периодическим, создающим отрицательные частоты между *N/2* и *N-1* точно так же, как и между *-N/2* и *0*. Отсчеты *0* и *N/2* являются разделительной линией между положительными и отрицательными частотами. Если Вам нужно освежить это в памяти, вернитесь назад к Главам 10 и 12.

В-третьих, в действительном преобразовании Фурье с подстановкой к синусоидальной составляющей волны была добавлена *j*, чтобы дать возможность представить частотный спектр комплексными числами. Чтобы осуществить обратное преобразование, к обычным синусоидальным и косинусоидальным волнам, мы можем просто исключить *j*. Это - небрежность, она появляется тогда, когда одна вещь только *представляет* другую. Для сравнения, комплексное ДПФ, уравнение 31.5, является формальным математическим уравнением с *j*, являющейся неотъемлемой его частью. При таком представлении мы не можем произвольно добавлять или исключать *j* более, чем мы можем добавлять или удалять любую другую переменную в уравнении.

В-четвертых, перед действительным преобразованием Фурье стоит масштабирующий коэффициент равный *двум*, в то время как перед комплексным преобразованием Фурье его нет. Возьмем, скажем, действительное преобразование Фурье косинусоидальной волны с *единичной* амплитудой. Спектральная величина, соответствующая этой косинусоидальной волне также будет равна *единице*. А теперь давайте повторим данный процесс с использованием комплексного ДПФ. В этом случае косинусоидальная волна соответствует *двум* спектральным величинам с положительной и отрицательной частотой. Обе величины с этими частотами имеют каждая значение равное *1/2*. Другими словами, величина с положительной частотой с амплитудой, равной *1/2* объединяется с величиной с отрицательной частотой, с амплитудой также равной *1/2*, давая в результате косинусоидальную волну с амплитудой, равной *единице*.

В-пятых, действительное преобразование Фурье требует специальной обработки двух выборок из частотной области: *Re* X*[0]* и *Re* X*[N/2]*, а комплексное преобразование Фурье нет. Предположим, что мы осуществили ДПФ сигнала во временной области, с целью получить сигнал в частотной области. Чтобы проделать обратные преобразования, мы осуществляем обратное ДПФ сигнала в частотной области, восстанавливая исходный сигнал во временной области. Однако для того чтобы сделать восстанавливаемый сигнал идентичным исходному сигналу, требуется произвести масштабирование. Для комплексного преобразования Фурье, где-нибудь в процессе работы, должен быть введен масштабирующий коэффициент *1/N*. Он может быть введен во время прямого преобразования, во время обратного преобразования или выполнен в виде отдельного шага между ними. Для действительного преобразования Фурье, как описывалось выше, требуется еще и дополнительный коэффициент равный двум (*2/N*). Однако

действительное преобразование Фурье требует также еще и дополнительного масштабирующего шага: где-нибудь в процессе работы *Re* X*[0]* и *Re* X*[N/2]* должны быть разделены на *два*. Выражаясь другими словами, с двумя этими отсчетами используется масштабирующий коэффициент *1/N*, в то время как с оставшейся частью спектра используется *2/N*. Как утверждалось ранее, этот неуклюжий этап и есть одно из наших сетований на действительное преобразование Фурье.

Почему действительное и комплексное преобразования Фурье отличаются в том, как обрабатываются эти две точки? Чтобы ответить на этот вопрос, вспомните что косинусные (или синусные) волны во временной области в спектре комплексного ДПФ расщепляются между положительными и отрицательными частотами. К этому, однако, здесь имеются два исключения, спектральные величины при *0* и *N/2*. Они соответствуют нулевой частоте (постоянная составляющая) и частоте Найквиста (половина частоты дискретизации). Поскольку эти точки находятся между положительной и отрицательной частями спектра, у них нет парной точки. Из-за того, что они не объединяются с другими величинами, они, по сравнению с другими частотами, соответственно, вносят только половину вклада во временную область.

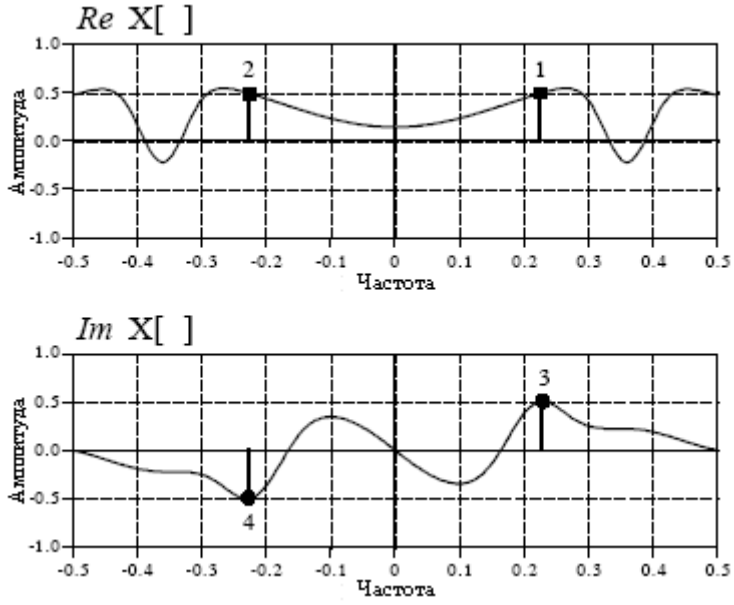


Рис. 31.1 Комплексный частотный спектр

Рис. 31.1 иллюстрирует частотный спектр комплексного ДПФ. Предполагается, что временная область, показанная на этом рисунке полностью *действительная*, т.е. ее мнимая часть равна нулю. Вскоре мы обсудим и идею сигналов мнимой временной области. Существуют два типичных подхода представления спектра комплексной частоты. Как показано здесь, нулевая частота может быть расположена в центре с положительными частотами, размещенными справа, и отрицательными слева. Это наилучший способ *поразмышлять* о полном спектре и *единственный* путь, которым может быть представлен непериодический спектр.

Сложность заключается в том, что спектр дискретного сигнала является периодическим (такой же, как у ДПФ и ДВПФ). Это означает, что все, что находится между *–0,5* и *0,5* повторяет себя бесконечное число раз слева и справа. В этом случае спектр между *0* и *1* содержит такую же информацию, как в промежутке от *–0,5* до *0,5*. Когда создаются графики, такие как на рис. 31.1, обычно используется промежуток от *–0,5* до *0,5*. Однако в большом количестве уравнений и программ используется промежуток от *0* до *1*. Например, в уравнения 31.5 и 31.6 индекс частоты *k* изменяется от *0* до *N-1*

(совпадает с промежутком от *0* до *1*). Тем не менее, если бы мы хотели, мы могли бы записать его изменяющимся от *–N/2* до *N/2* (совпадает с промежутком от *–0,5* до *0,5*).

Используя спектр на рис. 31.1 как путеводитель, мы можем исследовать, как обратное комплексное ДПФ восстанавливает сигнал во временной области. Обратное комплексное преобразование Фурье, записанное в полярной форме, определяется как:

$$x[n] = \sum_{k=0}^{N-1} X[k] e^{j 2\pi kn / N} . \tag{31.7}$$

Используя формулу Эйлера, его можно записать в прямоугольной форме как:

$$\begin{aligned} x[n] = & \sum_{k=0}^{N-1} \mathrm{Re}\, X[k] \left(\cos(2\pi kn / N) + j \sin(2\pi kn / N)\right) - \\ & - \sum_{k=0}^{N-1} \mathrm{Im}\, X[k] \left(\sin(2\pi kn / N) - j \cos(2\pi kn / N)\right). \end{aligned} \tag{31.8}$$

Обратное ДПФ обычно записывается в компактной форме в виде уравнения 31.7, хотя расширенная версия, уравнение 31.8, легче для понимания. Выражаясь словами, каждое значение действительной части частотной области вносит вклад в действительную часть косинусоидальной составляющей *и* в мнимую часть синусоидальной составляющей волны во временной области. Аналогично, каждое значение мнимой части частотной области вносит вклад в действительную часть синусоидальной составляющей *и* в мнимую часть косинусоидальной составляющей волны во временной области. Значение во временной области находится сложением всех этих действительных и мнимых составляющих. Важной концепцией является то, что каждое значение в частотной области производит обе и *действительную* и *мнимую* составляющие во временной области.

Например, представьте, что мы хотим восстановить косинусоидальную волну единичной амплитуды с частотой *2πk/N*. Для этого потребуются составляющие с положительной и с отрицательной частотой, обе из действительной части частотного спектра. Два маркера в виде квадрата на рис. 31.1 показывают пример этого для частоты, установленной на: *k/N = 0,23*. Составляющая с положительной частотой *0,23* (метка 1 на рис. 31.1) дает во временной области действительную косинусоидальную и мнимую синусоидальную волну:

$$\frac{1}{2}\cos(2 \cdot \pi \cdot 0{,}23 \cdot n) + \frac{1}{2} j \sin(2 \cdot \pi \cdot 0{,}23 \cdot n) .$$

Аналогично, составляющая с отрицательной частотой *-0,23* (метка 2 на рис. 31.1) также дает во временной области действительную косинусоидальную и мнимую синусоидальную волну:

$$\frac{1}{2}\cos(2 \cdot \pi \cdot (-0{,}23) \cdot n) + \frac{1}{2} j \sin(2 \cdot \pi \cdot (-0{,}23) \cdot n) .$$

Знак минус может быть вынесен за пределы синуса и косинуса в соответствии со следующими выражениями: *cos(-x)=cos(x)* и *sin(-x)=-sin(x)*. Это позволяет записать вклад, вносимый составляющей с отрицательной частотой, по-другому:

$$\frac{1}{2}\cos(2\cdot\pi\cdot 0{,}23\cdot n)-\frac{1}{2}j\sin(2\cdot\pi\cdot 0{,}23\cdot n).$$

Сложение вкладов от составляющих с положительной и отрицательной частотами приводит к восстановлению сигнала во временной области:

вклад от составляющей с положительной частотой $\frac{1}{2}\cos(2\cdot\pi\cdot 0{,}23\cdot n)+\frac{1}{2}j\sin(2\cdot\pi\cdot 0{,}23\cdot n)$

вклад от составляющей с отрицательной частотой $\frac{1}{2}\cos(2\cdot\pi\cdot 0{,}23\cdot n)-\frac{1}{2}j\sin(2\cdot\pi\cdot 0{,}23\cdot n)$

результирующий сигнал во временной области $\cos(2\cdot\pi\cdot 0{,}23\cdot n).$

Таким же образом мы можем синтезировать синусоидальную волну во временной области. В этом случае нам понадобятся составляющие с положительной и отрицательной частотами из мнимой части частотного спектра. На рис. 31.1 этот случай показан круглыми маркерами. В соответствии с уравнением 31.8 такие спектральные составляющие дают во временной области действительные синусоидальные и мнимые косинусоидальные волны. В то время как действительные синусоидальные волны складываются, косинусоидальные волны взаимно сокращаются:

вклад от составляющей с положительной частотой $-\frac{1}{2}\sin(2\cdot\pi\cdot 0{,}23\cdot n)-\frac{1}{2}j\cos(2\cdot\pi\cdot 0{,}23\cdot n)$

вклад от составляющей с отрицательной частотой $-\frac{1}{2}\sin(2\cdot\pi\cdot 0{,}23\cdot n)+\frac{1}{2}j\cos(2\cdot\pi\cdot 0{,}23\cdot n)$

результирующий сигнал во временной области $-\sin(2\cdot\pi\cdot 0{,}23\cdot n).$

Обратите внимание, что даже притом, что составляющая с положительной частотой имела *положительное* значение, генерируется *отрицательная* синусоидальная волна. Такая инверсия знака - присущая часть математики комплексного ДПФ. Как Вы помните, в действительном ДПФ обычно применялась такая же инверсия знака. То есть *положительное* значение в мнимой части частотного спектра соответствовало *отрицательной* синусоидальной волне. Большинство авторов включает такую инверсию знака в определение действительного преобразования Фурье, чтобы сделать его совместимым с его комплексным двойником. Дело в том, что эта инверсия *должна* использоваться в комплексном преобразовании Фурье, а в действительном преобразовании Фурье просто не обязательна.

Очень важна *симметрия* комплексного Преобразования Фурье. Как показано на рис. 31.1, сигнал в действительной временной области соответствует частотному спектру с *четной* действительной частью и *нечетной* мнимой частью. Другими словами, составляющие с отрицательными и положительными частотами имеют такой же знак в действительной части (так же, как точки 1 и 2 на рис. 31.1), но противоположный знак в мнимой части (точки 3 и 4).

Это поднимает другую тему: о мнимой части временной области. До сих пор мы предполагали, что временная область полностью действительна, то есть мнимая часть равна нулю. Однако комплексное преобразование Фурье не требует этого.

Какой физический смысл несет сигнал в мнимой временной области? Обычно не несет никакого. Это просто что-то, что допускается комплексной математикой, но не соответствующее тому миру, в котором мы живем. Однако существуют приложения, где оно может использоваться или им можно манипулировать, преследуя определенные математические цели.

Пример этого представлен в Главе 12. Мнимая часть временной области производит частотный спектр с четной действительной частью и нечетной мнимой частью. Это прямая противоположность спектру, созданному действительной частью временной области (рис. 31.1). Когда временная область содержит обе и действительную и мнимую части, частотный спектр представляет собой сумму двух спектров, как если бы они были рассчитаны индивидуально. В Главе 12 описывается, как это может быть использовано для создания алгоритма ДПФ, вычисляющего частотный спектр *двух* действительных сигналов одновременно. Один сигнал помещается в действительную часть временной области, в то время как другой в мнимую часть. После выполнения вычислений ДПФ, спектры двух сигналов разделяются при помощи четной/нечетной декомпозиции.

Семейство преобразований Фурье

Как у ДПФ, так и у других членов семейства преобразований Фурье есть действительная и комплексная версии. Они дают целый зверинец уравнений, показанных в таблице 31.1. Прежде чем изучить эти уравнения индивидуально, попробуйте понять их как хорошо организованную и симметричную *группу*. Организацию семейства преобразований Фурье описывают следующие комментарии. Они очень подробны, все время повторяются и постоянно действуют на нервы. Тем не менее, это основа основ необходимая для понимания теоретического ДПФ. Учите их хорошо.

1. Четыре преобразования Фурье

Сигнал во временной области может быть как *непрерывным*, так и *дискретным* и он может быть как *периодическим*, так и *непериодическим*. Это определяет четыре типа Преобразований Фурье: **Дискретное Преобразование Фурье** (дискретное, периодическое), **Дискретно-Временное Преобразование Фурье** (дискретное, непериодическое), **Ряды Фурье** (непрерывные, периодические) и **Преобразование Фурье** (интеграл Фурье – прим. перев.) (непрерывное, непериодическое). Не пытайтесь понять, в чем причина таких названий, ее просто нет.

Если сигнал в одной области дискретный, то в другой области он будет периодическим. Аналогично, если сигнал в одной области непрерывный, то в другой области он будет непериодическим. Непрерывные сигналы представляются с помощью круглых скобок ( ), в то время как дискретные сигналы представляются с помощью квадратных скобок [ ]. Обозначений того, является ли сигнал периодическим или непериодическим, не существует.

2. Действительное против Комплексного

Каждое из этих четырех преобразований имеет комплексную и действительную версию. У комплексной версии имеется сигнал в комплексной временной области и в комплексной частотной области. У действительной версии имеется сигнал в действительной временной области и два сигнала в действительной частотной области. В комплексных случаях используются и положительные и отрицательные частоты, в то время как в действительных преобразованиях используются только положительные

частоты. Комплексные преобразования обычно записываются в экспоненциальной форме; однако если это необходимо, с помощью формулы Эйлера можно преобразовать их к косинусоидальной и синусоидальной форме.

3. Анализ и синтез

У каждого преобразования есть уравнение анализа (называемое также прямым преобразованием) и уравнение синтеза (называемое также обратным преобразованием). Уравнение анализа описывает то, каким образом вычислить каждое значение в частотной области, опираясь на все значения во временной области. Уравнение синтеза описывает то, как вычислить каждое значение во временной области, опираясь на все значения в частотной области.

4. Обозначения временной области

Непрерывные сигналы временной области обозначаются $x(t)$, в то время как дискретные сигналы временной области обозначаются $x[n]$. Для комплексных преобразований эти сигналы комплексные. Для действительных преобразований эти сигналы действительные. Все сигналы временной области простираются от минус бесконечности до плюс бесконечности. Однако, если сигналы периодические, мы интересуемся только одним периодом, поскольку все остальное является избыточным. Переменные T и N обозначают период соответственно непрерывного и дискретного сигналов во временной области.

5. Обозначения частотной области

Непрерывные сигналы частотной области, если они комплексные, обозначаются как $X(\omega)$, а если они действительные, как $Re\ X(\omega)$ и $Im\ X(\omega)$. Дискретные сигналы частотной области, если они комплексные, обозначаются как $X[k]$, а если они действительные, как $Re\ X[k]$ и $Im\ X[k]$. У комплексных преобразований есть область отрицательных частот, простирающаяся от минус бесконечности до нуля, и область положительных частот, простирающаяся от нуля до плюс бесконечности. Действительные преобразования используют только область положительных частот. Если частотная область периодическая, мы интересуемся только одним периодом, поскольку все остальное является избыточным. Для непрерывных частотных областей, независимая переменная ω совершает один полный период, изменяясь от $-\pi$ до π. В дискретном случае мы пользуемся периодом, на котором k изменяется от 0до $N-1$.

6. Уравнения анализа

Уравнения анализа работают методом *корреляции*, т.е. умножая сигнал временной области на синусоиду и интегрируя (для непрерывной временной области) или суммируя (для дискретной временной области) результат по соответствующему участку временной области. Если сигнал временной области непериодический, соответствующий участок простирается от минус бесконечности до плюс бесконечности. Если сигнал временной области периодический, соответствующий участок ограничивается одним любым полным периодом. Показанные здесь уравнения записаны с интегрированием (или суммированием) на периоде: от 0 до T (или от 0 до $N-1$). Однако любой другой полный период даст точно такой же результат, т.е. от $-T$ до 0, от $-T/2$ до $T/2$ и т.д.

7. Уравнения синтеза

Уравнения синтеза описывают, как отдельные значения во временной области вычисляются из всех значений частотной области. Это осуществляется умножением частотной области на синусоиду и интегрированием (непрерывная частотная область) или суммированием (дискретная частотная область) результата по соответствующему участку частотной области. Если частотная область комплексная и непериодическая,

соответствующий участок простирается от минус бесконечности до плюс бесконечности. Если частотная область комплексная и периодическая, соответствующий участок ограничивается одним полным периодом, т.е. от $-\pi$ до π (непрерывная частотная область), или от *0* до *N-1* (дискретная частотная область). Если частотная область действительная и непериодическая, соответствующий участок простирается от нуля до плюс бесконечности, то есть только на области положительных частот. Наконец, если частотная область действительная и периодическая, соответствующий участок ограничивается половиной периода, содержащей область положительных частот или от *0* до π (непрерывная частотная область) или от *0* до *N/2* (дискретная частотная область).

8. Масштабирование

Для того чтобы заставить уравнения анализа и синтеза взаимно сокращаться, в одно или другое уравнение должен быть помещен масштабирующий коэффициент. В таблице 31.1 мы поместили масштабирующий коэффициент в уравнение анализа. Для комплексного случая такими масштабирующими коэффициентами являются: *1/N*, *1/T* или $1/2\pi$. Поскольку в действительных преобразованиях отрицательные частоты не используются, масштабирующий коэффициент в два раза больше: *2/N*, *2/T* или $1/\pi$. В действительные преобразования при вычислении мнимой части частотного спектра включается также знак минус (действие, используемое для того, чтобы сделать действительные преобразования более согласованным с комплексными преобразованиями). Наконец, уравнения синтеза для действительного ДПФ и действительных рядов Фурье имеют собственные инструкции по масштабированию, включая обработку *Re X[0]* и *Re X[N/2]*.

9. Вариации

В разных публикациях эти уравнения могут выглядеть по-разному. Вот несколько вариаций, которые могут встретиться:

- Использование *f* вместо ω, в соответствии с соотношением: $\omega=2\pi f$;
- Интегрирование выполняется на другом периоде, таком как: от *–T* до *0*, от *–T/2* до *T/2* или от *0* до *T*;
- Перенос всех или части масштабирующих коэффициентов в уравнение синтеза;
- Период заменяется основной частотой в соответствии с: $f_0=1/T$;
- Использование других имен переменных, например в ДВПФ ω может стать Ω, а в рядах Фурье *Re X[k]* и *Im X[k]* могут стать a_k и b_k.

Почему используется комплексное преобразование Фурье

Из этой главы явно очевидно, что комплексное ДПФ намного более сложное, чем действительное ДПФ. Являются ли выгоды от комплексного ДПФ действительно стоящими усилий по изучению запутанной математики? Ответ на этот вопрос зависит от того, кто Вы и для чего Вы собираетесь использовать ДПФ. Основная предпосылка этой книги состоит в том, чтобы большинство практических методов ЦОС могли быть поняты и использованы без обращения к комплексным преобразованиям. Если Вы изучаете ЦОС, чтобы использовать ее в своих, не связанных с ЦОС, научных исследованиях или инженерном деле, комплексное ДПФ, вероятно, является сверх убийственным.

Тем не менее, комплексная математика - первичный язык тех, кто специализируется в ЦОС. Если Вы не понимаете этого языка, Вы не сможете общаться с профессионалами в этой области. Сюда входит и способность понимать литературу по ЦОС: книги, научные труды, технические статьи и т.д. Почему комплексные методы так популярны в компании профессионалов ЦОС?

Таблица 31.1

Дискретное преобразование Фурье (ДПФ)

комплексное преобразование	действительное преобразование
синтез $x[n] = \sum_{k=0}^{N-1} X[k] e^{j 2\pi kn / N}$ *анализ* $X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j 2\pi kn / N}$ Временная область: x[n] - комплексная, дискретная и периодическая; n - изменяется на периоде от 0 до N-1. Частотная область: X[k] - комплексная, дискретная и периодическая; k – изменяется на периоде от 0 до N-1; k=0 до N/2 – положительные частоты; k=N/2 до N-1 – отрицательные частоты.	*синтез* $x[n] = \sum_{k=0}^{N/2} \mathrm{Re}\, X[k] \cos(2\pi kn / N) - \sum_{k=0}^{N/2} \mathrm{Im}\, X[k] \sin(2\pi kn / N)$ *анализ* $\mathrm{Re}\, X[k] = \frac{2}{N} \sum_{n=0}^{N-1} x[n] \cos(2\pi kn / N)$ $\mathrm{Im}\, X[k] = \frac{-2}{N} \sum_{n=0}^{N-1} x[n] \sin(2\pi kn / N)$ Временная область: x[n] - действительная, дискретная и периодическая; n - изменяется на периоде от 0 до N-1. Частотная область: ReX[k] - действительная, дискретн. и периодическая; ImX[k] - действительная, дискретн. и периодическая; k – изменяется на половине периода от 0 до N/2. Примечание: Перед использованием уравнения синтеза значения ReX[0] и ReX[N/2] должны быть разделены на два.

Дискретно-временное преобразование Фурье (ДВПФ)

комплексное преобразование	действительное преобразование
синтез $x[n] = \int_0^{2\pi} X(\omega) e^{j\omega n} d\omega$ *анализ* $X(\omega) = \frac{1}{2\pi} \sum_{n=-\infty}^{+\infty} x[n] e^{-j\omega n}$ Временная область: x[n] - комплексная, дискретная и непериодическая; n - изменяется от -∞ до +∞. Частотная область: X(ω) - комплексная, непрерывная и периодическая; ω – изменяется на одном периоде от 0 до 2π; ω=0 до π - положительные частоты; ω= π до 2π - отрицательные частоты;	*синтез* $x[n] = \int_0^{\pi} \mathrm{Re}\, X(\omega) \cos(\omega n) - \mathrm{Im}\, X(\omega) \sin(\omega n) d\omega$ *анализ* $\mathrm{Re}\, X(\omega) = \frac{1}{\pi} \sum_{n=-\infty}^{+\infty} x[n] \cos(\omega n)$ $\mathrm{Im}\, X(\omega) = \frac{-1}{\pi} \sum_{n=-\infty}^{+\infty} x[n] \sin(\omega n)$ Временная область: x[n] - действительная, дискретн. и непериодическая; n - изменяется от -∞ до +∞. Частотная область: ReX(ω) – действит., непрерывная и периодическая; ImX(ω) – действит., непрерывная и периодическая; ω – изменяется на половине периода от 0 до π.

Таблица 31.1 (продолжение)

Ряды Фурье

комплексное преобразование	действительное преобразование
синтез $x(t) = \sum_{k=-\infty}^{+\infty} X[k] e^{j2\pi kt/T}$ *анализ* $X[k] = \frac{1}{T}\int_0^T x(t) e^{-j2\pi kt/T} dt$ Временная область: x(t) - комплексная, непрерывная и периодическая; t – изменяется на одном периоде от 0 до T. Частотная область: X[k] - комплексная, дискретная и непериодическая; k – изменяется от -∞ до +∞; k>0 - положительные частоты; k<0 - отрицательные частоты.	*синтез* $x(t) = \sum_{k=0}^{+\infty} \mathrm{Re}\, X[k] \cos(2\pi kt/T) - \sum_{k=0}^{+\infty} \mathrm{Im}\, X[k] \sin(2\pi kt/T)$ *анализ* $\mathrm{Re}\, X[k] = \frac{2}{T}\int_0^T x(t) \cos(2\pi kt/T) dt$ $\mathrm{Im}\, X[k] = \frac{-2}{T}\int_0^T x(t) \sin(2\pi kt/T) dt$ Временная область: x(t) - действительная, непрерывная и периодическая; t – изменяется на одном периоде от 0 до T. Частотная область: ReX[k] – действит., дискретная и непериодическая; ImX[k] – действит., дискретная и непериодическая; k – изменяется от 0 до +∞. Примечание: Перед использованием уравнения синтеза значения ReX[0] должно быть разделено на два.

Преобразование Фурье (интеграл Фурье – прим. перев.)

комплексное преобразование	действительное преобразование
синтез $x(t) = \int_{-\infty}^{+\infty} X(\omega) e^{j\omega t} d\omega$ *анализ* $X(\omega) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt$ Временная область: x(t) - комплексная, непрерывная и непериодическая; t – изменяется от -∞ до +∞. Частотная область: X(ω) - комплексная, непрерывн. и непериодическая; ω – изменяется от -∞ до +∞; ω>0 - положительные частоты; ω<0 - отрицательные частоты.	*синтез* $x(t) = \int_0^{+\infty} \mathrm{Re}\, X(\omega) \cos(\omega t) - \mathrm{Im}\, X(\omega) \sin(\omega t) d\omega$ *анализ* $\mathrm{Re}\, X(\omega) = \frac{1}{\pi}\int_{-\infty}^{+\infty} x(t) \cos(\omega t) dt$ $\mathrm{Im}\, X(\omega) = \frac{1}{\pi}\int_{-\infty}^{+\infty} x(t) \sin(\omega t) dt$ Временная область: x(t) – действит., непрерывная и непериодическая; t – изменяется от -∞ до +∞. Частотная область: ReX(ω) – действит., непрерывн. и непериодическая; ImX(ω) – действит., непрерывн. и непериодическая; ω – изменяется от 0 до +∞.

Существует несколько причин, о которых мы уже говорили: компактные уравнения, симметрия между уравнениями анализа и синтеза, симметрия между временной и частотной областями, введение отрицательных частот, шаг к преобразованию Лапласа и *z*-преобразованию и т.д.

Существуют также и более философские причины, которые мы не обсуждали, что-то, что мы называем *истиной*. Мы начали эту главу, перечислив несколько аспектов, по которым действительное преобразование Фурье является неуклюжим. Проблемы исчезли, когда было введено комплексное преобразование Фурье. Прекрасно, сказали мы, комплексное преобразование Фурье разрешило все трудности.

Хотя это справедливо, это не дает надлежащих дивидендов комплексному преобразованию Фурье. Посмотрите на эту ситуацию с такой стороны. Комплексное преобразование Фурье *надлежащим* образом описывает, как ведут себя физические системы, несмотря на свою абстрактную природу. Когда мы ограничиваем математику действительными числами, возникают проблемы. Другими словами, проблемы не *разрешаются* комплексным преобразованием Фурье, они *вносятся* действительным преобразованием Фурье. В мире математики комплексное преобразование Фурье более истинно, чем действительное преобразование Фурье. В этом и заключается его огромная привлекательность для математиков и академических работников - группы людей, стремящейся не просто решить конкретную подручную задачу, а расширить человеческое знание.

ГЛАВА
32

ПРЕОБРАЗОВАНИЕ ЛАПЛАСА

Два главных метода обработки сигнала, свертка и анализ Фурье учат, что линейная система может быть полностью понята по ее импульсной или частотной характеристике. Это очень обобщенный подход, так как импульсные и частотные отклики могут иметь почти любую конфигурацию или форму. В действительности для большого числа приложений в науке и инженерной практике это *очень* типично. Многие параметры нашей вселенной взаимодействуют посредством *дифференциальных уравнений*. Например, напряжение на катушке индуктивности пропорционально производной протекающего через нее тока. Аналогично, сила, приложенная к массе, пропорциональна производной скорости массы. Физика полна отношениями такого вида. Частотные и импульсные отклики таких систем не могут быть произвольными, а должны соответствовать решению этих дифференциальных уравнений. Это означает, что их импульсные отклики могут состоять только из *экспонент* и *синусоид*. Преобразование Лапласа является техникой для анализа таких специальных систем при *непрерывных* сигналах. Подобной техникой, используемой в случае *дискретных* сигналов, является z-преобразование.

Природа s-области

Преобразование Лапласа - это хорошо обоснованная математическая техника для решения дифференциальных уравнений. Оно названо в честь великого французского математика Пьера Симона де Лапласа (1749-1827). Как и все преобразования, преобразование Лапласа преобразует один сигнал в другой в соответствии с некоторым фиксированным набором правил или уравнений. Как иллюстрируется на рис. 32.1, преобразование Лапласа преобразует сигнал во временной области в сигнал в **s-области**, называемой также **s-плоскостью**. Сигнал во временной области является непрерывным, простирается от минус бесконечности до плюс бесконечности, и может быть, как периодическим, так и непериодическим. Преобразование Лапласа допускает, чтобы временная область была *комплексной*, однако в обработке сигналов это требуется редко. В данном обсуждении и почти во всех практических приложениях сигнал во временной области полностью действительный.

Как показано на рис. 32.1, s-область является комплексной плоскостью, т.е. по горизонтальной оси расположены действительные числа, а вдоль вертикальной оси расположены мнимые числа. Расстояние вдоль действительной оси выражается переменной σ - прописная греческая буква сигма. Аналогично, вдоль мнимой оси используется переменная ω – естественная (циклическая – прим. перев.) частота (σ является скоростью изменения амплитуды сигнала или действительной частотой, ω является скоростью изменения фазы сигнала или мнимой частотой - прим. перев.). Эта система координат позволяет с помощью задания значений для σ и ω определить местоположение любой точки. Используя комплексные обозначения, каждое местоположение представляется посредством комплексной переменной s, где $s = \sigma + j\omega$. Так же, как и в случае преобразования Фурье, сигналы в s-области представлены заглав-

ными буквами. Например, сигнал во временной области $x(t)$ преобразуется в сигнал в s-области $X(s)$ или, что то же самое $X(\sigma, \omega)$. S-плоскость является непрерывной и простирается до бесконечности во всех четырех направлениях.

В дополнение к тому, что каждая точка в s-области имеет *местоположение*, определяемое комплексным числом, она имеет еще и *значение*, которое также является комплексным числом. Другими словами, каждое местоположение в s-плоскости имеет действительную часть и мнимую часть. Как и для любых комплексных чисел, действительная и мнимая части могут быть альтернативно выражены как модуль и фаза.

Так же, как преобразование Фурье анализирует сигнал в терминах синусоид, преобразование Лапласа анализирует сигнал в терминах синусоид *и* экспонент. С математической точки зрения это делает преобразование Фурье *подмножеством* более сложного преобразования Лапласа. На рис. 32.1 показано графическое описание того, как s-область соотносится с временной областью. Для того чтобы найти значения на s-плоскости вдоль вертикальной линии (значения для конкретного σ), сигнал временной области умножается сначала на экспоненциальную кривую $e^{-\sigma t}$. Левая половина s-плоскости перемножает временную область с экспонентами, *увеличивающимися* во времени $(\sigma < 0)$, в то время как правая половина с экспонентами, *уменьшающимися* во времени $(\sigma > 0)$. Затем берется комплексное преобразование Фурье экспоненциально взвешенного сигнала. Результирующий спектр размещается на s-плоскости вдоль вертикальной линии, причем верхняя половина s-плоскости содержит положительные частоты, а нижняя половина содержит отрицательные частоты. Обратите особое внимание на то, что значения на оси y $(\sigma = 0)$ s-плоскости точно соответствуют преобразованию Фурье сигнала временной области.

Как обсуждалось в предыдущей главе, комплексное преобразование Фурье дается как:

$$X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt .$$

Это выражение может быть расширено в преобразование Лапласа, во-первых, умножением сигнала временной области на экспоненциальный член:

$$X(\sigma, \omega) = \int_{-\infty}^{+\infty} \left[x(t) e^{-\sigma t} \right] e^{-j\omega t} dt .$$

Хотя это и не самая простая форма преобразования Лапласа, это вероятно наилучшее описание стратегии и функционирования метода. Чтобы представить уравнение в более короткой форме, два экспоненциальных члена могут быть объединены:

$$X(\sigma, \omega) = \int_{-\infty}^{+\infty} x(t) e^{-(\sigma + j\omega)t} dt .$$

Наконец *местоположение* на комплексной плоскости может быть представлено комплексной переменной s, где $s = \sigma + j\omega$. Это позволяет упростить уравнение до еще более компактного выражения:

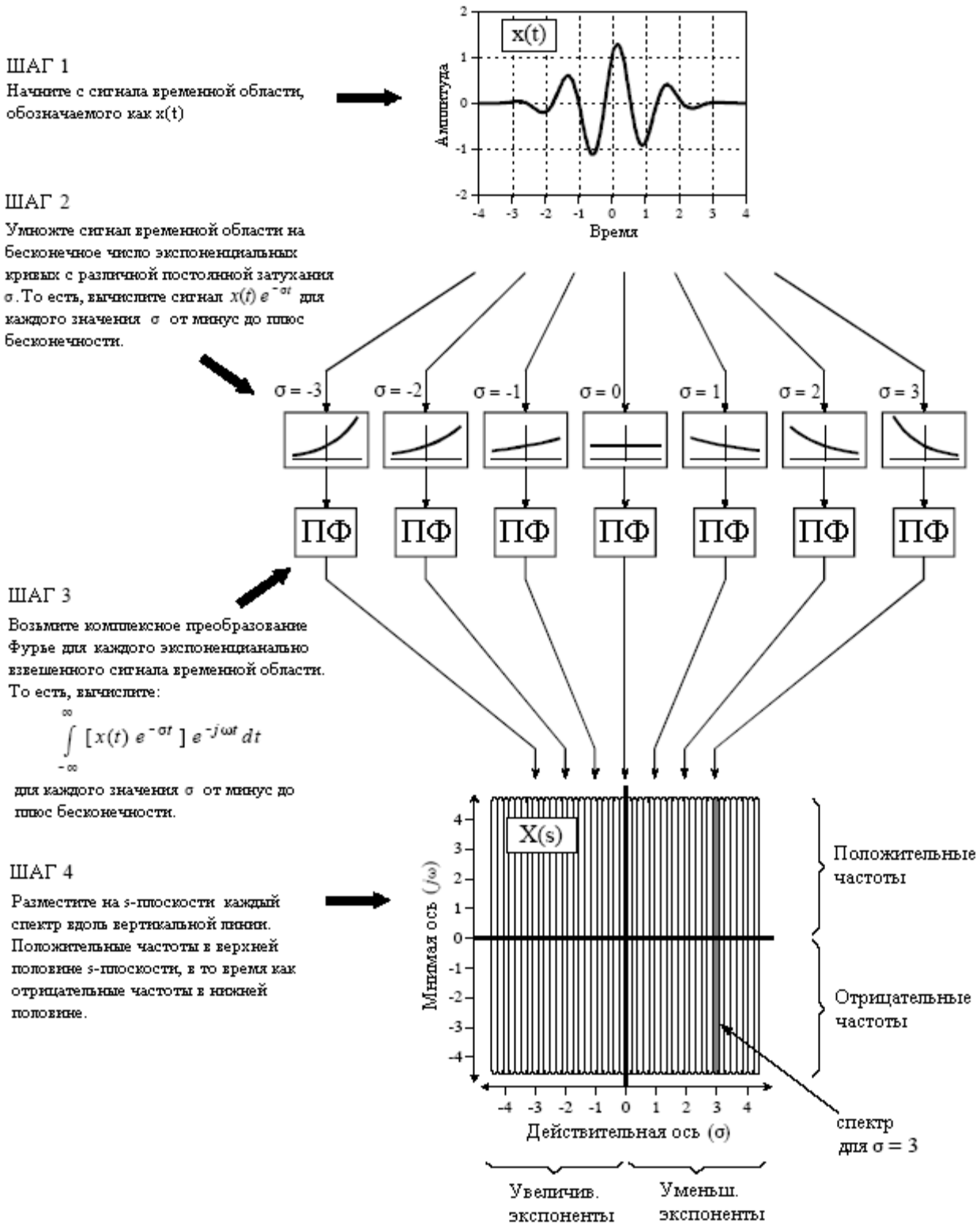


Рис. 32.1 Преобразование Лапласа

$$X(s) = \int_{-\infty}^{+\infty} x(t)e^{-st}dt. \qquad (32.1)$$

Это окончательная форма преобразования Лапласа, одного из наиболее важных уравнений в обработке сигналов и в электронике. Обратите особое внимание на член e^{-st}, называемый *комплексной экспонентой*. Как показано в приведенном выше выводе, комплексные экспоненты являются компактным способом представления, как синусоид, так и экспонент с помощью одного выражения.

Хотя мы объяснили преобразование Лапласа как двухэтапный процесс (умножение на экспоненциальную кривую, за которым следует преобразование Фурье), помните, что этот путь разбиения уравнения (32.1) на более простые компоненты проделан только с целью обучения. Преобразование Лапласа является единым, относящимся к $x(t)$ и $X(s)$, уравнением, а не пошаговой процедурой. Уравнение (32.1) описывает, как вычислить каждую *точку* на s-плоскости (идентифицируемую своим значением для σ и ω) на основе значений σ, ω и сигнала временной области $x(t)$. Использование преобразования Фурье для *одновременного* вычисления всех точек вдоль вертикальной линии просто удобно, а не необходимо. Однако очень важно запомнить то, что значения на s-плоскости вдоль оси y $(\sigma = 0)$ *точно* равны преобразованию Фурье. Как будет объяснено позже в этой главе, это ключевой момент того, почему преобразование Лапласа является полезным.

Для того чтобы далее исследовать характер уравнения (32.1), давайте посмотрим на несколько отдельных точек в s-области и изучим, как значения в этих местах связаны с сигналом временной области. Для начала вспомним, как отдельные точки в *частотной области* соотносятся с сигналом временной области. Каждая точка в частотной области идентифицируется специфическим значением ω, соответствующим двум синусоидам $cos(\omega t)$ и $sin(\omega t)$. Действительная часть находится умножением сигнала временной области на косинусоидальную волну, с последующим интегрированием от $-\infty$ до $+\infty$. Мнимая часть находится таким же образом, только здесь используется синусоидальная волна. Если мы имеем дело с *комплексным* преобразованием Фурье, значение, соответствующее отрицательной частоте $-\omega$, будет комплексно сопряженным (действительная часть такая же, а мнимая часть отрицательная) значению при ω. Преобразование Лапласа является просто расширением тех же самых концепций.

На рис. 32.2 показаны три *пары* точек на s-плоскости: A и A', B и B', C и C'. Так же, как и в комплексном частотном спектре, точки A, B и C (положительные частоты) являются комплексно сопряженными точкам A', B' и C' (отрицательные частоты). Верхняя половина s-плоскости является зеркальным отражением нижней половины, и для того чтобы выполнялось соответствие с сигналом действительной временной области, нужны обе эти половины. Другими словами, обращение с парами этих точек позволяет нам оперировать во временной области только с действительными числами в обход комплексной математики.

Поскольку каждая из этих пар имеет специфические значения для σ и $\pm\omega$, каждой паре соответствуют две формы волны: $cos(\omega t)e^{-\sigma t}$ и $sin(\omega t)e^{-\sigma t}$. Например, местоположение точек A и A' находится при $\sigma = 1{,}5$ и $\omega = \pm 40$, а, следовательно, соответствует формам волны: $cos(40t)e^{-1{,}5t}$ и $sin(40t)e^{-1{,}5t}$. Как показано на рис. 32.2, это синусоиды экспоненциально *затухающие* по амплитуде с течением времени. Точно таким же образом синусоидальная и косинусоидальная волны, соответствующие точкам B и B', имеют *постоянную* амплитуду, как результат того, что значение σ равно нулю. Аналогично, синусоидальная и косинусоидальная волны, соответствующие местоположениям C и C', экспоненциально *нарастают* по амплитуде, поскольку σ является отрицательной.

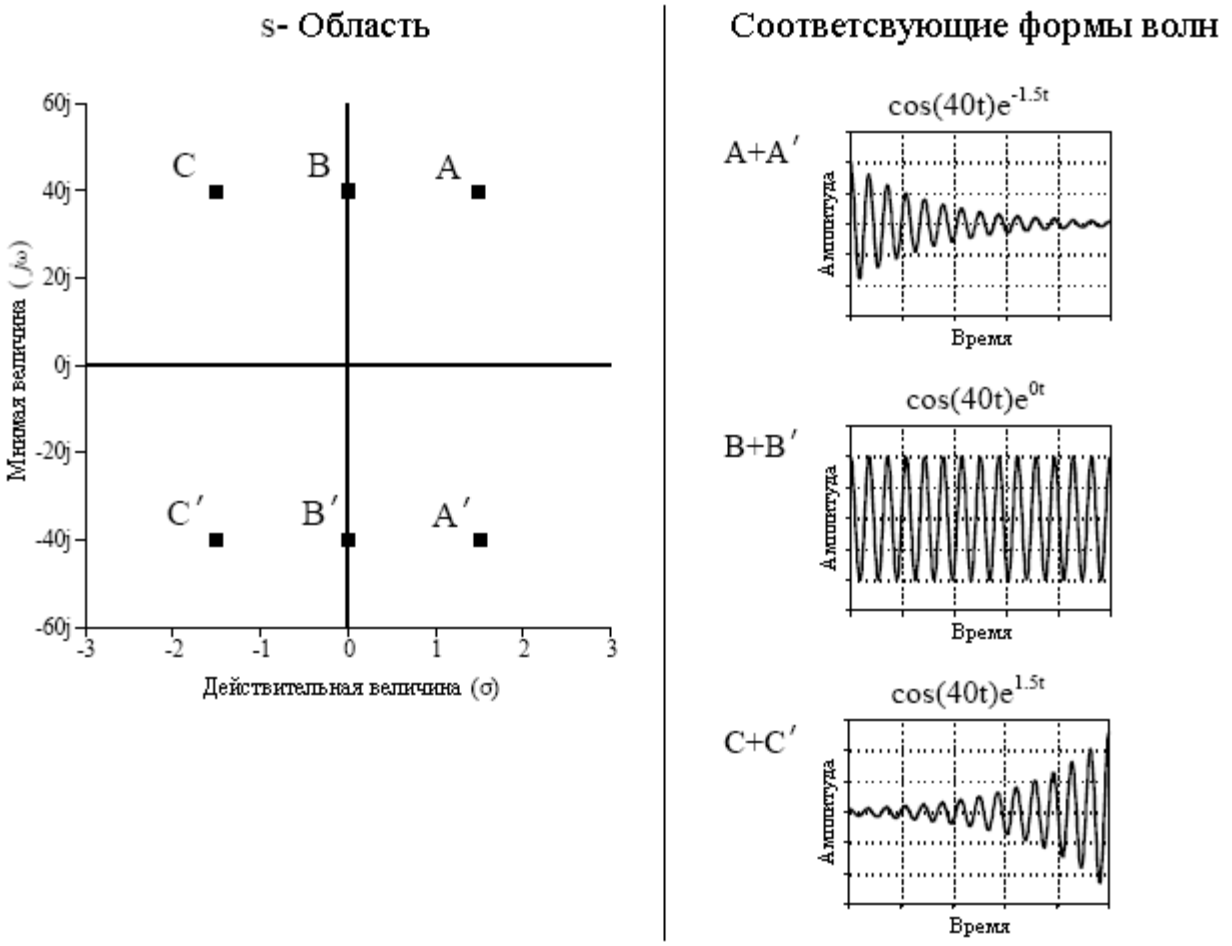


Рис. 32.2 Формы волн ассоциируемые с s-областью.

Значения в каждом месте s-плоскости состоят из *действительной части* и *мнимой части*. Действительная часть находится умножением сигнала временной области на экспоненциально взвешенную косинусоидальную волну с последующим интегрированием от -∞ до +∞. Мнимая часть находится таким же образом, только здесь используется экспоненциально взвешенная синусоидальная волна. В форме уравнения, используя действительную часть пары *A* и *A′*, это выглядит следующим образом:

$$\mathrm{Re}(\sigma = 1{,}5; \pm\omega = 40) = \int_{-\infty}^{+\infty} x(t)\cos(40t)e^{-1{,}5t}\,dt\,.$$

На рис.32.3 показан пример формы волны временной области, ее частотный спектр и ее представление в s-области. В качестве примера взят прямоугольный импульс шириной (длительностью – прим. перев.) равной *двум* и весом (амплитудой – прим. перев.) равным *единице*. Как показано на рисунке, действительная часть комплексного преобразования Фурье этого сигнала представляет собой синк функцию, а мнимая часть сигнала полностью нулевая. S-область представляет собой волнообразный двумерный сигнал, показанный здесь в виде топографических поверхностей действительной и мнимой частей.

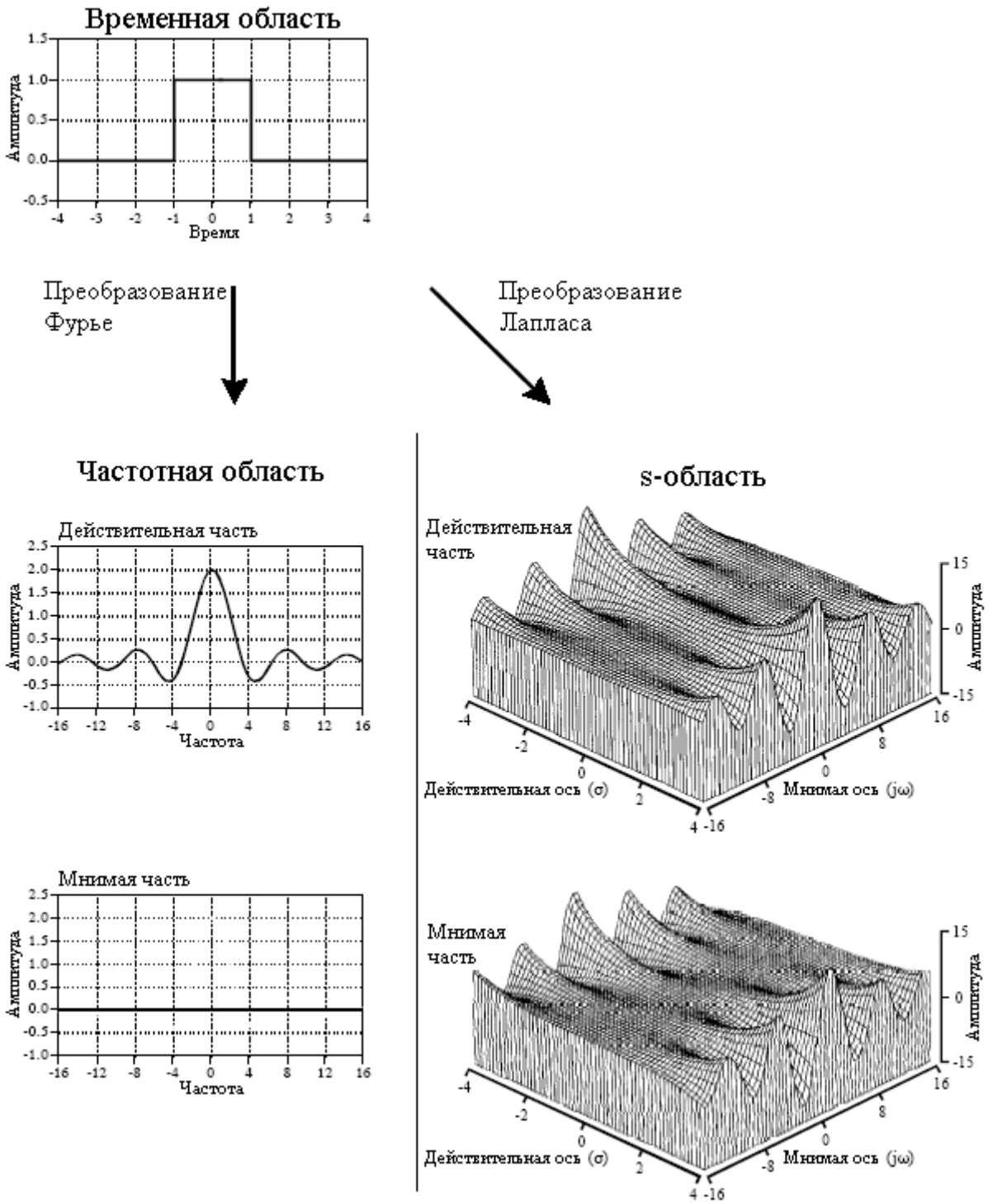


Рис. 32.3 Время, частота и s-области

Математика здесь работает следующим образом:

$$X(s) = \int_{-\infty}^{+\infty} x(t)e^{-st}dt = \int_{-1}^{1} 1e^{-st}dt\,.$$

Выражаясь словами, мы начинаем с определения преобразования Лапласа (уравнение (32.1)), присваиваем *x(t)* значение единицы и изменяем пределы, чтобы получить соответствие с длиной ненулевой части сигнала временной области. Вычисление этого

интеграла дает сигнал s-области, выраженный в виде комплексного местоположения *s* и комплексного значения *X(s)*:

$$X(s) = \frac{e^{s} - e^{-s}}{s}.$$

Хотя это наиболее компактная форма ответа, использование комплексных переменных делает его трудным для понимания и делает невозможным получение графического изображения подобного приведенному на рис. 32.3. Выходом из этой ситуации является замена комплексной переменной *s* на $\sigma + j\omega$ и затем разделение действительной и мнимой частей:

$$\mathrm{Re}\, X(\sigma,\omega) = \frac{\sigma\cos(\omega)\left[e^{\sigma} - e^{-\sigma}\right] + \omega\sin(\omega)\left[e^{\sigma} + e^{-\sigma}\right]}{\sigma^2 + \omega^2},$$

$$\mathrm{Im}\, X(\sigma,\omega) = \frac{\sigma\sin(\omega)\left[e^{\sigma} + e^{-\sigma}\right] - \omega\cos(\omega)\left[e^{\sigma} - e^{-\sigma}\right]}{\sigma^2 + \omega^2}.$$

Топографические поверхности на рис. 32.3 являются графиками этих уравнений. Эти уравнения весьма длинны, а математика для их вывода очень утомительна. Это приводит к практической проблеме: как мы узнаем, что при алгебре такой сложности, мы не допустили ошибки при вычислениях? Один способ контроля - это проверить, что эти уравнения упрощаются до *преобразования Фурье* вдоль оси *y*. Это выполняется приравниванием в уравнениях σ к нулю и упрощением:

$$\mathrm{Re}\, X(\sigma,\omega)\big|_{\sigma=0} = \frac{2\sin(\omega)}{\omega} \qquad \mathrm{Im}\, X(\sigma,\omega)\big|_{\sigma=0} = 0.$$

Как показано на рис. 32.3, это достоверные сигналы частотной области, точно такие же, как найденные непосредственным вычислением преобразования Фурье для формы волны временной области.

Стратегия преобразования Лапласа

То, как используется преобразование Лапласа в обработке сигналов, поможет объяснить аналогия. Представьте, что Вы едите ночью на поезде из одного города в другой. Ваша карта показывает, что путь очень прямой, но ночь так темна, что никакая окружающая местность Вам не видна. Не найдя себе лучшего занятия, Вы замечаете на стене пассажирского вагона альтиметр и решаете следить за изменениями возвышений вдоль пути следования.

Через несколько часов все это Вам надоело, и Вы завязываете беседу с проводником: "Интересный ландшафт." говорите Вы. "Кажется, что мы вообще-то подымаемся в гору, но есть несколько интересных отклонений, которые я заметил." Игнорируя очевидную незаинтересованность проводника, Вы продолжаете: "Недалеко от начала нашей поездки мы миновали какой-то резкий подъем, а за ним такой же резкий спуск. Позже мы встретились с небольшим снижением." Думая, что Вы можете быть опасным или сошли с ума проводник решает ответить: "Да, я полагаю, что это так. Наш пункт назначения расположен в основании большого горного хребта, этим и объясняется

общее увеличение подъема. Однако по пути мы проезжаем по предместьям большой горы и через центр долины."

Теперь, подумайте о том, как понимаете, взаимосвязь между подъемом и расстоянием вдоль пути следования поезда Вы, по сравнению с тем, как это понимает проводник. Поскольку Вы непосредственно измерили возвышение вдоль пути следования, Вы можете справедливо заявить, что Вы знаете *все* относительно этой взаимосвязи. Для сравнения, проводник знает всю ту же самую информацию, но в более простой и более интуитивной форме: расположение холмов и долин, которые являются *причиной* спусков и подъемов по пути следования. В то время как ваше описание сигнала может состоять из тысячи индивидуальных измерений, описание сигнала проводника будет содержать всего несколько параметров.

Для того чтобы показать, насколько это аналогично обработке сигнала, представьте, что мы пытаемся понять характеристики некоторой электрической цепи. Чтобы помочь нашему исследованию, мы тщательно измеряем импульсный отклик и/или частотный отклик. Как обсуждалось в предыдущих главах, импульсный и частотный отклики содержат *полную* информацию о такой линейной системе. Однако это не означает, что Вы познаете информацию *простейшим* способом. В частности, Вы понимаете частотный отклик, как набор значений, изменяющихся с изменением частоты. Частотный отклик может быть понят более легко, так же, как и в нашей аналогии с поездом, в терминах ландшафта, окружающего этот частотный отклик. То есть по характеристикам s-плоскости.

Помня об аналогии с поездом, вернитесь к рис. 32.3 и задайте вопрос: как очертания данной s-области помогают в понимании частотной характеристики? Ответом будет, не помогают никак! В этом примере s-плоскость представляет собой красивый график, но не дает правильного представления, почему частотная область ведет себя так, как она это делает. Это связано с тем, что преобразование Лапласа разработано для анализа специфического класса сигналов временной области: *импульсного отклика*, *состоящего из синусоид и экспонент*. Если преобразование Лапласа берется от некоторых других форм волн (таких как прямоугольный импульс на рис. 32.3), результирующая s-плоскость бессмысленна.

Как упоминалось во введении, в науке и инженерной практике системы, принадлежащие к вышеупомянутому классу весьма типичны. Это является следствием того, что синусоиды и экспоненты являются решениями *дифференциальных уравнений*, математики управляющей большей частью нашего физического мира. Например, все следующие системы управляются дифференциальными уравнениями: электрические цепи, распространение волн, поступательное и вращательное движение, электрические и магнитные поля, тепловой поток и т.д.

Представьте, что мы пытаемся понять некоторую линейную систему, которая управляется дифференциальными уравнениями такую, как электрическая цепь. Решение дифференциальных уравнений дает математический способ нахождения импульсного отклика. В качестве альтернативы мы могли бы измерить импульсный отклик, используя подходящие импульсные генераторы, осциллографы, регистраторы данных и т.п. Перед тем как мы просмотрим только что найденную импульсную характеристику, мы спрашиваем себя, что мы там *ожидаем* найти. Существует несколько характеристик форм волн, которые мы знаем, даже не глядя. Во-первых, импульсный отклик должен быть *причинным*. Другими словами импульсный отклик должен иметь значение равное нулю до тех пор, пока на входе не появится ненулевая величина в момент $t = 0$. Это ни что иное, как причина и следствие - то, на чем основана наша вселенная.

Вторая вещь, которую мы знаем об импульсной характеристике, это - то, что она будет состоять из *синусоид* и *экспонент* потому, что они являются решениями дифференциальных уравнений, которые управляют системой. Сколько бы мы не пытались, мы никогда не найдем эдакий тип системы, который имеет импульсный отклик

такой, как, например, прямоугольный импульс или волна треугольной формы. В-третьих, импульсный отклик будет *бесконечным* по длине. То есть он имеет ненулевые значения, которые простираются от $t = 0$ до $t = +\infty$. Это связано с тем, что синусоидальные и косинусоидальные волны имеют постоянную амплитуду, а экспоненциальное затухание стремится к нулю, но никогда его не достигнет. Если система, которую мы исследуем, является **устойчивой**, амплитуда импульсного отклика будет с течением времени уменьшаться, достигая нулевого значения при $t = +\infty$. Существует также возможность того, что система является **неустойчивой,** например, усилитель, который спонтанно возбуждается из-за чрезмерной глубины обратной связи. В этом случае импульсный отклик будет с течением времени *увеличиваться* по амплитуде, становясь бесконечно большим. Даже малейшие возмущения, приложенные к такой системе, дадут неограниченное значение на выходе.

Общая математика преобразования Лапласа очень похожа на математику преобразования Фурье. В обоих случаях заранее определенные формы волны умножаются на сигнал временной области, и результат интегрируется. На первый взгляд может показаться, что стратегия преобразования Лапласа такая же, как и преобразования Фурье: коррелировать сигнал временной области с набором базисных функций для осуществления декомпозиции формы волны. Но это не так! Даже если математика во многом такая же, смысл, стоящий за этими двумя техниками, слишком разный. Преобразование Лапласа *зондирует* форму волны временной области для определения ее ключевых характеристик: *частот* синусоид, и *постоянных затухания* экспонент. Следующий пример покажет, как это все работает.

Фильтр-пробка имеет два полюса (представлены ×) и два нуля (представлены ○).

Рис. 32.4 Пример полюсов и нулей

В центральном столбце на рис. 32.5 показан импульсный отклик *R-L-C* фильтра – “пробки”, обсуждавшегося в Главе 30. Он содержит импульс при $t=0$, за которым следует экспоненциально затухающая синусоида. Как показано, на рисунках от (а) до (е) мы будем *зондировать* этот импульсный отклик различными экспоненциально затухающими синусоидами. Каждая из этих зондирующих форм волн характеризуется двумя параметрами: ω, определяющая частоту синусоиды, и σ, определяющая скорость затухания. Другими словами, как это показано на чертеже s – плоскости на рис. 32.4, каждая зондирующая форма волны соответствует различным, обозначенным на рисунке черным квадратом, положениям на s – плоскости. Импульсный отклик зондируется *перемножением* его на форму волны и последующим интегрированием результата от $t=-\infty$ до $+\infty$. Это действие показано в правом столбце. Нашей целью является нахождение комбинаций σ и ω, которые точно погасят исследуемый импульсный отклик. Такое гашение может происходить в двух формах: площадь, ограниченная кривой может быть

либо *нулевой*, либо *практически бесконечной*. Все другие результаты не представляют интереса и могут быть игнорированы. Места на s – плоскости, которые дают нулевую форму гашения, называются **нулями** системы. Аналогичным образом, места, которые дают "практически бесконечную" форму гашения, называются **полюсами**. Полюса и нули аналогичны, в нашей истории с поездом, горам и долинам, представляющим ландшафт “вокруг” частотного отклика.

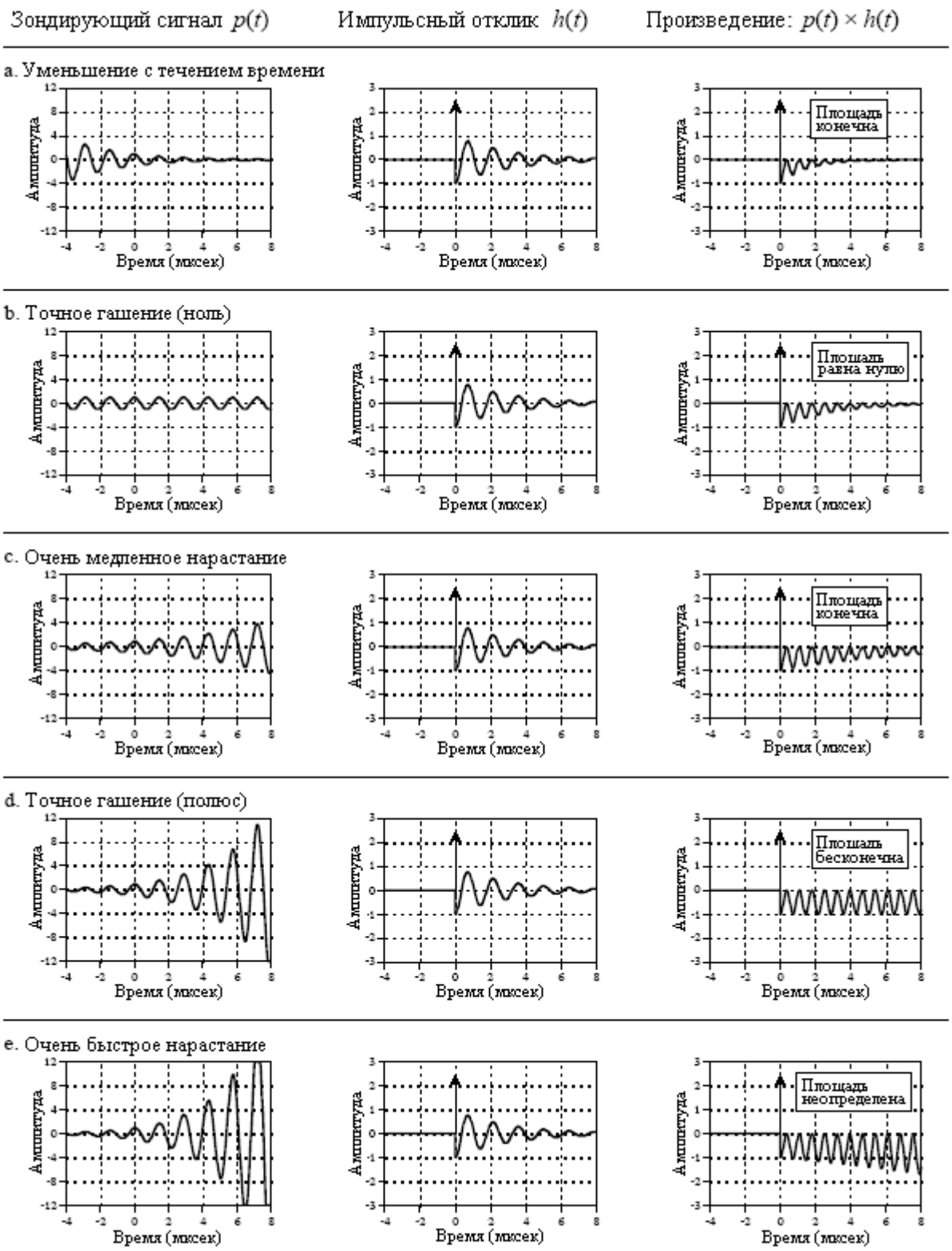


Рис. 32.5 Исследование импульсного отклика

Для начала рассмотрим, что происходит, когда форма зондирующей волны с течением времени уменьшается по амплитуде, как показано на рисунке (a). Это происходит тогда, когда $\sigma>0$ (правая половина s – плоскости). Поскольку оба и импульсный отклик, и зондирующий сигнал с течением времени уменьшаются, их произведение будет иметь то же свойство. Когда произведение двух волн будет проинтегрировано от минус до плюс бесконечности, результатом будет некоторое число, не представляющее особого интереса. В частности уменьшающийся зондирующий сигнал *не может* погасить уменьшающийся импульсный отклик. Это означает, что устойчивая система не будет иметь полюсов с $\sigma>0$. Другими словами, все полюса в устойчивой системе ограничены левой половиной s-плоскости. Фактически полюса в правой половине s – плоскости показывают, что система является неустойчивой (т.е. импульсный отклик с течением времени *нарастает*).

На рисунке (b) показан один из частных случаев, которые мы ищем. Когда эта форма волны перемножается с импульсным откликом, результирующий интеграл имеет значение, равное нулю. Это происходит потому, что площадь выше оси x (от дельта функции) точно равна площади ниже оси x (от выпрямленной синусоиды). Значения для σ и ω, дающие такую форму гашения, называют *нулем* системы. Как показано на чертеже s – плоскости на рис. 32.4, нули обозначаются небольшим кружочком (o).

На рисунке (c) показан следующий тест, который мы можем проделать. Здесь мы используем синусоиду, которая с течением времени экспоненциально *нарастает*, но *медленнее*, чем *уменьшается* с течением времени импульсный отклик. В произведении двух форм волн это также приводит к его уменьшению с течением времени. Как и в случае (a), это приводит к тому, что интеграл произведения есть некоторое неинтересное действительное число. Важным моментом является то, что здесь нет никакой формы точного гашения.

Нарушая порядок, посмотрите на рисунок (e), где представлена зондирующая форма волны, нарастающая *быстрее*, чем уменьшается импульсный отклик. Результирующий после перемножения сигнал с течением времени увеличивается по амплитуде. Это означает, что площадь, ограниченная кривой, становится больше с течением времени и общая площадь от t=-∞ до +∞ не определена. На математическом жаргоне, *интеграл не сходится*. Другими словами не во всех областях s – плоскости имеются конкретные значения. Часть s – плоскости, в которой интеграл вычисляется, называется **областью сходимости**. В некоторых математических методах, важно знать, какая часть s – плоскости находится в пределах области сходимости. Однако для приложений этой книги такая информация не требуется. Для данной дискуссии интересно только точное гашение.

На рисунке (d) зондирующая форма волны нарастает с точно такой же скоростью, что и уменьшается импульсный отклик. Это приводит к тому, что произведение двух форм волн имеет постоянную амплитуду. Другими словами - это разделительная линия между (c) и (e), дающая общую область, являющуюся просто *почти неопределенной* (да простят математики такое неточное описание). Выражаясь более точно, это - точка, находящаяся на границе области сходимости. Как уже упоминалось, значения σ и ω, которые дают эту форму точного гашения, называются полюсами системы. На s – плоскости полюса отмечаются крестиками (×).

Анализ электрических схем

Описывая то, как выглядят формы волн и как ими манипулируют, мы ввели преобразование Лапласа в графических терминах. Это наиболее интуитивный способ понимания подхода, но он весьма далек от того, как этот подход применяется в действительности. Преобразование Лапласа является полностью математическим методом, оно используется для написания и манипулирования математическими

уравнениями. Сложность здесь заключается в том, что в абстрактном характере комплексной алгебры легко потеряться и потерять все связи с реальным миром. Ваша задача состоит в том, чтобы соединить эти два представления вместе. Преобразование Лапласа является основным методом анализа электрических цепей. Имейте в виду, что с помощью этого метода может быть обработана *любая* система, подчиняющаяся дифференциальным уравнениям; электрические цепи являются просто примером, который мы сейчас используем.

Получение импульсного отклика системы способом “в лоб” заключается в решении дифференциальных уравнений, которым подчиняется эта система. Затем, импульсный отклик при помощи уравнения (32.1) может быть преобразован в *s*–область. К счастью, существует более простой путь: преобразовать каждую индивидуальную компоненту в s-область, а затем вычислить то, как они взаимодействуют. Это очень похоже на фазорное преобразование, представленное в Главе 30, где резисторы, катушки индуктивности и конденсаторы представляются как R, $j\omega L$ и $1/j\omega C$, соответственно. В преобразовании Лапласа резисторы, катушки индуктивности и конденсаторы становятся комплексными переменными - R, sL и $1/sC$. Обратите внимание, что фазорное преобразование является *частным случаем* преобразования Лапласа. То есть, когда σ в $s=\sigma+j\omega$ равно нулю, R становится R, sL становится $j\omega L$ и $1/sC$ становится $1/j\omega C$.

Так же, как и в Главе 30, мы будем обращаться с каждой из этих трех компонент, как с индивидуальной системой, рассматривая ток в качестве входного сигнала, а напряжение в качестве выходного сигнала. Когда мы говорим, что резисторы, катушки индуктивности и конденсаторы стали R, sL и $1/sC$, в *s*–области это относится к выходному сигналу, деленному на входной. Другими словами, данному выражению равно преобразование Лапласа *формы волны напряжения*, деленное на преобразование Лапласа *формы волны тока*.

В качестве примера этому, представьте, что мы пропускаем косинусоидальный ток единичной амплитуды с частотой ω_0 через катушку индуктивности. Результирующая форма волны *напряжения*, падающего на катушке индуктивности, может быть найдена решением дифференциального уравнения, которому подчиняется работа катушки:

$$v = L\frac{d}{dt}i(t) = L\frac{d}{dt}\cos(\omega_0 t) = \omega_0 L\sin(\omega_0 t).$$

Если мы начнем с формы волны тока в момент времени $t=0$, форма волны напряжения начнется также с того же самого момента (т.е. $i(t)=0$ и $v(t)=0$ для $t<0$). Эти формы волн напряжения и тока преобразуются в *s*–область с помощью уравнения (32.1):

$$I(s) = \int_0^\infty \cos(\omega_0 t)e^{-st}dt = \frac{\omega_0}{\omega_0^2 + s^2};$$

$$V(s) = \int_0^\infty \omega_0 L\sin(\omega_0 t)e^{-st}dt = \frac{\omega_0 Ls}{\omega_0^2 + s^2}.$$

Для того чтобы завершить этот пример, разделим напряжение *s*–области на ток *s*–области так, как если бы мы использовали закон Ома ($R=V/I$):

$$\frac{V(s)}{I(s)} = \frac{\dfrac{\omega_0 Ls}{\omega_0^2 + s^2}}{\dfrac{\omega_0}{\omega_0^2 + s^2}} = sL.$$

Мы нашли, что представление напряжения, падающего на катушке индуктивности в *s*–области, деленное на представление тока протекающего через катушку индуктивности в *s*–области, равно *sL*. Такой результат будет *всегда*, вне зависимости от того, с какого момента формы волны тока мы начнем. Аналогично, отношение напряжения *s*–области к току *s*–области всегда равно *R* для резисторов и *1/sC* для конденсаторов.

На рис. 32.6 показан пример цепи, которую мы проанализируем с помощью преобразования Лапласа, *R-L-C* фильтр-пробка, обсуждавшийся в Главе 30. Поскольку для всех электрических цепей этот анализ одинаков, мы будем описывать его по шагам.

Шаг 1. Преобразуем каждую компоненту в s–область. Другими словами, заменим значение каждого резистора на - *R*, каждой катушки индуктивности на - *sL* и каждого конденсатора на - *1/sC*. Это показано на рис. 32.6.

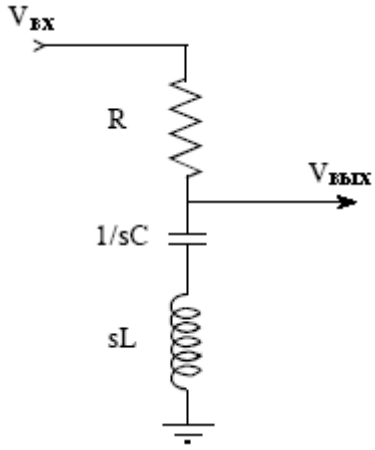


Рис. 32.6 Анализ фильтра-пробки в s-области.

Шаг 2. Найдем H(s), выходной сигнал, деленный на входной. Как описывалось в Главе 30, это выполняется за счет обращения с каждым из компонентов так, как с компонентом, подчиняющимся закону Ома с сопротивлением, задаваемым *R*, *sL* и *1/sC*. Это позволяет нам пользоваться стандартными уравнениями для последовательно соединенных сопротивлений, параллельно соединенных сопротивлений, делителей напряжения и т.д. Рассматривая *R-L-C* цепь в этом примере, как делитель напряжения (так же, как и в Главе 30), находим *H(s)*:

$$H(s) = \frac{V_{вых}(s)}{V_{вх}(s)} = \frac{sL + 1/sC}{R + sL + 1/sC} = \frac{sL + 1/sC}{R + sL + 1/sC} \cdot \left[\frac{s}{s}\right] = \frac{Ls^2 + 1/C}{Ls^2 + Rs + 1/C}.$$

Как Вы помните из анализа *Фурье*, частотный спектр выходного сигнала, деленный на частотный спектр входного сигнала, равен *частотному отклику* системы, обозначаемому символом *H(ω)*. Приведенное выше уравнение является его расширением на s-область. Сигнал *H(s)* называется **передаточной функцией** системы и эквивалентен представлению выходного сигнала в s-области, деленному на представление входного

сигнала в s-области. Далее, *H(s)* эквивалентна *преобразованию Лапласа* от импульсного отклика, точно так же, как *H(ω)* эквивалентна *преобразованию Фурье* от импульсного отклика.

До сих пор, все это было идентично методу предыдущей главы, за исключением использования *s* вместо *jω*. Отличием между этими двумя методами будет то, что произойдет, начиная с этого момента. Это значительно глубже того, чего мы можем достичь с *jω*. Мы можем графически изобразить частотный отклик или исследовать его каким-либо другим образом, однако это является математическим тупиком. Интересные аспекты преобразования Лапласа только что начались, сравните. Обнаружение *H(s)* является ключом к Лапласовому анализу, однако, для того чтобы передаточную функцию можно было использовать, она должна быть выражена в определенной форме. Это требует алгебраических преобразований двух следующих шагов.

Шаг 3. Запишите H(s) так, чтобы один полином располагался над другим. Это даст следующую запись передаточной функции:

$$H(s) = \frac{as^2 + bs + c}{\mathrm{a}s^2 + \mathrm{b}s + \mathrm{c}}. \qquad (32.2)$$

Если система подчиняется дифференциальному уравнению, то передаточную функцию всегда можно выразить в такой форме. Например, прямоугольный импульс, показанный на рис. 32.3, не является решением дифференциального уравнения и его преобразование Лапласа не может быть записано таким способом. Для сравнения, любая электрическая цепь, состоящая из резисторов, конденсаторов и катушек индуктивности, может быть описана в такой форме. Для *R-L-C* фильтра-пробки, используемого в этом примере, алгебраические преобразования, показанные в *шаге 2*, уже представили передаточную функцию в правильной форме, т.е.:

$$H(s) = \frac{as^2 + bs + c}{\mathrm{a}s^2 + \mathrm{b}s + \mathrm{c}} = \frac{Ls^2 + 1/C}{Ls^2 + Rs + 1/C},$$

где $a=L$, $b=0$, $c=1/C$ и a=*L*, b= *R*, c=*1/C*.

Шаг 4. Разложите полиномы числителя и знаменателя на сомножители. Это разобьет полиномы числителя и знаменателя на составляющие, каждая из которых содержит одно *s*. При перемножении составляющих друг с другом они должны дать результат эквивалентный исходным числителю и знаменателю. Другими словами уравнение приводится к виду:

$$H(s) = \frac{(s - z_1)(s - z_2)(s - z_3)\cdots}{(s - p_1)(s - p_2)(s - p_3)\cdots}. \qquad (32.3)$$

Корни числителя z_1, z_2, z_3, …, являются **нулями** уравнения, в то время как корни знаменателя p_1, p_2, p_3, …, являются **полюсами**. Это те же самые нули и полюса, с которыми мы встречались ранее в этой главе, в следующем разделе мы будем обсуждать, как они используются.

Если числитель и знаменатель является *полиномами второго порядка* или менее, разложение выражения s-области на множители осуществляется непосредственно. Другими словами, мы можем легко обращаться с членами *s* и s^2, но не с членами s^3, s^4, s^5, …. Это связано с тем, что корни полиномов второго порядка ax^2+bx+c могут быть

найдены при помощи квадратного уравнения $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. С помощью этого метода передаточная функция фильтра-пробки из нашего примера разлагается на:

$$H(s) = \frac{(s - z_1)(s - z_2)}{(s - p_1)(s - p_2)}$$

где $z_1 = \frac{j}{\sqrt{LC}}$, $p_1 = \frac{-R + \sqrt{R^2 - 4L/C}}{2L}$,

$z_2 = \frac{-j}{\sqrt{LC}}$, $p_2 = \frac{-R - \sqrt{R^2 - 4L/C}}{2L}$.

Так же, как в этом примере система второго порядка имеет максимум два нуля и два полюса. Количество полюсов в системе равно числу независимых компонент, запасающих энергию. Например, катушки индуктивности и конденсаторы запасают энергию, в то время как резисторы нет. Число нулей может быть равно или меньше, чем число полюсов.

Полиномы более высокого, чем второго порядка в общем случае не могут быть разложены с использованием только алгебры, для этого требуются более сложные численные методы. Как альтернатива, схемы могут быть построены в виде *каскадов* из звеньев *второго порядка*. Хорошим примером этому является семейство аналоговых фильтров, представленных в Главе 3. Например, восьми полюсный фильтр строится из четырех последовательно соединенных звеньев второго порядка каждое. Важным моментом здесь является то, что этот многоступенчатый подход используется для преодоления ограничений в *математике*, а не ограничений в *электронике*.

Важность полюсов и нулей

Чтобы сделать повествование менее абстрактным, будем использовать реальные значения компонент фильтра-пробки, который мы только что анализировали: $R=220\Omega$, $L=54\mu H$, $C=470pF$. Подстановка этих значений в выше приведенное уравнение, дает размещение нулей и полюсов:

$z_1=0+j6{,}277\times10^6$ $\quad$ $p_1=-2{,}037\times10^6+j5{,}937\times10^6$,

$z_2=0-j6{,}277\times10^6$ $\quad$ $p_1=-2{,}037\times10^6-j5{,}937\times10^6$.

Такое положение полюсов и нулей показано на рис. 32.7. Каждый ноль представлен кружочком, в то время как полюс представлен крестиком. Это называется **диаграммой нулей и полюсов**, и является наиболее типичным способом, с помощью которого показываются данные s-области. На рис. 32.7 показан также топографический вид s-плоскости. Для простоты показаны только амплитуды, но не следует забывать, что есть еще и соответствующие фазы. Так же, как горы и долины определяют форму поверхности земли, так же полюса и нули определяют форму поверхности s-плоскости. Но в отличии от гор и долин каждый полюс и ноль имеет точно такую же форму и размер, как и каждый другой полюс и ноль. Единственной уникальной характеристикой, которую

имеют нули и полюса, является их *местоположение*. Важность полюсов и нулей заключается в том, что они дают сжатое представление о значении *в любой точке на s-плоскости*. То есть, мы можем полностью описать характеристики системы, используя всего *несколько параметров*. В случае *R-L-C* фильтра-пробки, для представления системы нам нужно определить всего четыре комплексных параметра: z_1, z_2, p_1, p_2 (каждый состоит из действительной и мнимой части).

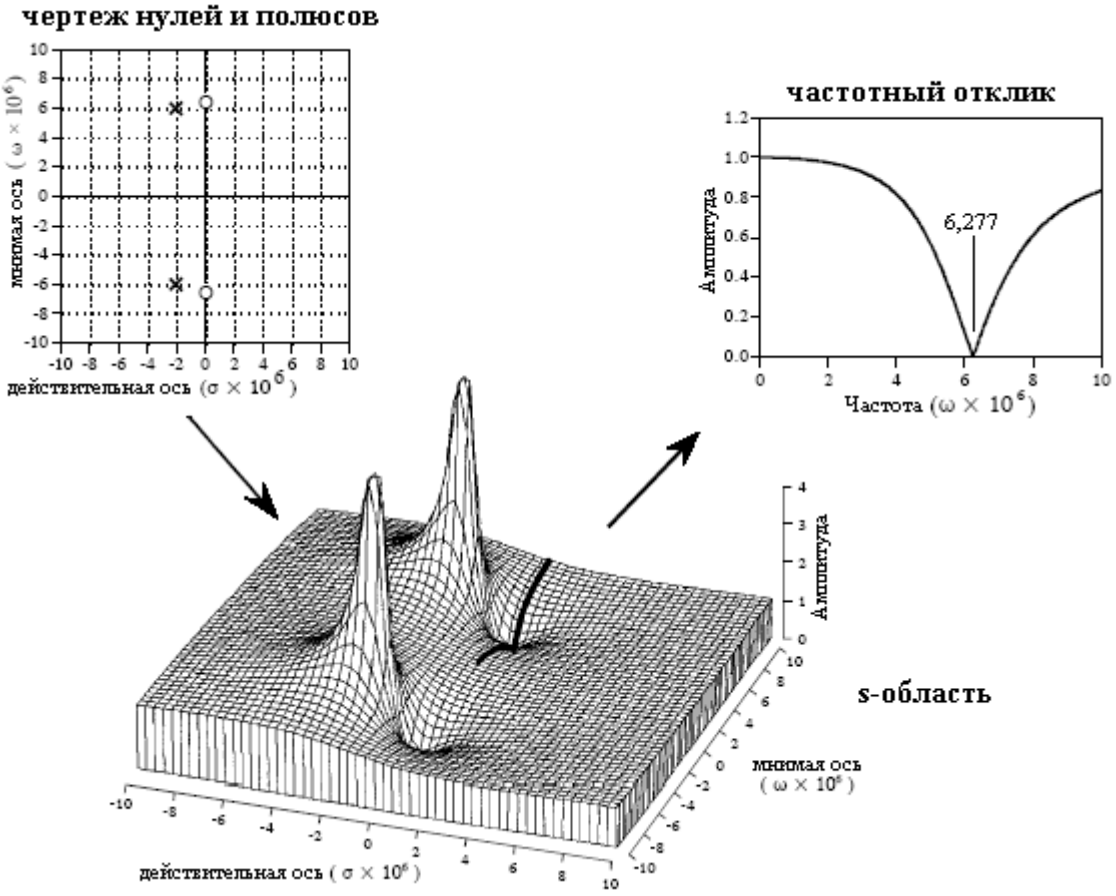


Рис. 32.7 Полюса и нули в s-области

Для лучшего понимания полюсов и нулей, представьте муравья ползущего по s-плоскости. Любому конкретному месту, где случается оказываться муравью (т.е. некоторому значению s), соответствует надлежащее значение передаточной функции *H(s)*. Значение передаточной функции представляет собой комплексное число, которое может быть выражено в виде модуля и фазы или в виде действительной и мнимой частей. Теперь пусть муравей перенесет нас к одному из нулей на *s*-плоскости. В этом месте значения действительной и мнимой частей передаточной функции, которые мы измеряем, будут равны нулю. Это можно понять, исследуя уравнение (32.3) для *H(s)*. Если местоположение *s* равно любому из нулей, то один из членов в числителе будет равен нулю. Это приводит к тому, что все выражение тоже становится равным нулю вне зависимости от других величин.

Далее, путешествие нашего муравья переносит нас к одному из полюсов, где мы снова измеряем значение действительной и мнимой частей *H(s)*. Измеряемое значение становится все больше и больше по мере того, как мы приближаемся к более точному местоположению полюса (отсюда и название). Это тоже может быть понято из уравнения (32.3). Если местоположение *s* равно любому из полюсов, знаменатель будет равен нулю, а деление на ноль делает все выражение бесконечно большим.

Исследовав уникальные места, путешествие нашего муравья продолжается по s-плоскости в произвольном направлении. Значение *H(s)* в каждом месте полностью зависит от относительного расположения полюсов и нулей, поскольку в этом странном ландшафте никаких других допустимых особенностей нет. Если мы находимся вблизи полюса, значение передаточной функции будет большим, если мы находимся вблизи нуля, значение передаточной функции будет маленьким.

Уравнение (32.3) описывает также взаимодействие *многочисленных* полюсов и нулей при формировании сигнала s-области. Помните, разница двух комплексных чисел дает *расстояние* между ними на комплексной плоскости. Например, $(s\text{-}z_0)$ является расстоянием между произвольным местоположением s и нулем, расположенным в z_0. Следовательно, уравнение (32.3) определяет для каждого местоположения s величину, равную *произведению* расстояний до всех нулей, деленную на *произведение* расстояний до всех полюсов.

Это приводит нас к сердцу этой главы, к тому, каким образом местоположение полюсов и нулей дает более глубокое понимание *частотного отклика* системы. Частотный отклик равен значению *H(s)* вдоль мнимой оси, на топографическом чертеже на рис. 32.7 он обозначен жирной черной линией. Представьте, что наш муравей стартует от начала координат и ползет вдоль этого пути. Вблизи начала координат расстояние до нулей приблизительно равно расстоянию до полюсов. Это приводит к сокращению числителя и знаменателя в уравнении (32.3), дающему частотную характеристику, равную единице на низких частотах. Ситуация не претерпевает значительных изменений до тех пор, пока муравей не начинает двигаться вблизи местоположения полюса и нуля. При приближении к нулю значение *H(s)* резко падает, становясь *нулевым* в момент, когда муравей опускается до *нуля*. Как только муравей проходит пару полюса и нуля, значение *H(s)* снова возвращается к единице. Используя такой способ визуального представления можно заметить, что ширина впадины зависит от расстояния между полюсом и нулем.

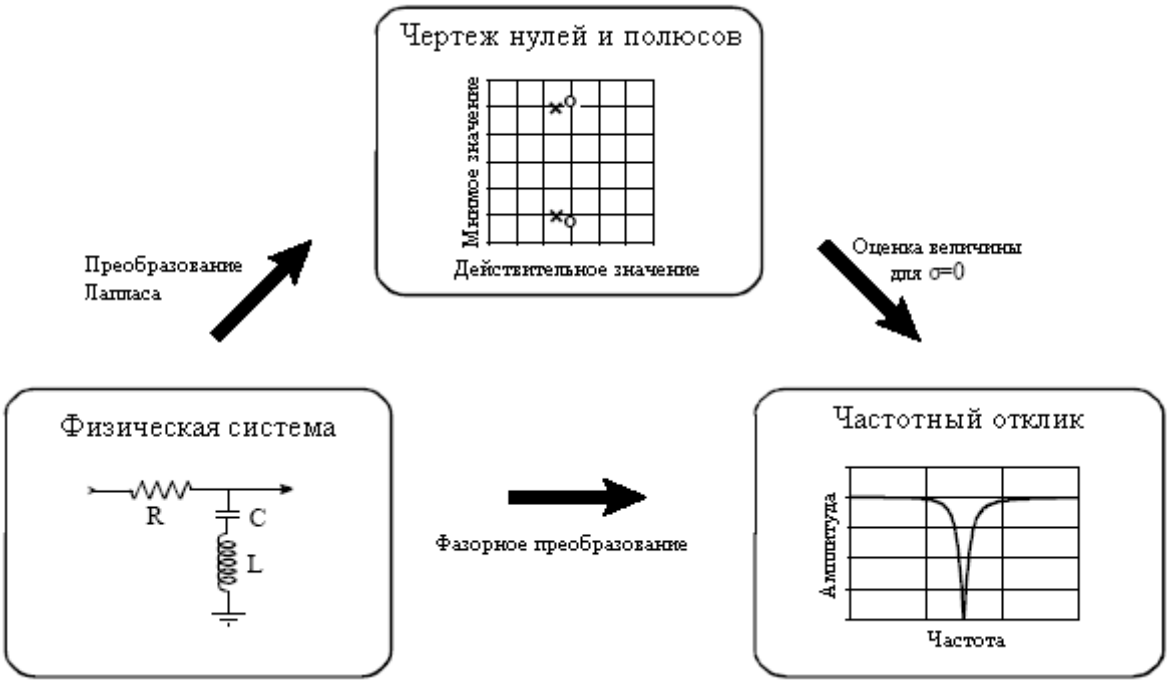


Рис. 32.8 Стратегия использования преобразования Лапласа

Рисунок 32.8 суммирует то, каким образом пользуются преобразованием Лапласа. Начинаем мы с физической системы, такой как электрическая цепь. Если мы желаем, то фазорное преобразование, как описано в Главе 30, может непосредственно дать нам

частотный отклик системы. Альтернативой является взятие преобразования Лапласа с использованием описанного ранее четырех шагового метода. Это приводит к математическому выражению для передаточной функции *H(s)*, которое может быть представлено в виде чертежа нулей и полюсов. Затем, с помощью оценки передаточной функции вдоль мнимой оси, т.е. заменой каждого *s* на *jω*, может быть найден частотный отклик. Несмотря на то, что оба метода приводят к одному и тому же результату, промежуточный чертеж нулей и полюсов обеспечивает понимание того, *почему* система ведет себя так, как она это делает, и каким образом это все может быть изменено.

Разработка фильтров в s-области

Наиболее мощным приложением преобразования Лапласа является проектирование систем *непосредственно* в *s*-области. Оно включает два этапа: во-первых, с помощью определения количества и местоположения полюсов и нулей осуществляется проектирование *s*-области. Это чисто математическая задача, целью которой является получение наилучшего частотного отклика. На втором этапе разрабатывается электронная схема, обеспечивающая такое представление s-области. Последнее является чем-то похожим на искусство, поскольку существует большое число схемных конфигураций обладающих заданным расположением плюсов и нулей.

Как было отмечено ранее, если система содержит более, чем два полюса и два нуля, *шаг 4* метода преобразования Лапласа является очень трудным. Типичным решением здесь является реализация множественных полюсов и нулей в *последовательных каскадах*. Например, шести полюсный фильтр реализуется в виде трех последовательных каскадов, каждый из которых содержит до двух полюсов и двух нулей. Поскольку каждый из этих каскадов может быть представлен в s-области посредством квадратичного числителя, деленного на квадратичный знаменатель, этот подход назван проектированием с **биквадратами**.

На рис. 32.9 показана типичная биквадратная схема, та, что использовалась в методе проектирования фильтра из Главы 3. Она называется схемой **Саллена-Ки**, после того, как в середине 1950-х авторы Р.П. Саллен и Е.Л. Ки опубликовали статью, описывавшую эту технику. Хотя существует несколько разновидностей, в наиболее типичной схеме используются два резистора одинакового номинала, два конденсатора одинакового номинала и усилитель с коэффициентом усиления лежащим между 1 и 3. Недоступные Саллену и Ки усилители сегодня могут быть сделаны из недорогих операционных усилителей с соответствующими резисторами в обратной связи. Опуская четыре шага процедуры схемотехнического анализа, местоположение двух полюсов данной схемы может быть выражено через значения ее компонент:

$$\sigma = \frac{A-3}{2RC},$$
$$\omega = \frac{\pm\sqrt{-A^2+6A-5}}{2RC}. \qquad (32.4)$$

Данные уравнения показывают, что оба полюса всегда лежат где-то на окружности радиуса *1/RC*. Точное местоположение на окружности зависит от величины усиления усилителя. Как показано на рис. 32.9а, при усилении равном 1 оба полюса располагаются на действительной оси. Частотным откликом подобной конфигурации является отклик низкочастотного фильтра с относительно сглаженным переходом между полосой

пропускания и подавления. Частота среза этой схемы –3dB (0,797) отмечена символом ω_0, и находится там, где окружность пересекает мнимую ось, т.е. $\omega_0=1/RC$.

По мере увеличения усиления, полюса перемещаются по окружности с соответствующим изменением частотного отклика. Как показано на рис. 32.9b, усиление величиной 1,586 перемещает полюса на угол 45 градусов, приводя к более крутому переходу в частотном отклике. Увеличение усиления передвигает полюса дальше, еще ближе к мнимой оси, давая пик в кривой показанного частотного отклика. Эти условия иллюстрируются на рис. 32.9c, где усиление равно 2,5. По мере дальнейшего увеличения усиления амплитуда пика продолжает расти до тех пор, пока не будет достигнуто усиление равное 3. Как показано на рис. 32.9d, это особый случай, когда полюса расположены непосредственно на мнимой оси. Теперь на соответствующем частотном отклике имеется пик с бесконечно большим значением. На практике это означает, что схема превратилась в генератор. Дальнейшее увеличение усиления толкает полюса дальше в правую половину s-плоскости. Как отмечалось ранее, это соответствует системе в неустойчивом состоянии (спонтанное генерирование).

Используя схему Саллена-Ки как строительный блок, может быть построено большое разнообразие типов фильтров. Например, низкочастотный **фильтр Баттерворта** строится за счет равномерного размещения выбранного числа полюсов на левой половине окружности так, как показано на рис. 32.10. На каждые два полюса в этой конфигурации требуется одна ступень Саллена-Ки.

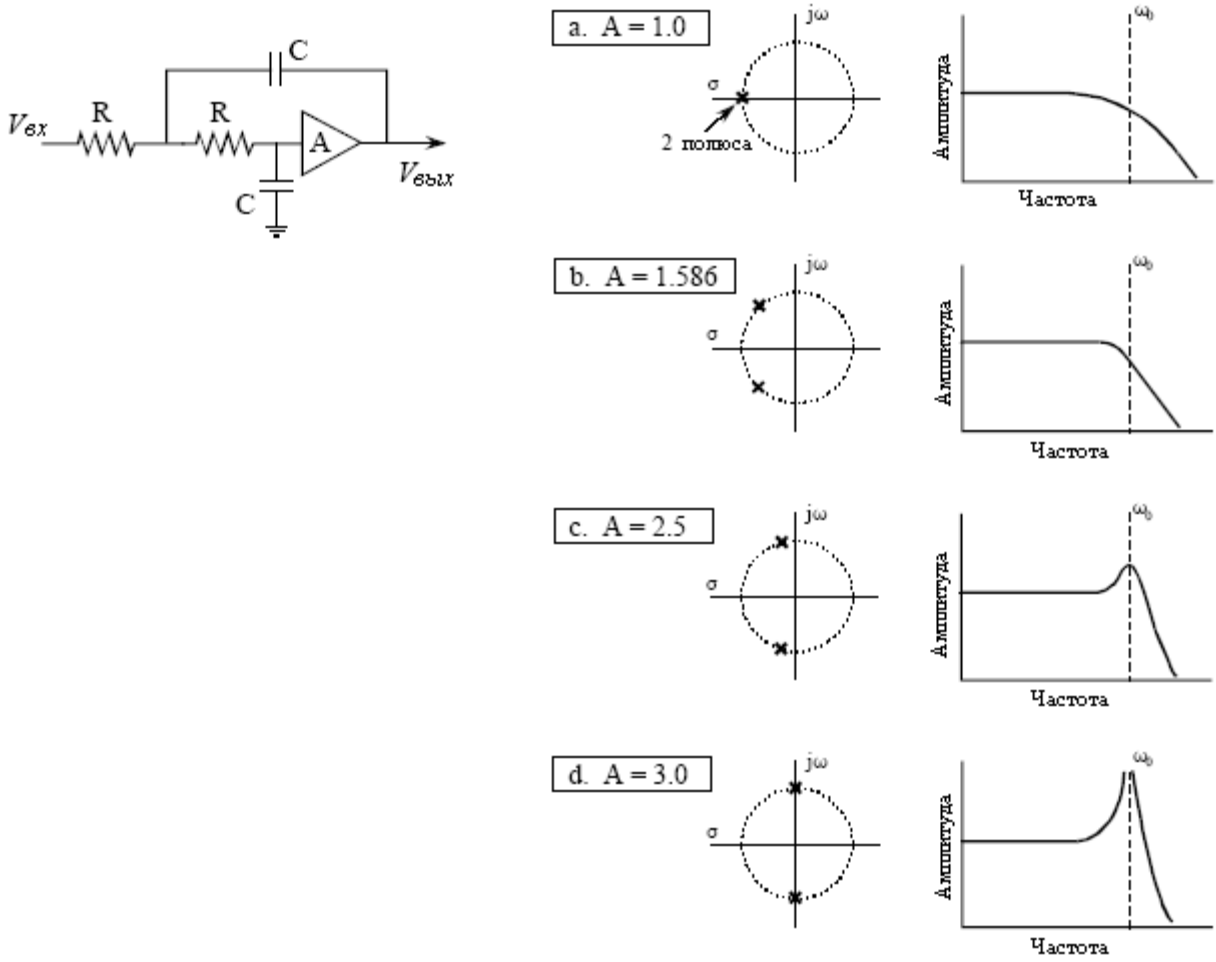


Рис. 32.9 Характеристики Саллена-Ки

Как описывалось в Главе 3, фильтр Баттерворта является максимально плоским, т.е. у него наиболее крутой переход между полосой пропускания и подавления при *отсутствии пиков* в частотном отклике. Чем больше используется полюсов, тем больше крутизна перехода. Поскольку все плюса в фильтре Баттерворта лежат на одной и той же окружности, во всех последовательно соединенных ступенях используются одни и те же значения для *R* и *C*. Единственным отличием между ступенями является усиление. Почему же этот набор расположенных по окружности полюсов дает оптимально плоский частотный отклик? Не ищите очевидного или интуитивного ответа на этот вопрос, он просто вытекает из математики.

Рисунок 32.11 показывает, как могут быть изменены положения полюсов Фильтра Баттерворта, чтобы сделать **фильтр Чебышева**. Как обсуждалось в Главе 3, в фильтре Чебышева за счет допустимых пульсаций в полосе пропускания достигается более крутой переход, чем у Баттерворта. В s-области это соответствует сплющиванию окружности из полюсов в *эллипс*. Чем больше сплющен эллипс, тем больше пульсаций в полосе пропускания, и больше крутизна перехода. При построении фильтра из последовательно соединенных ступеней Саллена-Ки, резисторы и конденсаторы в каждой ступени имеют различные значения.

Рисунок 32.11 показывает также следующий уровень сложности в стратегии проектирования фильтра, **эллиптический фильтр**. В эллиптическом фильтре за счет допустимых пульсаций, как в полосе пропускания, так и в полосе подавления достигается наиболее крутой переход из возможных. В s-области это соответствует размещению нулей на мнимой оси, причем первый находится около частоты среза. Существует несколько классов эллиптических фильтров, и они значительно более сложны в проектировании, чем конфигурации Баттерворта и Чебышева. Это связано с тем, что полюса и нули эллиптического фильтра не лежат на простой геометрической фигуре, а их математическое описание включает эллиптические функции и интегралы (отсюда и название).

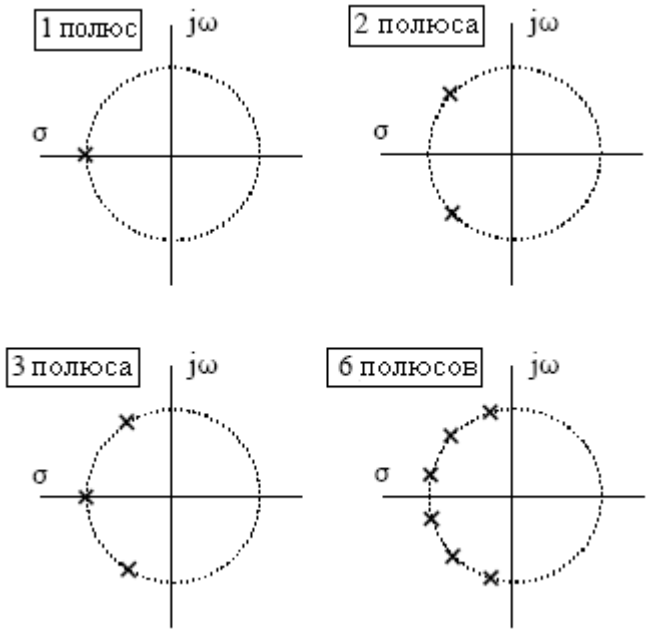


Рис. 32.10 s-плоскость Баттерворта

Поскольку каждый биквадрат дает два полюса, фильтры **четного порядка** (2 полюсные, 4 полюсные, 6 полюсные и т.д.) могут быть сконструированы при помощи последовательного соединения биквадратных ступеней. Однако для фильтров **нечетного порядка** (1 полюсные, 3 полюсные, 5 полюсные и т.д.) требуется то, чего биквадраты

просто не могут обеспечить - один полюс, находящийся на действительной оси. Для обеспечения этого, последовательно присоединяют ни что иное, как простую *R-C* цепь. Например, девяти полюсный фильтр может быть сконструирован из пяти ступеней: 4 биквадрата Саллена-Ки плюс одна ступень, состоящая из простого конденсатора и резистора.

Данные классические образцы расположения полюсов и нулей относятся к низкочастотным фильтрам, однако, они могут быть модифицированы для частотных откликов другого типа. Это осуществляется за счет проектирования низкочастотного фильтра с последующим математическим преобразованием в s-области. Мы начинаем с вычисления местоположения полюсов фильтра низкой частоты, а затем в форме уравнения (32.3), записываем передаточную функцию *H(s)*. С помощью замены каждого *"s"* на *"1/s"* находится передаточная функция соответствующего фильтра высокой частоты, а затем выражение опять приводится к виду полюсов и нулей уравнения (32.3). Это определяет местоположение новых нулей и полюсов, которые реализуют фильтр высоких частот. Более сложные преобразования s-области позволяют из исходных фильтров низкой частоты создавать полосно-пропускающие и полосно-подавляющие фильтры. Данный тип математических манипуляций в s-области является центральной темой проектирования фильтров, и этому предмету посвящены целые книги. Проектирование аналоговых фильтров это 90% *математики* и только 10% *электроники*.

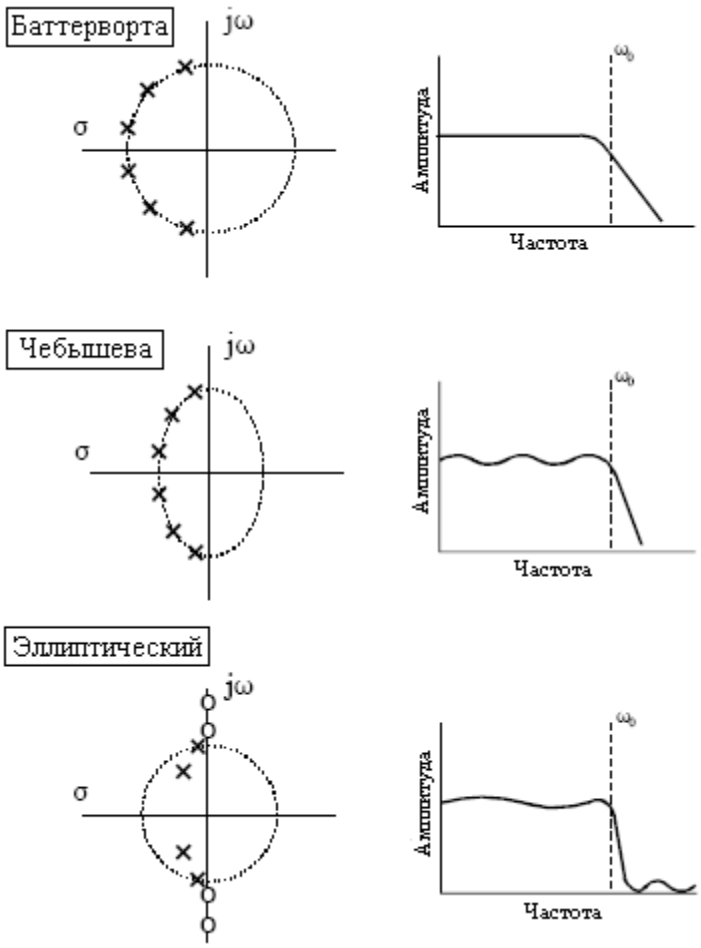


Рис. 32.11 Классические образцы полюсов и нулей

К счастью, проектирование фильтров высокой частоты, использующих ступени Саллена-Ки, не требует таких математических манипуляций. Замена *"s"* на *"1/s"* в s-области в схеме, соответствует замене резисторов на конденсаторы, а конденсаторов на

резисторы. На s-плоскости такая взаимная замена перемещает полюса в новое положение и добавляет два ноля непосредственно в начало координат. Это приводит к тому, что в области постоянной составляющей (нулевая частота) частотный отклик имеет нулевое значение, такое же, какое бы Вы ждали для фильтра верхних частот. Последнее позволяет раскрыть весь потенциал схемы Саллена-Ки, использовать два полюса и два нуля.

ГЛАВА
33

Z - ПРЕОБРАЗОВАНИЕ

Точно так же, как аналоговые фильтры проектируются с использованием преобразования Лапласа, рекурсивные цифровые фильтры разрабатываются при помощи параллельной техники, называемой z-преобразованием. Общая стратегия этих двух преобразований одинакова: зондирование импульсного отклика синусоидами и экспонентами, с целью отыскания полюсов и нулей системы. Преобразование Лапласа имеет дело с дифференциальными уравнениями, s-областью и s-плоскостью. Соответственно z-преобразование имеет дело с конечно-разностными уравнениями, z-областью и z-плоскостью. Однако, эти две техники, не являются зеркальным изображением друг друга, s-плоскость строится в прямоугольной системе координат, в то время как z-плоскость использует полярный формат. Часто проектирование рекурсивных цифровых фильтров начинается с одного из классических аналоговых фильтров, таких как Баттерворта, Чебышева или эллиптических. Затем, для получения желаемого цифрового фильтра, выполняется последовательность математических преобразований. Рамки этой математики устанавливает *z*-преобразование. Такой подход используется в представленной в Главе 20 программе для проектирования фильтра Чебышева и детально обсуждается в этой главе.

Природа z-области

Для того чтобы подчеркнуть, что преобразование Лапласа и *z*-преобразование параллельные техники, начнем с преобразования Лапласа и покажем, как оно может быть трансформировано в *z*-преобразование. Как отмечалось в предыдущей главе, преобразование Лапласа определяется соотношением, связывающим сигналы временной области и s-области:

$$X(s) = \int_{t=-\infty}^{\infty} x(t)e^{-st}dt,$$

где *x(t)* и *X(s)* - соответственно представления сигнала временной области и s-области. Как обсуждалось в предыдущей главе, это уравнение дает анализ сигнала временной области в терминах синусных и косинусных волн, имеющих экспоненциально изменяющиеся амплитуды. Это можно увидеть, заменив комплексную переменную *s* ее эквивалентным выражением $\sigma+j\omega$. Используя это альтернативное обозначение, преобразование Лапласа примет вид:

$$X(\sigma,\omega) = \int_{t=-\infty}^{\infty} x(t)e^{-\sigma t}e^{-j\omega}dt.$$

Если мы сосредоточимся на сигналах *действительной* временной области (обычный случай), то верхняя и нижняя половины *s*-плоскости будут зеркальным отображением друг друга, а член $e^{-j\omega}$ упрощается до простых косинусных и синусных волн. Такое уравнение идентифицирует каждое *местоположение* в *s*-плоскости двумя параметрами: σ и ω. *Значение* в каждом месте - это комплексное число, состоящее из действительной части и мнимой части. Для того чтобы найти действительную часть, сигнал временной области умножается на косинусоидальную волну с частотой ω и с амплитудой, которая изменяется экспоненциально в соответствии с параметром затухания σ. Тогда значение действительной части $X(\sigma,\omega)$ равно интегралу от результирующей формы волны. Значение мнимой части $X(\sigma,\omega)$ находится подобным образом за исключением использования синусоидальной волны вместо косинусоидальной. Если все это звучит для Вас не очень знакомо, то перед тем как продолжить, Вам следует еще раз просмотреть предыдущую главу.

Преобразование Лапласа может быть видоизменено в *z*-преобразование в три этапа. Первый этап наиболее очевиден: преобразовать сигналы из непрерывных в дискретные. Это делается путем замены переменной времени t на номер отсчета n и подмены интеграла суммированием:

$$X(\sigma,\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-\sigma n} e^{-j\omega n}.$$

Заметьте, что в $X(\sigma,\omega)$ используются круглые скобки, показывающие что оно является *непрерывным*, а не дискретным. Даже притом, что теперь мы имеем дело с дискретным сигналом временной области $x[n]$, параметры σ и ω могут все еще принимать непрерывный диапазон значений. На втором этапе осуществляется перезапись экспоненциальных членов. Математически экспоненциальный сигнал может быть представлен одним из следующих двух способов:

$$y[n] = e^{-\sigma n} \qquad \text{или} \qquad y[n] = r^{-n}.$$

Как показано на рис. 31.1, оба эти уравнения дают экспоненциальную кривую. Первое выражение управляет затуханием сигнала через параметр σ. Если σ положительно, то с увеличением номера отсчета n, форма волны будет *уменьшаться* по величине. Подобным же образом, если σ отрицательно, кривая будет постоянно *увеличиваться*. Если σ точно равно нулю, сигнал будет иметь постоянную величину равную *единице*. Второе выражение для управления затуханием формы волны использует параметр r. Форма волны будет уменьшаться, если $r>1$ и увеличиваться, если $r<1$. Сигнал будет иметь постоянную величину, когда $r=1$. Эти два уравнения являются просто двумя разными способами выражения одного и того же. Один способ может быть заменен другим с помощью использования соотношения:

$$r^{-n} = \left[e^{\ln(r)}\right]^{-n} = e^{-n\ln(r)} = e^{-\sigma n},$$

где $\sigma=ln(r)$.

Второй этап видоизменения преобразования Лапласа в *z*-преобразование завершается при помощи использования *другой* экспоненциальной формы:

$$X(r,\omega) = \sum_{n=-\infty}^{\infty} x[n] r^{-n} e^{-j\omega n} .$$

Хотя это идеально правильное выражение *z*-преобразования, оно не является наиболее компактной формой для комплексной записи. В преобразовании Лапласа эта проблема была преодолена введением новой комплексной переменной *s*, определенной как $s=\sigma+j\omega$. Точно таким же образом мы будем определять новую переменную для *z*-преобразования:

$$z = re^{j\omega} .$$

Это определяет комплексную переменную *z* как комбинацию двух действительных переменных *r* и ω в полярных обозначениях.

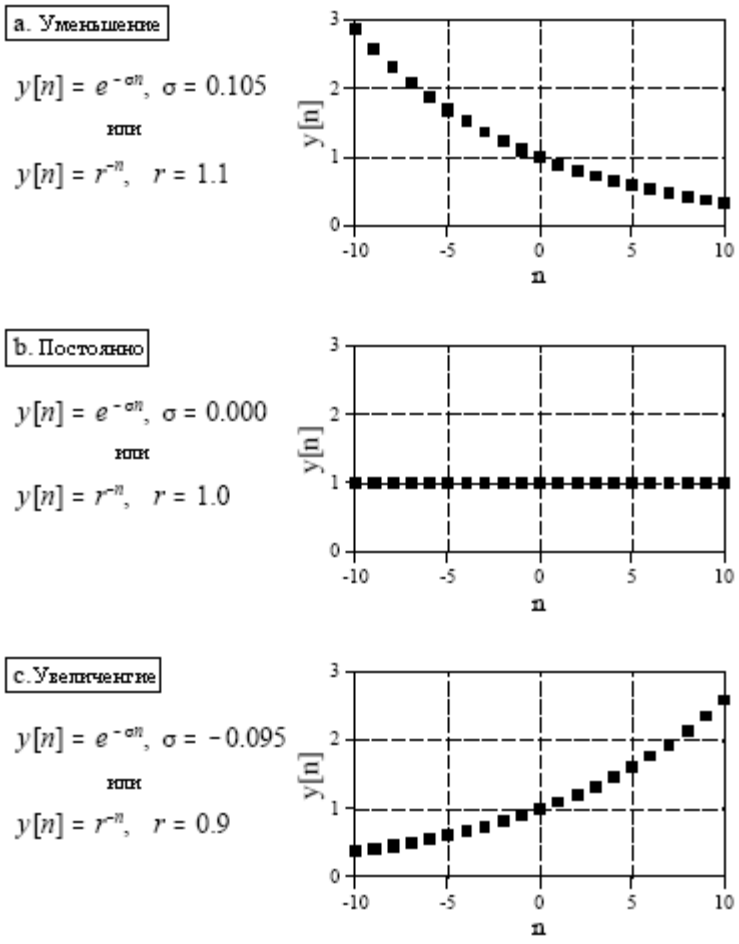


Рис. 33.1 Экспоненциальные сигналы

Третий этап в получении z-преобразования - это замена *r* и ω на *z*. Это дает стандартную форму z-преобразования:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n} . \qquad (33.1)$$

Почему в z-преобразовании вместо $e^{-\sigma n}$ используется r^n, а вместо s используется z? Как описывалось в Главе 19, рекурсивные фильтры реализуются набором *коэффициентов рекурсии*. Для анализа таких систем в z-области мы должны уметь осуществлять преобразование этих коэффициентов рекурсии в *передаточную функцию* z-области, и обратно. Как мы вскоре покажем, определение z-преобразования в такой манере (r^n и z) дает простейшее средство перехода между этими двумя важными представлениями. В действительности, определение z-области таким способом делает переход от одного представления к другому *тривиальным*.

Рисунок 33.2 иллюстрирует разницу между s-плоскостью преобразования Лапласа и z-плоскостью z-преобразования. Местоположение на s-плоскости идентифицируется двумя параметрами: переменной экспоненциального затухания σ вдоль горизонтальной оси, и переменной частоты ω вдоль вертикальной оси. Другими словами, эти два действительных параметра расположены в *прямоугольной* системе координат. Такая геометрия следует из определения s, комплексной переменной, представляющей местоположение на s-плоскости соотношением $s=\sigma+j\omega$.

Для сравнения, z-плоскость использует переменные r и ω, размещенные в *полярных* координатах. Расстояние от начала координат r является величиной экспоненциального затухания. Угловое расстояние ω, измеренное от положительной части горизонтальной оси, является частотой. Такая геометрия следует из определения z как: $z=re^{-j\omega}$. Другими словами, комплексная переменная, представляющая местоположение на z-плоскости, сформирована за счет объединения двух действительных параметров в полярной форме.

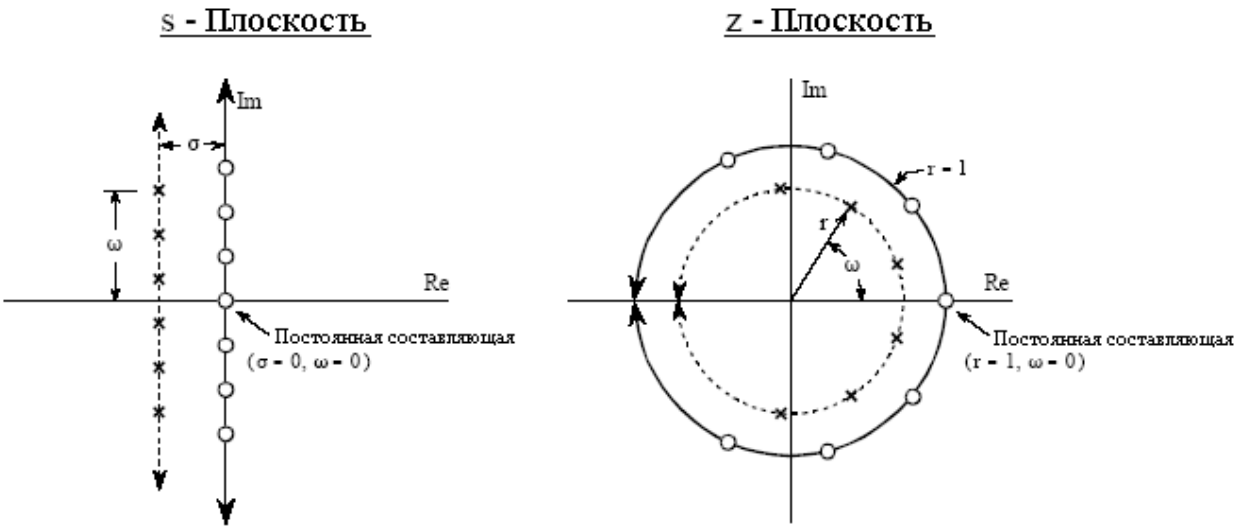


Рис. 33.2 Связь между s-плоскостью и z-плоскостью

Эти различия выражаются в том, что вертикальные *линии* на s-плоскости, на z-плоскости соответствуют *окружностям*. Например, на рис. 33.2 на s-плоскости показан образец полюсов и нулей, в котором все полюса и нули лежат на вертикальных линиях. Эквивалентные полюса и нули на z-плоскости лежат на окружностях концентрических началу координат. Это становится понятным при исследовании представленного ранее соотношения,: $\sigma=ln(r)$. Например, вертикальная ось s-плоскости (т.е. $\sigma=0$) соответствует **единичной окружности** z-плоскости (т.е. $r=1$). Вертикальные линии в левой половине s-плоскости соответствуют окружностям внутри единичной окружности z-плоскости. Подобно этому вертикальные линии в правой половине s-плоскости соответствуют окружностям снаружи единичной окружности z-плоскости. Другими словами левая и правая стороны s-плоскости отвечают внутренней и внешней сторонам единичной окружности соответственно. Например, непрерывная система неустойчива, когда полюса занимают *правую половину* s-плоскости. Точно так же дискретная система неустойчива,

когда полюса находятся *вне* единичной окружности на *z*-плоскости. Когда сигнал временной области полностью действителен (наиболее типичный случай), верхняя и нижняя половины *z*-плоскости являются зеркальным отображением друг друга, точно так же, как и в случае *s*-области.

Обратите особое внимание на то, как используется в обоих преобразованиях переменная частоты ω. *Непрерывная* синусоида может иметь любую частоту, лежащую между постоянной составляющей и бесконечностью. Это означает, что *s*-плоскость должна допускать изменение ω от минус до плюс бесконечности. Для сравнения, *дискретная* синусоида может иметь частоту только между постоянной составляющей и *1/2* частоты дискретизации. То есть частота должна находиться между *0* и *0,5*, когда она представлена в виде части частоты дискретизации (в соответствии с теоремой Котельникова – прим. перев.) или между *0* и π, когда она представлена в виде круговой частоты (т.е. $\omega=2\pi f$). Это соответствует геометрии *z*-плоскости, когда мы интерпретируем ω как угол, выраженный в *радианах*. То есть положительные частоты соответствуют углам от *0* до π радиан, в то время как отрицательные частоты соответствуют углам от *0* до $-\pi$ радиан. Поскольку *z*-плоскость отображает частоту отличающимся чем *s*-плоскость способом, некоторые авторы, для того чтобы различать эти две частоты, используют разные символы. Типичным обозначением для представления частоты в *z*-области является использование Ω (заглавная буква омега) и для частоты в *s*-области ω (прописная буква омега). В этой книге мы будем использовать ω для представления обоих видов частоты, но помните об этом при работе с другими источниками.

На *s*-плоскости, значения, которые лежат вдоль вертикальной оси, эквивалентны частотному отклику системы. То есть преобразование Лапласа, вычисленное при $\sigma=0$ эквивалентно преобразованию Фурье. Аналогичным образом частотная характеристика в z-области находится на единичной окружности. Это можно увидеть при вычислении z-преобразования (уравнение (33.1)) для *r=1*, приводящем к упрощению уравнения до дискретно-временного преобразования Фурье (ДВПФ). Частота нуля размещается здесь (постоянная составляющая) при значении *один* на горизонтальной оси в *z*-плоскости. Положительные частоты спектра размещены в отсчетах против часовой стрелки от положения постоянной составляющей, занимая верхнюю полуокружность. Подобным образом отрицательные частоты расположены от положения постоянной составляющей по часовой стрелке, формируя нижнюю полуокружность. Положительные и отрицательные частоты спектра встречаются в общих точках $\omega=\pi$ и $\omega=-\pi$. Такая круговая геометрия соответствует также *периодическим* частотным спектрам дискретных сигналов. То есть когда угол частоты, увеличиваясь, выходит за пределы π, то встречаются те же значения, что и между *0* и π. Когда же Вы обегаете вокруг круга, то Вы видите ту же картину снова и снова.

Анализ рекурсивных систем

Как отмечалось в Главе 19, рекурсивный фильтр описывается при помощи **конечно-разностного уравнения**:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + \ldots + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \ldots, \quad (33.2)$$

где $x[\ ]$ и $y[\ ]$ - входной и выходной сигналы соответственно, а члены "*a*" и "*b*" – **рекурсивные коэффициенты**. Очевидной полезностью этого уравнения является описание того, как программист может построить фильтр. Одинаково важным аспектом является то, что оно представляет собой математическое соотношение между входом и выходом, которое должно удовлетворяться постоянно. Так же, как непрерывные системы подчиняются *дифференциальным* уравнениям, рекурсивные дискретные системы функционируют в соответствии с данным *конечно-разностным* уравнением. Из этого

соотношения мы можем получить ключевые характеристики системы: импульсный отклик, переходную характеристику, частотную характеристику, чертеж нулей и полюсов и т.д.

Анализ мы начинаем с того, что берем *z*-преобразование (уравнение (33.1)) от обеих частей уравнения (33.2). Другими словами мы хотим посмотреть, как выглядит это управляющее соотношение в z-области. При изрядном количестве алгебраических преобразований мы можем представить соотношение в виде дроби *Y[z]/X[z]*, то есть в виде представления z-области выходного сигнала, деленного на представление z-области входного сигнала. Так же, как и в случае с преобразованием Лапласа, это называется **передаточной функцией системы** и обозначается *H[z]*. Вот то, что мы находим:

$$H[z] = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \ldots}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3} - \ldots}. \qquad (33.3)$$

Это один из двух способов, которыми может быть записана передаточная функция. Данная форма важна потому, что она непосредственно содержит рекурсивные коэффициенты. Например, предположим нам известны рекурсивные коэффициенты цифрового фильтра такие же, какие могут быть взяты из таблицы для проектирования фильтра:

$a_0=0,389$	
$a_1=-1,558$	$b_1=2,161$
$a_2=2,338$	$b_2=-2,033$
$a_3=-1,558$	$b_3=0,878$
$a_4=0,389$	$b_4=-0,161$

Не беспокоясь об утомительной комплексной алгебре, мы можем непосредственно записать передаточную функцию системы:

$$H[z] = \frac{0,389 - 1,558 z^{-1} + 2,338 z^{-2} - 1,558 z^{-3} + 0,389 z^{-4}}{1 - 2,161 z^{-1} + 2,033 z^{-2} - 0,878 z^{-3} + 0,161 z^{-4}}.$$

Обратите внимание на то, что коэффициенты “*b*” входят в передаточную функцию с отрицательным знаком перед ними. Альтернативно, некоторые авторы записывают это уравнение, используя только знак суммы, но изменяя знак всех коэффициентов “*b*”. А вот в этом проблема. Если Вам дан набор рекурсивных коэффициентов (в виде таблицы или программы проектирования фильтра), то шансов, что коэффициенты “*b*” будут иметь противоположный знак от того, что Вы ожидаете, будет 50 на 50. Если Вы не выявите этого несоответствия, фильтр будет чрезвычайно неустойчивым.

В уравнении (33.3) передаточная функция выражается с использованием отрицательных степеней *z* таких как: z^{-1}, z^{-2}, z^{-3} и т.д. После того, как подставлен набор фактических рекурсивных коэффициентов, мы можем преобразовать передаточную функцию в более обычную форму, которая использует положительные степени, то есть z^1 z^2, z^3,…. Умножением обоих числителя и знаменателя для нашего примера на z^4 получим:

$$H[z] = \frac{0,389 z^4 - 1,558 z^3 + 2,338 z^2 - 1,558 z + 0,389}{z^4 - 2,161 z^3 + 2,033 z^2 - 0,878 z + 0,161}$$

Зачастую использовать положительные степени значительно легче, и для некоторых методов z-области они просто *необходимы*. Почему бы сразу не записывать уравнение (33.3) с использованием положительных степеней и вообще забыть об отрицательных степенях? Нельзя! Трюк с умножением числителя и знаменателя на z в самой высокой степени (такой как z^4 в нашем примере) может использоваться только тогда, когда число рекурсивных коэффициентов уже известно. Уравнение же (33.3) записано для *произвольного* числа коэффициентов. Все дело в том, что в ЦОС обычно используются как положительные, так и отрицательные степени, и Вам следует знать, как осуществляется переход из одной формы в другую.

Передаточная функция рекурсивной системы полезна потому, что ей можно манипулировать способами, которые рекурсивные коэффициенты не допускают использовать. Сюда относятся такие задачи как: объединение последовательных и параллельных каскадов в единую систему, проектирование фильтров посредством задания местоположения полюсов и нулей, преобразование аналоговых фильтров в цифровые и т.д. Такие действия выполняются при помощи алгебры представленной в z-области умножением, сложением и разложением на множители. После того как все эти действия завершены, новая передаточная функция записывается в форме уравнения (33.3), позволяя идентифицировать новые коэффициенты рекурсии.

Так же, как и в случае s-области важной характеристикой z-области является то, что передаточная функция может быть выражена посредством **полюсов** и **нулей**. Это дает вторую основную форму представления z-области:

$$H[z] = \frac{(z - z_1)(z - z_2)(z - z_3)\ldots}{(z - p_1)(z - p_2)(z - p_3)\ldots}. \qquad (33.4)$$

Каждый из полюсов (p_1, p_2, p_3 …) и нулей (z_1, z_2, z_3 …) является комплексным числом. Для перехода от (33.4) к (33.3) осуществляют перемножение скобок и объединение подобных членов полученного выражения. Хотя это может потребовать большого количества алгебраических преобразований, в принципе они достаточно прямые, и легко могут быть вписаны в компьютерную программу. Переход от уравнения (33.3) к уравнению (33.4) значительно более трудный, поскольку он требует *разложения полиномов* на множители. Как обсуждалось в Главе 32, если передаточная функция второго порядка или менее (т.е. нет степеней z выше, чем z^2) для разложения полиномов на множители может быть использовано квадратное уравнение. При степени системы большей, чем вторая, алгебраические методы, в общем, не могут быть использованы, и должны применяться численные методы. К счастью это требуется достаточно редко, проектирование цифровых фильтров *начинается* с определения местоположения полюсов и нулей (уравнение 33.4) и *заканчивается* рекурсивными коэффициентами (уравнение33.3), а не наоборот.

Как и для всех комплексных чисел, для полюсов и нулей местоположения могут быть представлены в полярной или прямоугольной форме. Полярная система обозначений имеет преимущество, будучи более совместимой, с естественной организацией z-плоскости. Для сравнения, прямоугольная форма в основном предпочитается для математической работы, то есть обычно проще манипулировать $\sigma+j\omega$, по сравнению с $re^{j\omega t}$.

В качестве примера использования этих уравнений спроектируем фильтр - “пробку” посредством следующих этапов: (1) определим местоположение полюсов и нулей на z-плоскости, (2) запишем передаточную функцию в форме уравнения (33.4), (3) преобразуем передаточную функцию в форму уравнения (33.3) и (4) найдем рекурсивные коэффициенты, требуемые для реализации фильтра. На рис. 33.3 показан пример, который мы будем использовать: фильтр-пробка сформирован двумя полюсами и двумя нулями, расположенными в местоположениях

в полярной форме	в прямоугольной форме
$z_1=1{,}00e^{j(\pi/4)}$	$z_1=0{,}7071+j0{,}7071$
$z_2=1{,}00e^{j(-\pi/4)}$	$z_2=0{,}7071-j0{,}7071$
$p_1=0{,}90e^{j(\pi/4)}$	$p_1=0{,}6364+j0{,}6364$
$p_2=0{,}90e^{j(-\pi/4)}$	$p_2=0{,}6364-j0{,}6364.$

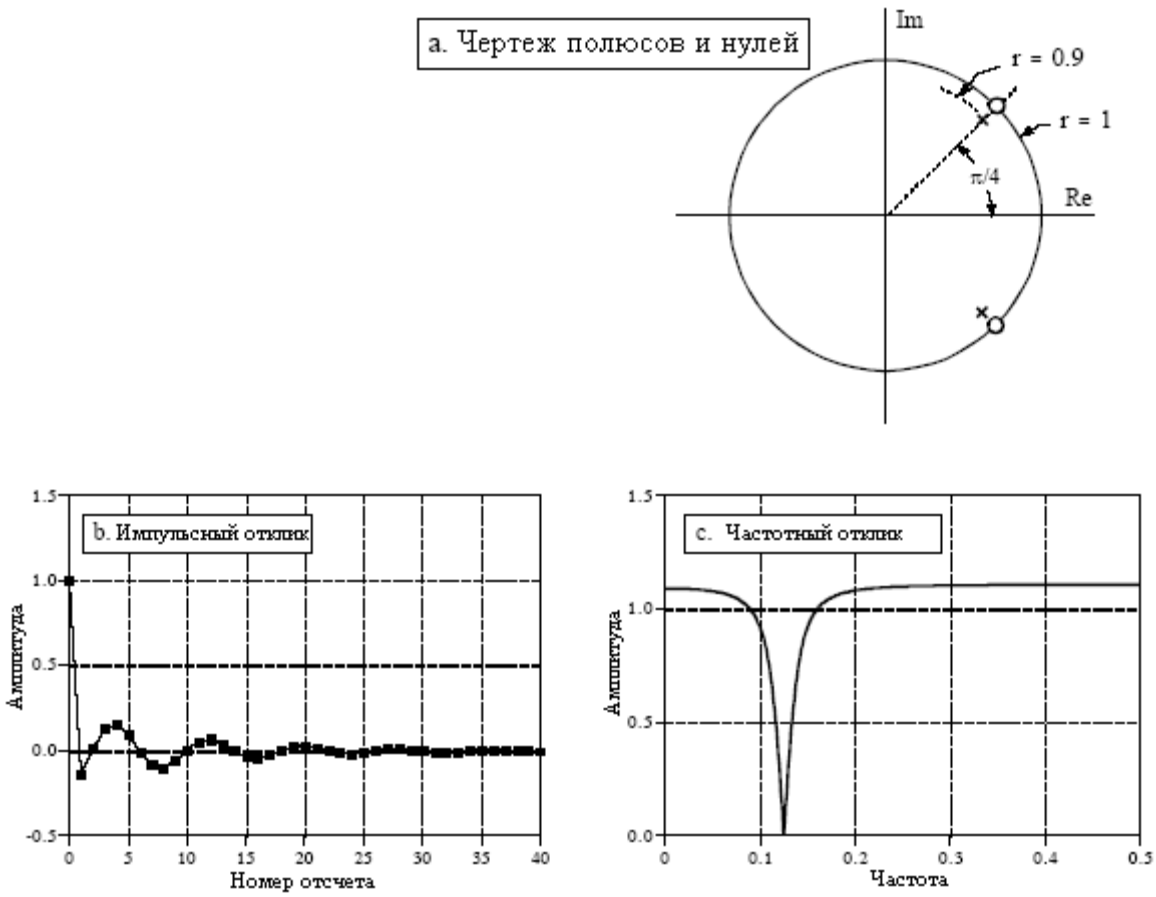


Рис. 33.3 Фильтр-пробка, спроектированный в z-области

Для того чтобы понять, что это действительно фильтр - “пробка”, сравним данный чертеж полюсов и нулей с рис. 32.7 - фильтром “пробкой” на *s*-плоскости. Единственное различие заключается в том, что для того чтобы из z-плоскости найти частотную характеристику, мы перемещаемся по *единичной окружности*, в противоположность перемещению по *вертикальной оси*, для того чтобы найти частотную характеристику из s-плоскости. Из полярной формы представления полюсов и нулей можно увидеть, что на циклической частоте равной $\pi/4$, соответствующей *0,125* от частоты дискретизации, находится центр впадины частотной характеристики.

Так как местоположение полюсов и нулей известно, простой подстановкой значений можно записать передаточную функцию в виде уравнения (33.4):

$$H[z]=\frac{[z-(0{,}7071+j0{,}7071)][z-(0{,}7071-j0{,}7071)]}{[z-(0{,}6364+j0{,}6364)][z-(0{,}6364-j0{,}6364)]}.$$

Для того чтобы найти рекурсивные коэффициенты, реализующие данный фильтр, передаточная функция должна быть приведена к виду уравнения (33.3). Для начала расширим выражение, перемножив его составляющие члены:

$$H[z]=\frac{z^2-0{,}7071z+j0{,}7071z-0{,}7071z+0{,}7071^2-j0{,}7071^2-j0{,}7071z+j0{,}7071^2-j^2\,0{,}7071^2}{z^2-0{,}6364z+j0{,}6364z-0{,}6364z+0{,}6364^2-j0{,}6364^2-j0{,}6364z+j0{,}6364^2-j^2\,0{,}6364^2}$$

Затем мы находим и сокращаем подобные члены. До тех пор пока выполняется условие, что верхняя половина *z*-плоскости является зеркальным отображением нижней половины (что имеет место всегда, если мы работаем с *действительным* импульсным откликом), все члены, содержащие "*j*" в выражении будут сокращаться:

$$H[z]=\frac{1{,}000-1{,}414z+1{,}000z^2}{0{,}810-1{,}273z+1{,}000z^2}.$$

Хотя это выражение представлено в виде одного полинома деленного на другой, здесь не используются отрицательные показатели степени при *z*, как этого требует уравнение (33.3). Последнее можно изменить делением обоих и числителя, и знаменателя на *z* в наиболее высокой в этом выражении степени, в данном случае на z^2:

$$H[z]=\frac{1{,}000z^{-2}-1{,}414z^{-1}+1{,}000}{0{,}810z^{-2}-1{,}273z^{-1}+1{,}000}.$$

Поскольку теперь передаточная функция имеет вид уравнения (33.3), рекурсивные коэффициенты могут быть непосредственно извлечены из ее рассмотрения:

$a_0=1{,}000$
$a_1=-1{,}414$ $\qquad$ $b_1=1{,}273$
$a_2=1{,}000$ $\qquad$ $b_2=-0{,}810$

Этот пример показывает общую стратегию получения рекурсивных коэффициентов из диаграммы полюсов и нулей. В особых случаях возможен непосредственный вывод простых уравнений, непосредственно связывающих положения полюсов и нулей с рекурсивными коэффициентами. Например, система, содержащая два полюса и два нуля, называемая **биквадратной,** имеет следующие связывающие соотношения:

$$\begin{aligned} a_0&=1\\ a_1&=-2r_0\cos(\omega_0)\\ a_2&=r_0^2\\ \\ b_1&=2r_p\cos(\omega_p)\\ b_2&=-r_p^2 \end{aligned} \qquad (33.5)$$

А как же мы найдем частотную характеристику, после того как передаточная функция определена? Существуют три метода: один математический и два численных (программирование). Математический метод основан на поиске значений на z-плоскости лежащих на единичной окружности. Это выполняется вычислением значений передаточной функции *H(z)* для *r=1*. А именно, мы начинаем с записи передаточной

функции в виде уравнения (33.3) или (33.4). Затем заменяем каждое z на $e^{-j\omega}$ (т.е. $re^{-j\omega}$ при $r=1$). Это дает математическое уравнение частотной характеристики $H(\omega)$. Трудность заключается в том, что результирующее выражение находится в очень неудобной форме. Для получения чего-то узнаваемого как, например, амплитуды и фазы обычно требуется большое число алгебраических преобразований. Хотя этот метод дает точное уравнение для частотного отклика, его трудно автоматизировать компьютерными программами, подобными требуемым в пакетах по проектированию фильтров.

Второй метод по нахождению частотного отклика также использует подход оценки z-плоскости на единичной окружности. Разница состоит в том, что мы находим не математическое решение для всей кривой, а только *отсчеты* частотной характеристики. Компьютерная программа выполняет циклы для может быть 1000 равно отделенных частот, лежащих между $\omega=0$ и $\omega=\pi$. Подумайте о муравье, движущемся между 1000 дискретных точек по верхней половине единичной окружности z-плоскости. В каждом из этих мест при помощи вычисления передаточной функции находятся амплитуда и фаза частотной характеристики.

Этот метод хорошо работает и часто используется в пакетах по проектированию фильтров. Его главным ограничением является то, что он не учитывает *шум округления*, воздействующий на характеристики системы. Даже если частотный отклик, найденный этим методом, выглядит идеально, реализованная система может оказаться полностью неустойчивой!

Это приводит к третьему методу: нахождению частотной характеристики из рекурсивных коэффициентов, которые фактически используются для реализации фильтра. Для начала, пропуская импульс через систему, мы находим импульсный отклик фильтра. Вторым шагом для нахождения частотного отклика системы, мы берем ДПФ от импульсного отклика (конечно же, используя БПФ). Единственным критическим моментом, о котором следует помнить в этой процедуре, является то, что необходимо взять достаточно большое количество отсчетов импульсной характеристики так, чтобы отброшенные отсчеты *не оказывали влияния* на результат. Хотя по теоретическим критериям этого могут быть написаны книги, практические правила гораздо проще. Используйте столько отсчетов, сколько Вы *думаете* необходимо. После того как частотная характеристика найдена, вернитесь и повторите процедуру, используя в два раза больше отсчетов. Если два частотных отклика адекватно похожи, Вы можете быть уверены, что усечение импульсного отклика не обмануло Вас никоим образом.

Последовательное и параллельное соединения

Для упрощения утомительной алгебры z-области сложные рекурсивные фильтры обычно проектируются по каскадам. Рисунок 33.4 иллюстрирует два обычных способа, которыми могут быть соединены отдельные каскады: последовательное соединение и параллельное соединение с суммированием выходных сигналов. Например, для формирования *полосно-пропускающего* фильтра, могут быть соединены последовательно низкочастотный и высокочастотный каскады. Подобно этому параллельное соединение низкочастотного и высокочастотного каскадов может сформировать *полосно-заграждающий* фильтр. Будем называть два объединяемых каскада с их рекурсивными коэффициентами, именуемыми a_0, a_1, a_2, b_1, b_2 и A_0, A_1, A_2, B_1, B_2, соответственно *системой 1* и *системой 2*. Нашей целью является объединение этих каскадов (последовательно или параллельно) в простой рекурсивный фильтр с рекурсивными коэффициентами, задаваемыми a_0, a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4, который мы будем называть *системой 3*.

Как Вы помните из предыдущих глав, частотные отклики систем соединенных последовательно объединяются с помощью *умножения*. А также, частотные отклики систем, соединенных параллельно, объединяются с помощью *сложения*. Те же самые

правила употребляются и по отношению к передаточным функциям z-области. Это позволяет объединить рекурсивные системы, перенеся задачу в z-область, выполнить там необходимые умножения и сложения и затем вернуться к рекурсивным коэффициентам окончательной системы.

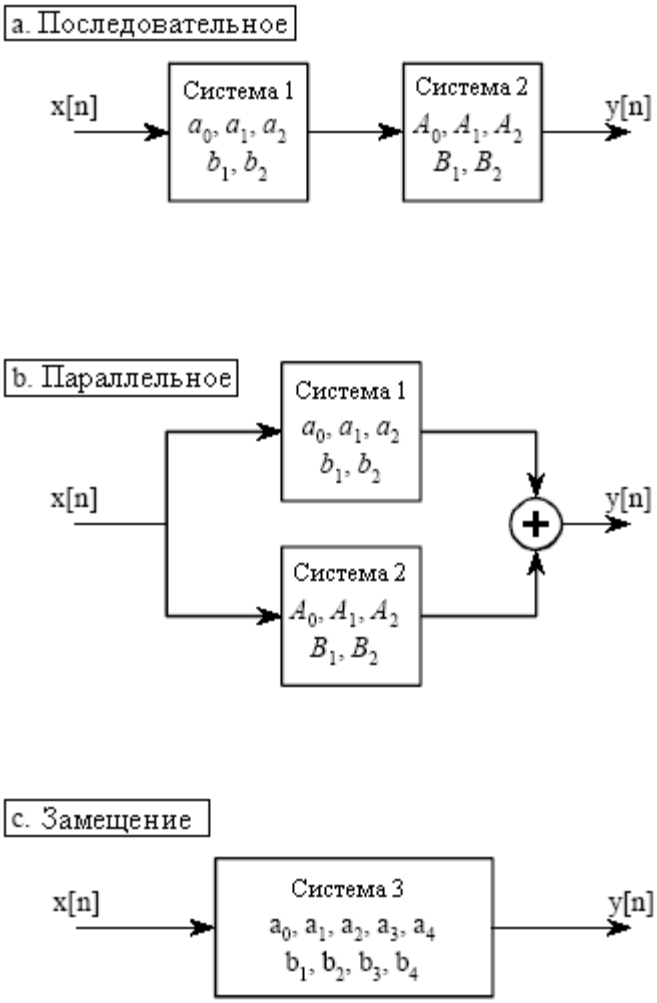


Рис. 33.4 Последовательное и параллельное соединение каскадов

В качестве примера этого способа выполним алгебраические преобразования по соединению двух биквадратных каскадов последовательно. Передаточная функция каждого из каскадов находится с помощью записи уравнения (33.3) с использованием соответствующих рекурсивных коэффициентов. Затем при помощи перемножения передаточных функций двух каскадов находится передаточная функция $H[z]$ всей системы:

$$H[z] = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - b_1 z^{-1} - b_2 z^{-2}} \times \frac{A_0 + A_1 z^{-1} + A_2 z^{-2}}{1 - B_1 z^{-1} - B_2 z^{-2}}.$$

Перемножим полиномы и объединим подобные члены:

$$H[z] = \frac{a_0 A_0 + (a_0 A_1 + a_1 A_0) z^{-1} + (a_0 A_2 + a_1 A_1 + a_2 A_0) z^{-2} + (a_1 A_2 + a_2 A_1) z^{-3} + (a_2 A_2) z^{-4}}{1 - (b_1 + B_1) z^{-1} - (b_2 + B_2 - b_1 B_1) z^{-2} - (-b_1 B_2 - b_2 B_1) z^{-3} - (-b_2 B_2) z^{-4}}$$

Поскольку данное выражение записано в виде уравнения 33.3, мы можем непосредственно извлечь рекурсивные коэффициенты, реализующие систему последовательно соединенных каскадов.

$$a_0=a_0A_0$$
$$a_1=a_0A_1+a_1A_0 \qquad b_1=b_1+B_1$$
$$a_2=a_0A_2+a_1A_1+a_2A_0 \qquad b_2=b_2+B_2-b_1B_2$$
$$a_3=a_1A_2+a_2A_1 \qquad b_3=-b_1B_2-b_2B_1$$
$$a_4=a_2A_2 \qquad b_4=-b_2B_2$$

Очевидным недостатком этого метода является большое количество алгебраических преобразований, требуемых для выполнения умножения и перегруппировки полиномиальных членов. К счастью, весь алгоритм может быть выражен в виде короткой компьютерной программы, показанной в таблице 33.1. Хотя последовательное и параллельное соединения требуют разной математики, они используют почти одну и туже программу. В частности, разница между двумя алгоритмами состоит только в одной строке программного кода, что позволяет объединить оба алгоритма в одной простой программе.

Таблица 33.1

```
100 'COMBINING RECURSION COEFFICIENTS OF CASCADE AND PARALLEL
STAGES
110 '
120 '                                   'INITIALIZE VARIABLES
130 DIM A1[8], B1[8]                    'a and b coefficients for system 1, one of the stages
140 DIM A2[8], B2[8]                    'A and B coefficients for system 2, one of the stages
150 DIM A3[16], B3[16]                  'a and b coefficients for system 3, the combined system
160 '
170                                     'Indicate cascade or parallel combination
180 INPUT "Enter 0 for cascade, 1 for parallel: ", CP%
190 '
200 GOSUB XXXX                          'Mythical subroutine to load: A1[ ], B1[ ], A2[ ], B2[ ]
210 '
220 FOR I% = 0 TO 8                     'Convert the recursion coefficients into transfer functions
230 B2[I%] = -B2[I%]
240 B1[I%] = -B1[I%]
250 NEXT I%
260 B1[0] = 1
270 B2[0] = 1
280 '
290 FOR I% = 0 TO 16                    'Multiply the polynomials by convolving
300 A3[I%] = 0
310 B3[I%] = 0
320 FOR J% = 0 TO 8
330 IF I%-J% < 0 OR I%-J% > 8 THEN GOTO 370
340 IF CP% = 0 THEN A3[I%] = A3[I%] + A1[J%] * A2[I%-J%]
350 IF CP% = 1 THEN A3[I%] = A3[I%] + A1[J%] * B2[I%-J%] + A2[J%] * B1[I%-J%]
360 B3[I%] = B3[I%] + B1[J%] * B2[I%-J%]
370 NEXT J%
380 NEXT I%
```

```
390 '
400 FOR I% = 0 TO 16                'Convert the transfer function into recursion coefficients.
410 B3[I%] = -B3[I%]
420 NEXT I%
430 B3[0] = 0
440 '                               'The recursion coefficients of the combined system now
450 END                             'reside in A3[ ] & B3[ ]
```

Эта программа работает методом преобразования рекурсивных коэффициентов каждого отдельного каскада в передаточные функции вида уравнения (33.3) (строки 220-270). После объединения соответствующим образом этих передаточных функций (строки 290-380), будучи рекурсивными коэффициентами, информация возвращается назад (строки 400-430).

Сердцем этой программы является то, как представляются и объединяются полиномы передаточной функции. Например, числителем передаточной функции первого каскада является: $a_0+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}+\ldots$. Этот полином представлен в программе с помощью хранимых в векторе A1[0], A1[1], A1[2], A1[3], … коэффициентов a_0, a_1, a_2, a_3, …. Подобным же образом, числитель передаточной функции второго каскада представлен значениями, хранимыми в A2[0], A2[1], A2[2], A2[3], …, а числитель объединенной системы в A3[0], A3[1], A3[2], A3[3], …. Идея состоит в том, чтобы представлять и манипулировать *полиномами* только посредством их *коэффициентов*. Вопрос заключается в том, как нам вычислить A3[ ] с учетом того, что все и A1[ ], и A2[ ], и A3[ ] представляют собой полиномы? Ответ заключается в том, что когда перемножаются два полинома, с их коэффициентами происходит операция *свертки*. В виде уравнения это выглядит как: A1[ ]*A2[ ]= A3[ ]. Это позволяет для нахождения передаточной функции последовательно соединенных каскадов использовать стандартный алгоритм свертки, осуществляя свертку двух векторов числителей и двух векторов знаменателей.

Процедура объединения передаточных функций каскадов, соединенных параллельно, является чуть-чуть более сложной. В алгебраической форме, в соответствии с

$$\frac{w}{x}+\frac{y}{z}=\frac{wz+xy}{xz}$$

появляются дроби.

Поскольку каждая из передаточных функций является дробью (один полином делится на другой), мы объединяем параллельные каскады с помощью перемножения знаменателей и суммирования взятых крест на крест произведений. Это означает, что знаменатель вычисляется так же, как и для последовательно соединенных каскадов, а вот числитель вычисляется более сложно. В строке 340 для нахождения числителя объединенной передаточной функции с числителями *последовательно* соединенных каскадов выполняется операция свертки. В строке 350, как сумма сверток двух числителей с двумя знаменателями, вычисляется числитель *параллельно* соединенных каскадов. Вычисление знаменателя для обоих случаев выполняется в строке 360.

Инверсия спектра

В Главе 14 описывается техника КИХ-фильтра, называемая *инверсией спектра*. Это такой способ изменения ядра фильтра, когда частотная характеристика переворачивается сверху вниз. Все полосы пропускания заменяются полосами заграждения и наоборот. Например, низкочастотный фильтр превращается в высокочастотный, полосно-

пропускающий фильтр превращается в полосно-заграждающий и т.д. Подобная процедура может быть проделана и с рекурсивными фильтрами, хотя она значительно менее успешна.

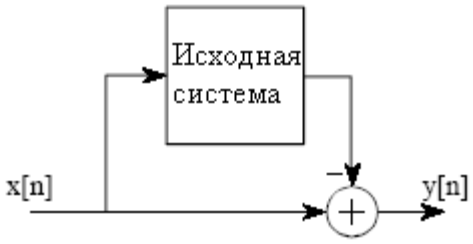


Рис. 33.5 Инверсия спектра

Как показано на рис. 33.5, инверсия спектра осуществляется посредством вычитания выходного сигнала системы из исходного сигнала. Эта процедура может рассматриваться как объединение в параллель двух каскадов, где один из каскадов оказывается *повторитель* (выходной сигнал полностью идентичен входному). Используя данный подход, можно показать, что коэффициенты “b” остаются без изменения, а модифицированные коэффициенты “a” определяются по:

$$\begin{aligned} a_0 &= 1 - a_0 \\ a_1 &= -a_1 - b_1 \\ a_2 &= -a_2 - b_2 \\ a_3 &= -a_3 - b_3 \\ &\dots \end{aligned} \qquad (33.6)$$

На рис. 33.6 показана инверсия спектра двух типичных частотных характеристик: низкочастотного фильтра (a) и фильтра “пробки” (c). Они дают соответственно высокочастотный фильтр (b) и полосно-пропускающий фильтр (d). Как же выглядят результирующие частотные характеристики? У высокочастотного фильтра они абсолютно ужасны! Хотя у полосно-пропускающего фильтра они лучше, пик все же не такой острый, как у фильтра “пробки”, из которого он был получен. Такие посредственные результаты по сравнению с превосходным исполнением, которое мы наблюдали в Главе 14, особенно разочаровывают. Откуда же такая разница? Ответ заключается в том, о чем в проектировании фильтров часто забывают в *фазовой характеристике*.

Для иллюстрации того, что виновной становится фаза, рассмотрим систему называемую **преобразователем Гильберта** (**квадратурным фильтром** - прим. перев). Преобразователь Гильберта это не специфический прибор, но некоторая система со следующим частотным откликом: амплитуда = 1, а фаза = 90 градусов для всего диапазона частот. Это означает, что амплитуда любой синусоиды, прошедшей через преобразователь Гильберта, останется без изменений, а фаза изменится на одну четвертую периода. Преобразователи Гильберта могут быть аналоговыми или дискретными (то есть реализованные с помощью аппаратных средств или программного обеспечения) и обычно используются в связи для реализации различных способов модуляции и демодуляции.

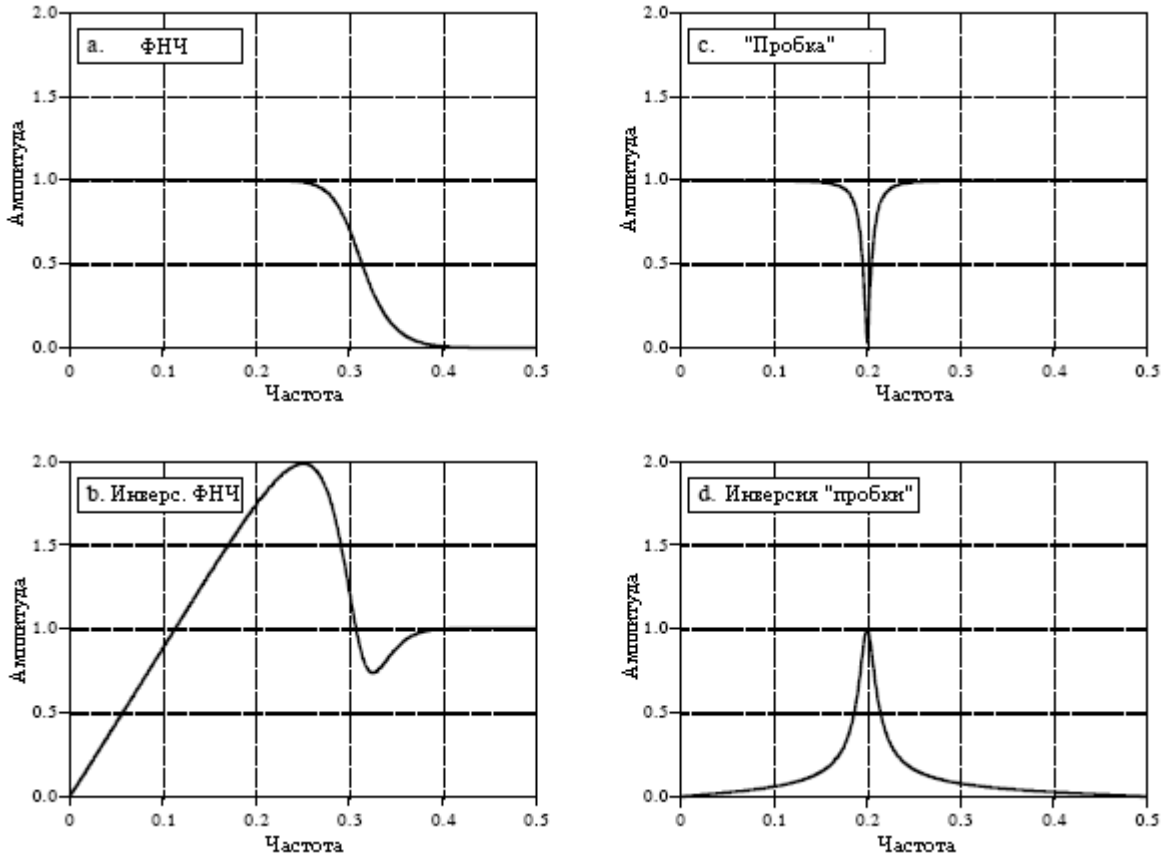


Рис. 33.6 Примеры инверсии спектра

Теперь предположим, что мы осуществляем инверсию спектра преобразователя Гильберта, вычитая его выходной сигнал из исходного сигнала. Рассматривая только *амплитуду* частотной характеристики, мы можем прийти к заключению, что вся система будет иметь выходной сигнал равный *нулю*. То есть поскольку амплитуда выходного сигнала преобразователя Гильберта идентична амплитуде исходного сигнала, они будут взаимно компенсироваться. Конечно же, это совершенно не верно. Две синусоиды будут полностью взаимно компенсироваться, только если у них одна и та же амплитуда *и* фаза. В действительности амплитуда частотного отклика этой сложной системы равна $\sqrt{2}$, а фазовый сдвиг -45^0. То есть, вместо того чтобы быть равной нулю (наше наивное предположение), амплитуда выходного сигнала имеет *большую* величину, чем амплитуда входного сигнала!

Инверсия спектра хорошо работает в Главе 14 потому, что там используется специфический вид фильтра - фильтр с *нулевой фазой*. То есть ядро фильтра симметрично слева направо. Когда система не вносит никакого фазового сдвига, результат вычитания выходного сигнала из входного диктуется исключительно амплитудами. Поскольку рекурсивные фильтры подвержены сдвигу фаз, инверсия спектра обычно приводит к неудовлетворительным фильтрам.

Изменение усиления

Предположим, что у нас есть рекурсивный фильтр и нам нужно изменить рекурсивные коэффициенты таким образом, чтобы выходной сигнал изменился по амплитуде. Это, например, может быть, понадобится для подстраховки того, что у фильтра в полосе пропускания будет единичный коэффициент усиления. Способ достижения этого очень прост: умножить коэффициенты “a” на некоторый множитель,

величина которого равна тому, во сколько раз мы хотим изменить усиление, а коэффициенты “b” оставить нетронутыми.

Перед регулировкой усиления мы, вероятно, хотели бы знать его текущее значение. Поскольку величина усиления должна быть определена на частотах в *полосе пропускания* фильтра, эта процедура зависит от типа используемого фильтра. Величину усиления фильтров низких частот измеряют на *нулевой* частоте, в то время как величину усиления фильтров высоких частот измеряют на частоте равной *0,5* от максимально допустимой частоты фильтра. Вывести выражение для определения величины усиления на этих двух специальных частотах достаточно просто. Вот как это делается.

Прежде всего, выведем уравнение для определения величины усиления на нулевой частоте. Идея состоит в том, чтобы принудительно привести каждый из входных отсчетов к значению *единицы*, дабы в результате каждый из выходных отсчетов имел значение *G* - усиления системы, которое мы пытаемся найти. Мы начнем с записи рекурсивного уравнения, математического соотношения между входным и выходным сигналами:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + ... + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + ...$$

Затем, мы подставим *единицу* вместо каждого входного отсчета, а вместо каждого выходного отсчета - *G*. Другими словами мы вынудим систему работать на нулевой частоте. Тогда уравнение примет вид:

$$G = a_0 + a_1 + a_2 + a_3 + ... + b_1 G + b_2 G + b_3 G + ...$$

Записывая его относительно *G*, получим величину усиления системы на нулевой частоте в зависимости от рекурсивных коэффициентов:

$$G = \frac{a_0 + a_1 + a_2 + a_3 + ...}{1 - (b_1 + b_2 + b_3 + ...)}. \qquad (33.7)$$

Для того чтобы сделать фильтр с коэффициентом усиления равным единице на нулевой частоте, используя данное соотношение, вычислите существующее усиление, а затем разделите все коэффициенты “*a*” на *G*.

Подобным же образом находится усиление и на частоте 0,5: данную частоту принудительно делают рабочей для входного и выходного сигналов и смотрят, каков отклик системы. На частоте равной 0,5 отсчеты входного сигнала чередуются между *–1* и *1*. То есть последовательность отсчетов следующая: *1*, *-1*, *1*, *-1*, *1*, *-1*, *1* и т.д. Соответствующий выходной сигнал с амплитудой равной величине усиления системы также чередуется по знаку: *G*, *-G*, *G*, *-G*, *G*, *-G* и т.д. Подставляя эти сигналы в рекурсивное уравнение, получим:

$$G = a_0 - a_1 + a_2 - a_3 + ... - b_1 G + b_2 G - b_3 G + ...$$

Записывая это уравнение относительно *G*, получим величину усиления системы на частоте *0,5* в зависимости от рекурсивных коэффициентов:

$$G = \frac{a_0 - a_1 + a_2 - a_3 + ...}{1 - (-b_1 + b_2 - b_3 + ...)}. \qquad (33.8)$$

Так же как и раньше, делением всех коэффициентов "*a*" на данную вычисленную величину *G*, фильтр может быть нормализован к единичному усилению. Вычисление уравнения (33.8) с помощью компьютерной программы требует некоторого способа генерирования отрицательных знаков для нечетных коэффициентов, и положительных знаков для четных коэффициентов. Наиболее обычным способом является умножение каждого коэффициента на $(-1)^k$, где k – рабочий индекс коэффициента. То есть по мере того как k пробегает значения *0*, *1*, *2*, *3*, *4*, *5*, *6* и т.д., выражение $(-1)^k$ принимает значения *1*, *-1*, *1*, *-1*, *1*, *-1*, *1* и т.д.

Проектирование фильтров Чебышева - Баттерворта

Типичный способ проектирования рекурсивных цифровых фильтров показан на примере программы, представленной в Главе 20. Проектирование начинается с чертежа на *s*-плоскости полюсов и нулей *аналогового* фильтра и преобразования его в желаемый *цифровой* фильтр с помощью нескольких математических *преобразований*. Для снижения сложности алгебраических преобразований фильтр проектируется в виде нескольких каскадов, соединенных последовательно, причем, каждый каскад фильтра реализует одну пару полюсов. Затем, рекурсивные коэффициенты для каждого каскада объединяются в рекурсивные коэффициенты всего фильтра. Это очень изворотливый и сложный алгоритм, как раз самый подходящий способ закончить эту книгу. Вот как он работает!

Управление циклом

На рис. 33.7 показана программа и блок схема алгоритма для способа, скопированного из Главы 20. Основная часть программы после инициализации и ввода параметров представляет собой цикл, пробегающий по всем парам полюсов фильтра. В программе цикл реализуется петлей FOR-NEXT, находящейся в строках с 320 по 460, количество проходов цикла контролируется в блок-схеме алгоритма блоком 11. Например, для *6* полюсного фильтра цикл будет выполнен три раза, при этом индекс цикла P% по мере выполнения цикла, будет принимать значения *1*, *2*, *3*. То есть шести полюсный фильтр реализуется тремя каскадами, по два полюса на каскад.

Объединение коэффициентов

Во время выполнения каждого цикла подпрограмма *1000*, приведенная на рис. 33.8, вычисляет рекурсивные коэффициенты *для всех каскадов*. Последние возвращаются из программы в виде пяти переменных: A0, A1, A2, B1, B2. В блоке 10 блок-схемы алгоритма (строки 360-440 программы) эти коэффициенты объединяются с коэффициентами предыдущих каскадов, содержащихся в векторах A[ ] и B[ ]. В конце первого цикла A[ ] и B[ ] содержат коэффициенты для первого каскада фильтра. В конце второго цикла A[ ] и B[ ] содержат коэффициенты первого и второго каскадов. Когда все циклы будут завершены, A[ ] и B[ ] содержат коэффициенты необходимые для реализации всего фильтра.

Коэффициенты объединяются так, как это было показано ранее в таблице 33.1, за исключением небольших модификаций, призванных сделать код программы более компактным. Во-первых, во время выполнения цикла, был смещен на *два* индекс векторов A[ ] и B[ ]. Например, a_0 располагается в A[2], a_1 и b_1 располагаются в A[3] и B[3] и т.д. Это было сделано для предотвращения доступа программы к значениям, лежащим за пределами заданных векторов. Данный сдвиг устраняется в блоке 12 блок-схемы алгоритма (строки 480-520 программы), так что окончательные рекурсивные коэффициенты размещаются в A[ ] и B[ ] без всякого смещения индекса.

Во-вторых, вектора A[ ] и B[ ] должны быть приведены к исходному состоянию с коэффициентами, не все из которых равны нулю, соответствующими *идентичной* системе. Это выполняется в строках со 180 по 240. Во время первого цикла рекурсивные коэффициенты первого каскада объединяются с информацией первоначально представленной в этих векторах. Если бы с самого начала здесь присутствовали только

нули, вектора бы всегда оставались нулевыми. В-третьих, используются два временных вектора TA[ ] и TB[ ]. Они содержат старые значения векторов A[ ] и B[ ] во время выполнения операции свертки, освобождая A[ ] и B[ ] для новых значений.

Чтобы закончить программу, блок 13 (строки 540-670) приводит фильтр к единичному усилению в полосе пропускания. Это выполняется так, как было описано ранее: с помощью уравнения (33.7) или (33.8) вычисляется существующее усиление, и для осуществления нормализации на него делятся все коэффициенты "*a*". Промежуточные переменные SA и SB являются соответственно суммами коэффициентов "a" и "b".

Вычисление местоположения полюсов на *s*-плоскости

Независимо от типа разрабатываемого фильтра, эта программа начинается с фильтра Баттерворта нижних частот с частотой среза $\omega=1$, представленного на *s-плоскости*. Как описывалось в предыдущей главе, фильтры Баттерворта имеют полюса, расположенные на *s*-плоскости точно по окружности. Поскольку фильтр является низкочастотным, нули не используются. Радиус окружности, соответствующий частоте среза $\omega=1$, равен *единице*. Блок 3 блок-схемы алгоритма (строки 1080 и 1090) вычисляет местоположение каждой пары полюсов в прямоугольных координатах. Программные переменные RP и IP являются соответственно действительной и мнимой частями местоположения полюса. Эти программные переменные соответствуют σ и ω, где пара полюсов расположена при $\sigma\pm j\omega$. Вычисляемое местоположение конкретного полюса зависит от числа полюсов в фильтре и обрабатываемого в данный момент каскада, программные переменные NP и P% соответственно.

Деформация круга в эллипс

Для реализации фильтра Чебышева данные *круговые* образцы полюсов должны быть трансформированы в *эллиптические* образцы. Относительная сплюснутость эллипса определяет количество пульсаций частотной характеристики в полосе пропускания фильтра. Если местоположение полюсов на окружности задается при помощи σ и ω, соответствующее местоположение на эллипсе σ' и ω' задается при помощи:

$$\sigma' = \frac{\sigma \sinh(v)}{k};$$

$$\omega' = \frac{\omega \cosh(v)}{k}, \qquad (33.9)$$

где

$$v = \frac{\sinh^{-1}(1/\epsilon)}{NP},$$

$$k = \cosh\left(\frac{1}{NP}\cosh^{-1}\frac{1}{\epsilon}\right),$$

$$\epsilon = \left[\left(\frac{100}{100-PR}\right)^2 - 1\right]^{1/2}.$$

```
100 'CHEBYSHEV FILTER-  COEFFICIENT CALCULATION
110 '
120                                 'INITIALIZE VARIABLES
130 DIM A[22]                       'holds the "a" coefficients
140 DIM B[22]                       'holds the "b" coefficients
150 DIM TA[22]                      'internal use for combining stages
160 DIM TB[22]                      'internal use for combining stages
170 '
180 FOR I% = 0 TO 22
190   A[I%] = 0
200   B[I%] = 0
210 NEXT I%
220 '
230 A[2] = 1
240 B[2] = 1
250 PI = 3.14159265
260                                 'ENTER THE FILTER PARAMETERS
270 INPUT "Enter cutoff frequency  (0 to .5):       ", FC
280 INPUT "Enter  0 for LP,  1 for HP filter:       ", LH
290 INPUT "Enter percent ripple    (0 to 29):       ", PR
300 INPUT "Enter number of poles (2,4,...20):       ", NP
310 '
320 FOR P% = 1 TO NP/2              'LOOP FOR EACH POLE-ZERO PAIR
330   '
340   GOSUB 1000                    'The subroutine in Fig. 33-8
350   '
360   FOR I% = 0 TO 22              'Add coefficients to the cascade
370     TA[I%] = A[I%]
380     TB[I%] = B[I%]
390   NEXT I%
400   '
410   FOR I% = 2 TO 22
420     A[I%] = A0*TA[I%] + A1*TA[I%-1] + A2*TA[I%-2]
430     B[I%] =      TB[I%] - B1*TB[I%-1] - B2*TB[I%-2]
440   NEXT I%
450   '
460 NEXT P%
470 '
480 B[2] = 0                        'Finish combining coefficients
490 FOR I% = 0 TO 20
500   A[I%] = A[I%+2]
510   B[I%] = -B[I%+2]
520 NEXT I%
530 '
540 SA = 0                          'NORMALIZE THE GAIN
550 SB = 0
560 FOR I% = 0 TO 20
570   IF LH = 0 THEN SA       = SA  +  A[I%]
580   IF LH = 0 THEN SB       = SB  +  B[I%]
590   IF LH = 1 THEN SA       = SA  +  A[I%] * (-1)^I%
600   IF LH = 1 THEN SB       = SB  +  B[I%] * (-1)^I%
610 NEXT I%
620 '
630 GAIN = SA / (1 - SB)
640 '
650 FOR I% = 0 TO 20
660   A[I%] = A[I%] / GAIN
670 NEXT I%
680 '                               'The final recursion coefficients are
690 END                             'in A[ ] and B[ ]
```

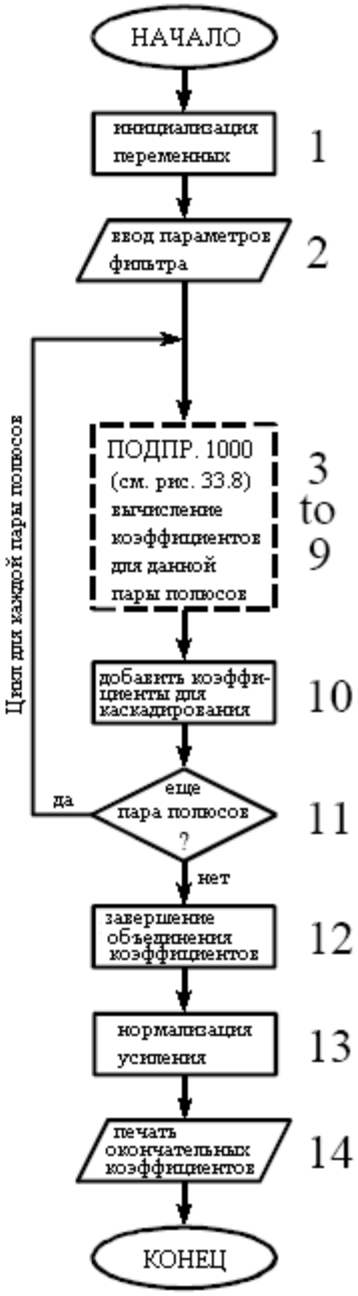


Рис. 33.7 Проектирование фильтра Чебышева - Баттерворта

Точно так же, как при работе с окружностью используются функции обычного синуса и косинуса, для задания эллипса в этих уравнениях используются функции гиперболического синуса и косинуса. Сплюснутость эллипса управляется переменной PR, которая численно равна проценту пульсаций амплитудно-частотной характеристики в полосе пропускания фильтра. Переменные $\in$, v и k введены для упрощения уравнения и представлены в программе с помощью ES, VX и KX соответственно. В дополнение к преобразованию из окружности в эллипс для удержания частоты среза равной единице эти

уравнения корректируют местоположения полюсов. Поскольку многие языки программирования не поддерживают гиперболические функции, здесь используются следующие тождества:

$$\sinh(x) = \frac{e^{x} - e^{-x}}{2}$$

$$\cosh(x) = \frac{e^{x} + e^{-x}}{2}$$

$$\sinh^{-1}(x) = \log_e\left[x + \left(x^2 + 1\right)^{1/2}\right]$$

$$\cosh^{-1}(x) = \log_e\left[x + \left(x^2 - 1\right)^{1/2}\right]$$

Эти уравнения приводят к неверным результатам для PR≥*30* и PR=*0*. Чтобы использовать эту программу для расчета фильтров Баттерворта (т.е. при нулевой пульсации, PR=*0*), строки программы, реализующие эти уравнения, должны быть обойдены (строка 1120).

Преобразование непрерывности в дискретность

Наиболее общим методом преобразования образцов полюсов и нулей из *s*-области в *z*-область является **билинейное преобразование**. Это математический метод *конформного отображения*, в котором одна комплексная плоскость алгебраически искривляется или деформируется в другую комплексную плоскость. Билинейное преобразование трансформирует *H(s)* в *H(z)* при помощи следующей подстановки:

$$s \rightarrow \frac{2\left(1 - z^{-1}\right)}{T\left(1 + z^{-1}\right)}. \qquad (33.10)$$

То есть мы записываем уравнение для *H(s)*, а затем заменяем каждое *s* выражением приведенным выше. В большинстве случаев используется *T=2tan(1/2)=1,093*. Это приводит к тому, что диапазон частот от *0* до *π* радиан/секунда в *s*-области, отображается в диапазон частот от 0 до бесконечности радиан в *z*-области. Не вдаваясь в подробности, отметим, что билинейное преобразование обладает всеми желаемыми свойствами необходимыми для превращения *s*-плоскости в z-плоскость, так же, как и для отображения линий в окружности. Вот пример того, как это все работает. Для непрерывной системы с единственной парой полюсов, расположенных в $p_1=\sigma+j\omega$ и $p_2=\sigma-j\omega$, передаточная функция в *s*-области определяется как:

$$H(s) = \frac{1}{(s - p_1)(s - p_2)}.$$

```
1000 'THIS SUBROUTINE IS CALLED FROM FIG. 33-7,  LINE 340
1010 '
1020 'Variables entering subroutine:     PI, FC, LH, PR, HP, P%
1030 'Variables exiting subroutine:      A0, A1, A2, B1, B2
1040 'Variables used internally:         RP, IP, ES, VX, KX, T, W, M, D, K,
1050 '                                   X0, X1, X2, Y1, Y2
1060 '
1070 '                                   'Calculate pole location on unit circle
1080 RP = -COS(PI/(NP*2) + (P%-1) * PI/NP)
1090 IP =  SIN(PI/(NP*2) + (P%-1) * PI/NP)
1100 '
1110 '                                   'Warp from a circle to an ellipse
1120 IF PR = 0 THEN GOTO 1210
1130 ES = SQR( (100 / (100-PR))^2 -1 )
1140 VX = (1/NP) * LOG( (1/ES) + SQR( (1/ES^2) + 1) )
1150 KX = (1/NP) * LOG( (1/ES) + SQR( (1/ES^2) - 1) )
1160 KX = (EXP(KX) + EXP(-KX))/2
1170 RP = RP * ( (EXP(VX) - EXP(-VX) ) /2 ) / KX
1180 IP  = IP * ( (EXP(VX) + EXP(-VX) ) /2 ) / KX
1190 '
1200 '                                   's-domain to z-domain conversion
1210 T  = 2 * TAN(1/2)
1220 W  = 2*PI*FC
1230 M  = RP^2 + IP^2
1240 D = 4 - 4*RP*T + M*T^2
1250 X0 = T^2/D
1260 X1 = 2*T^2/D
1270 X2 = T^2/D
1280 Y1 = (8 - 2*M*T^2)/D
1290 Y2 = (-4 - 4*RP*T - M*T^2)/D
1300 '
1310 '                                   'LP TO LP, or LP TO HP
1320 IF LH = 1 THEN K = -COS(W/2 + 1/2) / COS(W/2 - 1/2)
1330 IF LH = 0 THEN K =  SIN(1/2 - W/2) / SIN(1/2 + W/2)
1340 D = 1 + Y1*K - Y2*K^2
1350 A0 = (X0 - X1*K + X2*K^2)/D
1360 A1 = (-2*X0*K + X1 + X1*K^2 - 2*X2*K)/D
1370 A2 = (X0*K^2 - X1*K + X2)/D
1380 B1 = (2*K + Y1 + Y1*K^2 - 2*Y2*K)/D
1390 B2 = (-K^2 - Y1*K + Y2)/D
1400 IF LH = 1 THEN A1 = -A1
1410 IF LH = 1 THEN B1 = -B1
1420 '
1430 RETURN
```

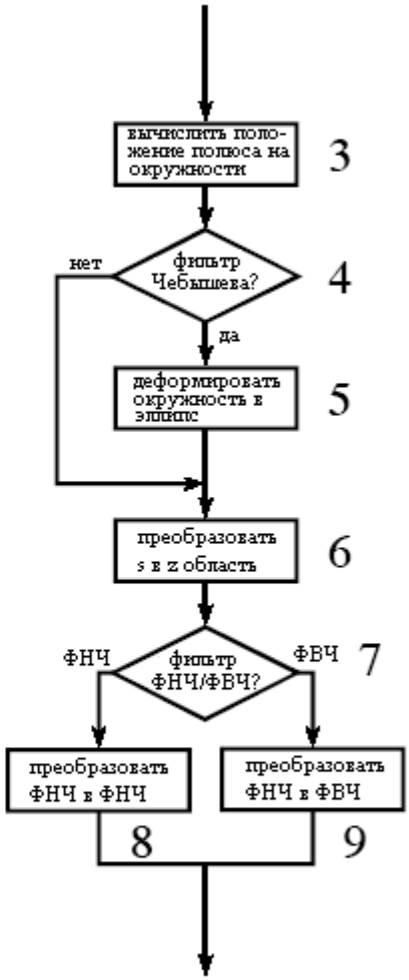


Рис. 33.8 Подпрограмма, вызываемая программой на рис. 33.7

Билинейное преобразование при помощи замены каждого s на выражение, показанное в (33.10), превращает непрерывную систему в дискретную. При этом создается передаточная функция в z-области, также содержащая два полюса. Сложность заключается в том, что после подстановки передаточная функция получается в очень неудобном виде:

$$H(z) = \frac{1}{\left(\frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)} - (\sigma + j\omega)\right)\left(\frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)} - (\sigma - j\omega)\right)}.$$

Выполнив длительные и утомительные алгебраические преобразования, это выражение можно привести к стандартному виду уравнения (33.3) с рекурсивными коэффициентами, определяемыми как:

$$a_0 = \frac{T^2}{D};$$

$$a_1 = \frac{2T^2}{D};$$

$$a_2 = \frac{T^2}{D};$$ (33.11)

$$b_1 = \frac{8 - 2MT^2}{D};$$

$$b_2 = \frac{-4 - 4\sigma T - MT^2}{D},$$

где $M=\sigma^2+\omega^2$, $T=2\tan(1/2)$, $D=4-4\sigma T+MT^2$. Переменные M, T и D не имеют физического смысла, они вводятся просто для того, чтобы сделать уравнение короче.

Эти уравнения используются в строках 1200-1290 для преобразования, хранящегося в переменных RP и IP, местоположения пары полюсов в s-области непосредственно в рекурсивные коэффициенты, хранящиеся в переменных X0, X1, X2, Y1, Y2. Другими словами, мы вычисляем промежуточный результат: рекурсивные коэффициенты одного каскада *низкочастотного* фильтра с частотой среза равной *единице*.

Преобразование частоты НЧ в НЧ

Преобразование частоты рекурсивного фильтра также выполняется с помощью метода конформного отображения. Предположим, что нам известна передаточная функция рекурсивного фильтра низкой частоты с частотой среза равной единице. Передаточная функция подобного фильтра низкой частоты с другой частотой среза W находится при помощи **преобразования НЧ в НЧ**. Это может быть сделано и с помощью *подстановки переменных* так же, как в случае билинейного преобразования. Мы начинаем с записи передаточной функции фильтра с единичной частотой среза, а затем заменяем каждое z^{-1} на следующее выражение:

$$z^{-1} \rightarrow \frac{z^{-1} - k}{1 - kz^{-1}},$$ (33.12)

где

$$k = \frac{\sin\left(\frac{1}{2} - \frac{W}{2}\right)}{\sin\left(\frac{1}{2} + \frac{W}{2}\right)}.$$

Это дает передаточную функцию фильтра с новой частотой среза. Следующие расчетные уравнения являются результатом применения такой подстановки к передаточным функциям, в основе которых лежат биквадратные уравнения, то есть к системам, содержащим не более чем два полюса и два нуля:

$$a_0=(a_0-a_1k+a_2k^2)/D;$$

$$a_1=(-2a_0k+a_1+a_1k^2-2a_2k)/D;$$

$$a_2=(a_0k^2-a_1k+a_2)/D; \qquad (33.13)$$

$$b_1=(2k+b_1+b_1k^2-2b_2k)/D;$$

$$b_2=(-k^2-b_1k+b_2)/D,$$

где $D=1+b_1k-b_2k$,

$$k=\frac{\sin\left(\frac{1}{2}-\frac{W}{2}\right)}{\sin\left(\frac{1}{2}+\frac{W}{2}\right)}.$$

Преобразование частоты НЧ в ВЧ

Для преобразования отклика системы из низкочастотного в высокочастотный при одновременном изменении частоты среза вышеупомянутое преобразование может быть модифицировано. Это осуществляется с помощью **преобразования НЧ в ВЧ** через следующую подстановку:

$$z^{-1}\rightarrow\frac{-z^{-1}-k}{1+kz^{-1}}, \qquad (33.14)$$

где

$$k=-\frac{\cos\left(\frac{W}{2}+\frac{1}{2}\right)}{\cos\left(\frac{W}{2}-\frac{1}{2}\right)}.$$

Так же, как и ранее все это может быть сведено к расчетным уравнениям рекурсивных коэффициентов систем с двумя полюсами и двумя нулями. Как оказалось, за исключением двух небольших отличий эти уравнения идентичны уравнениям (33.13). Отличается значение k (оно такое, как дано в уравнении (33.14)), и у двух коэффициентов a_1 и b_1 знак изменен на противоположный. Эти уравнения, обеспечивающие желаемую частоту среза и желаемый выбор фильтра верхних или фильтра низких частот, вычисляются в программе в строках с 1330 по 1410.

Лучшее и худшее в ЦОС

Эта книга базируется на простой посылке: **большинство способов цифровой обработки сигналов может быть применено и понято при минимальном использовании математики**. Идея заключается в том, чтобы обеспечить ученых и инженеров *инструментом* для решения проблем ЦОС, которые возникают в их научных исследованиях и проектной деятельности, напрямую не связанных с ЦОС.

Эти последние четыре главы являются другой стороной медали: методы ЦОС, которые могут быть поняты *только* при помощи обширной математики. Например, рассмотрим только что описанную процедуру проектирования фильтров Чебышева-Баттерворта. Это самое *лучшее* в ЦОС, ряд изящных математических шагов, ведущих к оптимальному решению. Однако, это и самое *худшее* в ЦОС, метод проектирования настолько сложный, что большинство ученых и инженеров будут искать другую альтернативу.

Где вписываетесь в эту схему Вы? Это зависит от того, *кто* Вы и для *чего* Вы планируете использовать ЦОС. Материал последних четырех глав дает теоретическую основу для обработки сигналов. Если Вы планируете делать *карьеру* в ЦОС, Вы должны иметь детальное понимание этой математики. С другой стороны, специалистам других областей науки и инженерного дела необходимо знать только, как *используется* ЦОС, а не как она производится. Для этой группы теоретический материал, скорее фон, чем центральная тема.

Толковый словарь

Автокорреляция: Корреляция сигнала с самим собой. Полезна поскольку преобразование Фурье автокорреляции является энергетическим спектром исходного сигнала.

Аддитивность: Математическое свойство, являющееся необходимым для линейных систем. Если *a* на входе дает *p* на выходе, и если *b* на входе дает *q* на выходе, то *a+b* на входе дает *p+q* на выходе.

Амплитудная модуляция: Метод, используемый в радиосвязи для объединения сигнала, переносящего информацию (наподобие звукового), с несущим колебанием. Обычно реализуется при помощи перемножения двух этих сигналов.

Анализ: Прямое преобразование Фурье; вычисление частотной области по временной области. Смотри для сравнения *синтез*.

Апертура дискретизации: Область непрерывного изображения, которая при оцифровке дает отдельный пиксель. Обычно почти такого же размера, что и интервал между отсчетами.

Ассемблер: Язык программирования низкого уровня, непосредственно манипулирующий регистрами и внутренними аппаратными средствами микропроцессора. Смотри для сравнения *язык высокого уровня*.

ASCII: Метод для представления букв и цифр в двоичном виде (акроним от: американский стандартный код для обмена информацией – прим. ред. перев.). Каждому символу присваивается число между *0* и *127*. Широко применяется в компьютерах и телекоммуникации.

Бабочка: Основная схема вычислений, используемая в БПФ. Преобразует два комплексных числа в два других комплексных числа.

Базисные функции: набор форм волн, используемых при разложении. Например, базисными функциями для разложения Фурье являются волны синуса и косинуса единичной амплитуды.

Бегущая сумма: Операция, используемая с дискретными сигналами, имитирующая интегрирование непрерывного сигнала. Также называется *дискретным интегралом*.

Белый шум: Случайный шум, обладающий пологим частотным спектром. Появляется тогда, когда в каждом отсчете временной области не содержится никакой информации относительно других отсчетов. Смотри для сравнения *шум 1/f*.

Бесконечная импульсная характеристика (БИХ): Импульсная характеристика, имеющая бесконечное число ненулевых значений, наподобие затухающей экспоненты. Часто используется для того, чтобы подчеркнуть, что фильтр реализуется с помощью рекурсии, а не свертки.

Биквадраты: Аналоговые или цифровые системы с двумя полюсами и не более, чем с двумя нулями. Часто соединяются последовательно для создания более сложных конструкций фильтра.

Билинейное преобразование: Техника, используемая для отображения *s*-плоскости на *z*-плоскости. Позволяет преобразовать аналоговые фильтры в эквивалентные цифровые фильтры.

БПФ свертка: Метод свертки сигналов посредством перемножения их частотных спектров. Носит такое название потому, что для эффективного перехода между временной и частотной областями используется БПФ.

Бункерование: Метод формирования гистограммы, при котором данные (или сигнал) имеют многочисленные уровни квантования, как, например, в числах с плавающей запятой.

Быстрое преобразование Фурье (БПФ): Эффективный алгоритм для вычисления дискретного преобразования Фурье (ДПФ). В некоторых случаях снижает время исполнения в *сотни раз*.

Восстанавливающий фильтр: Низкочастотный аналоговый фильтр, расположенный после цифро-аналогового преобразователя. Сглаживает ступенчатую форму волны за счет удаления частот выше половины частоты дискретизации.

Восстановление сигнала: Возврат сигнала к его первоначальной форме после того, как он был изменен или некоторым образом ухудшился. Одно из основных применений *фильтрации*.

Временная область: Сигнал, имеющий в качестве независимой переменной время. Используется также в виде общей ссылки на любую область, в которой осуществлялся сбор данных.

Высококачественное воспроизведение (Hi-Fi): Воспроизведение музыки с высоким качеством таким, какое обеспечивают проигрыватели компакт-дисков.

Высокоскоростная свертка: Другое название БПФ свертки.

Высота звука: Восприятие человеком основной частоты непрерывного тона. Смотри для сравнения *тембр*.

Гамма кривая: Математическая функция или таблица поиска, соотносящая значение хранимого пикселя с его яркостью в показываемом изображении. Также называется *преобразование шкалы серого*.

Гарвардская архитектура: Внутренняя организация компьютера, при которой программа и данные расположены в отдельных запоминающих устройствах, доступ к которым осуществляется по отдельным шинам; типично для микропроцессоров, используемых в ЦОС. Смотри для сравнения *фон Неймановскую архитектуру*.

Гармоники: Частотные составляющие периодического сигнала, всегда в целое число раз кратные основной частоте. Основная частота является первой гармоникой, составляющая с частотой в два раза больше - второй гармоникой и т.д.

Гауссиан: Колоколообразная кривая с общей формой вида: e^{x^2}. Гауссиан обладает многими уникальными свойствами. Также называется *нормальным распределением*.

Гиперпространство: Термин, используемый при обнаружении цели и анализе нейронных сетей. Один параметр может быть интерпретирован графически как *линия*, два параметра как *плоскость*, три параметра как *пространство*, а больше чем три параметра как *гиперпространство*.

Гистограмма: Показывает распределение значений в сигнале. Ось *x* показывает возможные значения, которые могут принимать отсчеты; ось *y* показывает число отсчетов, имеющих каждое значение.

Гомоморфный: Метод ЦОС для отделения сигналов, объединенных нелинейным способом наподобие умножения или свертки. При помощи соответствующего преобразования нелинейная задача преобразуется к линейной.

GIF: Типичный формат файла изображения, использующий LZW сжатие (без потерь). Широко используется в интернете для графики. Смотри для сравнения *TIFF* и *JPEG*.

Двойная точность: Стандарт для системы обозначений с плавающей запятой, использующей, для представления каждого числа, *64* бита. Смотри для сравнения *одинарная точность*.

Двунаправленная фильтрация: Рекурсивный метод, используемый для получения фильтра с нулевой фазой. Сначала сигнал фильтруется слева направо, затем промежуточный сигнал фильтруется справа налево.

Действительное БПФ: Модифицированная версия БПФ. Когда временная область полностью действительна (т.е. мнимая часть временной области равна нулю), оно приблизительно на 30 % быстрее, чем стандартное БПФ.

Действительное ДПФ: дискретное преобразование Фурье, использующее только действительные (обычные) числа. Менее мощная, но более простая техника, чем комплексное ДПФ. Смотри для сравнения *комплексное ДПФ*.

Действительное преобразование Фурье: Любое из членов семейства преобразований Фурье, использующее только действительные (в противоположность мнимым или комплексным) числа. Смотри для сравнения *комплексное преобразование Фурье*.

Действительная часть: Часть комплексного числа, у которого нет члена *j*, такая как 3 в *3+2j*. В действительном преобразовании Фурье *действительная* часть относится к части частотной области, содержащей амплитуды косинусных волн даже тогда, когда члены с *j* отсутствуют.

Декомпозиция: Процесс разложения сигнала на две или более аддитивных компоненты. Часто, определенно относится к *прямому преобразованию Фурье*, разлагающему сигнал на синусоиды.

Дельта кодирование: Широкий термин, относящийся к методам, в которых данные хранятся в виде разницы между смежными отсчетами. Используется в АЦП, сжатии данных и многих других приложениях.

Дельта сигма: Метод аналого-цифрового преобразования популярный в обработке голоса и

музыки. Использует очень высокую частоту дискретизации всего с одним битом на отсчет и с последующей децимацией.

Дельта функция: Нормализованный импульс. Дискретная дельта функция является сигналом, состоящим из всех нулей за исключением нулевого отсчета, имеющего значение единицы. Непрерывная дельта функция подобна, но более абстрактна.

Децибел УМЗ: Уровень мощности звука. Для выражения интенсивности звуковой волны используется логарифмический масштаб: *0* dB УМЗ едва уловим; *60* dB УМЗ - нормальная речь и *140* dB УМЗ вызывает повреждение уха.

Децимация: Снижение частоты дискретизации оцифровываемого сигнала. Обычно включает низкочастотную фильтрацию с последующим отбрасыванием отсчетов. Смотри для сравнения *интерполяция*.

Диаграмма нулей и полюсов: Термин, используемый в преобразовании Лапласа и *z*-преобразовании. Графическое изображение местоположения полюсов и нулей на *s*-плоскости или *z*-плоскости.

Динамический диапазон: Самая большая амплитуда, с которой может иметь дело система, деленная на присущий системе шум. Употребляется также для указания числа битов, используемых в АЦП. Может также употребляться с иными параметрами, чем амплитуда; смотри *динамический диапазон по частоте*.

Динамический диапазон по частоте: Отношение самой высокой из частот, с которыми может иметь дело система, к самой низкой частоте. Аналоговые системы обычно имеют намного больший динамический диапазон по частоте, чем цифровые системы.

Дискретная производная: Операция над дискретными сигналами, аналогичная производной от непрерывных сигналов. Более хорошее название - *первая разность*.

Дискретно-временное преобразование Фурье (ДВПФ): Член семейства преобразования Фурье, связанный с *дискретными* и *апериодическими* сигналами временной области.

Дискретное косинусное преобразование (ДКП): Родственное преобразованию Фурье. Разлагает сигнал на косинусные волны. Используется в сжатии данных.

Дискретное преобразование Фурье (ДПФ): Член семейства преобразования Фурье, связанный с *дискретными* и *периодическими* сигналами временной области.

Дискретный интеграл: Операция над дискретными сигналами, аналогичная интегралу от непрерывных сигналов. Более хорошее название – *бегущая сумма*.

Дискретный сигнал: Сигнал, использующий квантованные переменные, наподобие находящегося в компьютере оцифрованного сигнала.

δ импульс: Сигнал, состоящий из всех нулей, кроме очень короткого импульса. Для дискретных сигналов импульс состоит из единственного ненулевого отсчета. Для непрерывных сигналов ширина импульса должна быть намного короче, чем отклик присущий любой системе, с которой используется сигнал.

Единичная окружность: Окружность на *z*-плоскости с *r=1*. Значения на этой окружности являются частотной характеристикой системы.

Единичный импульс: Другое название *дельта функции*.

Естественная частота: По сравнению с периодами в секунду (герц), частота, выраженная в радианах в секунду. Чтобы преобразовывать частоту (в герцах) в естественную частоту, умножьте частоту на 2π.

JPEG: Типичный формат файла изображения, использующий сжатие (с потерями) преобразованием. Широко используется в интернете для графики. Смотри для сравнения *GIF* и *TIFF*.

Завал: Жаргон, используемый для описания крутизны перехода между полосой пропускания и полосой заграждения фильтра. *Быстрый* завал означает то, что переход резкий; *медленный* завал означает то, что он постепенный.

Зависимая переменная: В сигнале зависимая переменная зависит от значения независимой переменной. Пример: когда напряжение изменяется во времени, время является независимой переменной, а напряжение является зависимой переменной.

Закон "А": Стандарт компандирования, используемый в Европе. Позволяет при помощи неодинаковых уровней квантования представить цифровые голосовые сигналы всего *8* битами вместо *12* битов. Для сравнения смотри *закон мю*.

Закон мю: Стандарт компандирования, используемый в США. Позволяет за счет неодинаковых уровней квантования представить

цифровые голосовые сигналы всего *8* битами вместо *12* битов. Смотри для сравнения *закон* "*A*".

Закрытие: морфологическое действие, определяемое как операция эрозии, сопровождаемая операцией расширения.

Заострение: Операция по обработке изображения, делающая контуры более резкими.

Захват кадра: аналого-цифровой преобразователь, используемый для оцифровки и хранения кадра (изображения) телевизионного сигнала.

Звонкие: Звуки речи человека, которые образуются в результате прохождения импульсов воздуха через голосовые связки. Примером звонких звуков являются гласные. Смотри для сравнения *фрикативные*.

Зрительная ямка: Небольшая область сетчатки глаза, оптимизированная под зрение с высоким разрешением.

z-область: Область, определяемая *z*-преобразованием. Также называется z-плоскостью.

z-преобразование: Математический метод, используемый для анализа дискретных систем, описываемых конечно-разностными уравнениями, наподобие рекурсивных (БИХ) фильтров. Преобразует сигнал временной области в сигнал z-области.

Изображение шкалы серого: цифровое изображение, где каждый пиксель отображен в оттенках серого между черным и белым; также называется черно-белым изображением.

Импульсная декомпозиция: Разбивает *N* точечный сигнал на *N* сигналов, каждый из которых содержит один отсчет из исходного сигнала, при всех других отсчетах равных нулю. Это основа свертки.

Импульсная последовательность: Сигнал, состоящий из серии импульсов, расположенных на одинаковом расстоянии друг от друга.

Импульсная характеристика: Выходной сигнал системы при подаче на вход нормализованного импульса (дельта функции).

Инвариантность сдвига: Свойство многих систем. Сдвиг входного сигнала не дает ничего больше, чем сдвиг выходного сигнала. Это означает, что характеристики системы (или другой независимой переменной) во времени не изменяются.

Инверсия спектра: Метод изменения ядра фильтра таким образом, чтобы соответствующая частотная характеристика перевернулась сверху вниз. Это может превратить фильтры нижних частот в фильтры верхних частот, полосно-пропускающие в полосно-заграждающие и т.д.

Интеграл свертки: Математическое уравнение, определяющее свертку для непрерывных систем; аналогично *сверточной сумме* для дискретных систем.

Интервал между отсчетами: Промежуток между отсчетами при оцифровке непрерывного изображения. Определяется как расстояние между пикселями от центра до центра.

Интерполяция: Увеличение частоты дискретизации оцифрованного сигнала. Обычно осуществляется посредством размещения нулей между исходными отсчетами с последующим использованием низкочастотной фильтрации. Смотри для сравнения *децимация*.

Истинно-отрицательный: Один из четырех возможных результатов теста обнаружения цели. Цель не присутствует, и тест правильно показывает, что ее нет.

Истинно-положительный: Один из четырех возможных результатов теста обнаружения цели. Цель присутствует, и тест правильно показывает, что она есть.

Исходный код: Компьютерная программа в написанной программистом форме; отличается от *исполняемого кода* - формы, которая непосредственно может быть запущена на компьютере.

Итеративный: Способ нахождения решения посредством постепенного подбора переменных в правильном направлении до достижения сходимости. Используется при реконструкции изображений в компьютерной томографии и в нейронных сетях.

Кадр: Отдельное изображение в телевизионном сигнале. В стандарте телевидения NTSC частота смены кадров составляет 30 кадров в секунду.

Калибратор пар линий: Инструмент, используемый для измерения разрешающей способности системы изображения. Содержит набор светлых и темных линий, которые к одному из концов сближаются друг с другом.

Кепстр: Трансформировано из "спектр". Используется в гомоморфной обработке для описания спектров, в тех случаях, когда

временная и частотная области обменялись своим естественным применением.

Классификаторы: Параметры, извлекаемые из большого набора данных и представляющие его. Например, размер области, амплитуда пика, крутизна фронта и т.д. Используется при распознавании образов.

Кодирование в частотной области: Один из двух основных способов, которыми информация может кодироваться в сигнале. Информация содержится в амплитуде, частоте и фазе составляющих сигнал синусоид. Лучшим примером являются звуковые сигналы. Смотри для сравнения *кодирование во временной области*.

Кодирование во временной области: Информация в сигнале, содержащаяся в очертании формы волны. Смотри для сравнения *кодирование в частотной области*.

Кодирование длины повторов: Простой метод сжатия данных со многими вариациями. Символы, которые повторяются много раз подряд, заменяются кодами, указывающими символ и длину повтора.

Кодирование Хаффмана: Способ сжатия данных, присваивающий часто встречающимся символам меньшее количество битов, чем редко встречающимся символам.

Кольцевой буфер: Метод хранения данных, используемый при обработке в реальном времени; каждый вновь полученный отсчет заменяет наиболее старый отсчет в памяти.

Комбинированный сигнал: Интегральные схемы, содержащие и аналоговую и цифровую электронику, наподобие АЦП помещенного в цифровой процессор обработки сигнала.

Компандирование: "S" образная нелинейность, позволяющая оцифровывать голосовой сигнал, используя всего 8 битов вместо 12 битов. Европа использует *закон* "*A*", тогда как Соединенные Штаты используют версию называемую *закон мю*.

Комплексная плоскость: Графическая интерпретация комплексных чисел с действительной частью по оси *x* и мнимой частью по оси *y*. Это аналог *числовой оси*, используемой с обычными числами.

Комплексная экспонента: Комплексное число в форме: e^{a-bj}. Она полезна в науке и технике, поскольку при помощи формулы Эйлера позволяет представлять синусоиды.

Комплексное ДПФ: Дискретное преобразование Фурье, использующее комплексные числа. Более сложная и мощная техника, чем *действительное* ДПФ.

Комплексное преобразование Фурье: Любой из четырех членов семейства преобразования Фурье, записанный при помощи комплексных чисел. Смотри для сравнения *действительное преобразование Фурье*.

Комплексное сопряжение: Изменение знака мнимой части комплексного числа. Часто обозначается звездочкой, помещаемой рядом с переменной. Пример: если $A = 3+2j$ то $A^* = 3-2j$.

Комплексные числа: *Действительные числа*, (используемые в каждодневной математике) плюс *мнимые числа* (числа содержащие член *j*, где $j = \sqrt{-1}$). Пример: $3+2j$.

Композитный видео сигнал: Аналоговый телевизионный сигнал, содержащий синхронизирующие импульсы, разделяющие строки и кадры.

Компьютерная томография: Метод, используемый для реконструкции изображения внутренностей объекта по его рентгеновским проекциям. Широко используется в медицине; одно из наиболее ранних приложений ЦОС. Старое название КАТ сканнер.

Конечная импульсная характеристика (КИХ): Импульсная характеристика, имеющая конечное число ненулевых значений. Часто используется для того, чтобы показать, что фильтр реализуется с помощью свертки, а не с помощью рекурсии.

Контрастность: Различие между яркостью объекта и яркостью фона. Смотри для сравнения *яркость*.

Концевые эффекты: Плохо ведущие себя концы отфильтрованного сигнала, как результат того, что ядро фильтра погружено во входном сигнале не полностью.

Корреляция: Математическая операция, выполняемая так же, как и свертка, за исключением переворота одного сигнала слева направо. Это оптимальный способ обнаружения известной формы волны в сигнале.

Коэффициент вариации: Общий способ оценки небольших видоизменений (шума) в данных. Определяется как: 100%×стандартное отклонение/среднее значение.

Кривая рабочей характеристики приемника: Графическое изображение, показывающее, как выбор порога воздействует на эффективность решения задачи обнаружения цели.

Круговая свертка: Совмещение имен, которое может произойти во временной области при перемножении сигналов частотной области. Каждый период во временной области выходит за свои пределы в смежные периоды.

Кругообразность: Видимость того, что конец сигнала соединен с его началом. Она появляется при рассмотрении только одного периода периодического сигнала.

Крутой спуск: Стратегия, используемая при разработке итеративных алгоритмов. Аналогично поиску низины долины, посредством движения вниз по холму.

Кули и Тьюки: Отдадим должное Дж. У. Кули и Дж. У. Тьюки за то, что они подарили миру БПФ в статье, опубликованной в 1965 году.

Линейная система: По определению - система, обладающая свойствами аддитивности и однородности.

Линейная фаза: Система с фазовой характеристикой, представляющей собой прямую линию. Обычно она важна, поскольку означает, что импульсная характеристика симметрична слева направо, это делает нарастающий фронт в выходном сигнале, выглядящим точно так же, как падающий фронт. Смотри также *нулевая фаза*.

Ложноотрицательный: Один из четырех возможных результатов попытки определения цели. Цель присутствует, но некорректно классифицируется, будто ее нет.

Ложноположительный: Один из четырех возможных результатов попытки определения цели. Цель отсутствует, но некорректно классифицируется, будто она есть.

Математическая эквивалентность: Способ использования комплексных чисел для представления действительных задач. Основывается на формуле Эйлера, приравнивающем синусоиды с комплексными экспонентами. Смотри для сравнения *подстановка*.

Микропроцессор ЦОС: Тип микропроцессора, спроектированный для быстрых математических вычислений. Часто содержит конвейер и/или Гарвардскую архитектуру. Называется также RISC процессором.

Мнимая часть: Часть комплексного числа, имеющая член *j* такая же, как и 2 в *3+2j*. В действительном преобразовании Фурье *мнимая часть* также относится к части частотной области, содержащей амплитуды синусоидальных волн, хотя член *j* не используется.

Модуляционная характеристика (МХ): Жаргон из области обработки изображений для *частотной характеристики*.

Морфологический: Обычно относится к простым нелинейным операциям, выполняемым с двоичными изображениями, наподобие эрозии и расширения.

Мультиплексирование: Объединение двух или более сигналов вместе для передачи. Это может быть выполнено многими различными способами.

Мультиплексирование в частотной области: Способ объединения сигналов для одновременной передачи, за счет размещения их в разных частях частотного спектра.

Мультичастотная: Системы, использующие более чем одну частоту дискретизации. Часто применяются в АЦП и ЦАП для получения более хороших характеристик работы при использовании меньшего количества электроники.

MFLOPS: Миллион операций с плавающей запятой в секунду; типичный способ выражения скорости компьютера. Смотри для сравнения *MIPS*.

MIPS: Миллион операций в секунду; общий способ выражения скорости компьютера. Смотри для сравнения *MFLOPS*.

MPEG: Стандарт сжатия для видео, наподобие цифрового телевидения.

Независимая переменная: Зависимая переменная в сигнале зависит от величины независимой переменной. Пример: Когда напряжение изменяется во времени, время является независимой переменной, а напряжение является зависимой переменной.

Непрерывный сигнал: Сигнал, сформированный из непрерывных (в противоположность дискретным) переменных. Пример: изменяющееся во *времени напряжение*. Часто взаимозаменяемо используется с фразой *аналоговый сигнал*.

Несущая волна: Термин, используемый в амплитудной модуляции радиосигналов. Относится к высокочастотной синусоидальной

волне, которая объединяется с сигналом более низкой частоты несущим информацию.

Нормальное распределение: Колоколообразная кривая с формой вида: e^{x^2}. Также называется *Гауссиан*.

Нулевая фаза: Система с полностью нулевой фазовой характеристикой. Появляется только тогда, когда импульсная характеристика обладает симметрией слева направо относительно нулевого отсчета. Смотри также *линейная фаза*.

Нуль: Термин, используемый в преобразовании Лапласа и z-преобразовании. Когда передаточная функция s-области или z-области записывается в виде одного полинома, деленного на другой полином, корни числителя являются *нулями* системы. Смотри также *полюс*.

NTSC: Телевизионный стандарт, используемый в США, Японии и других странах. Смотри для сравнения *PAL* и *SECAM*.

Область: Независимая переменная сигнала. Например, напряжение, изменяющееся во времени находится во *временной области*. Другими обычными областями являются *пространственная область* (наподобие изображений) и *частотная область* (результат преобразования Фурье).

Область сходимости: Термин, используемый в преобразовании Лапласа и z-преобразовании. Те области на s-плоскости и z-плоскости, которые имеют определенное значение.

Обнаружение цели: Решение присутствует ли объект или условие, основанное на измеренных значениях.

Обработка в реальном режиме времени: Обработка данных по мере их сбора, вместо их сохранения для более позднего использования. Пример: алгоритмы ЦОС для управления эхом при междугородних звонках.

Обратная свертка: Операция обратная свертке: если $x[n]*h[n] = y[n]$, то при заданных только $h[n]$ и $y[n]$ найти $x[n]$. Обратная свертка обычно выполняется делением частотных спектров.

Обратное преобразование: Уравнение синтеза преобразования Фурье, вычисляющее временную область из частотной области. Смотри для сравнения *прямое преобразование*.

Обратное проецирование: техника, применяемая в компьютерной томографии для реконструкции изображения по его видам. Если не используется совместно с более передовыми методами, дает плохое качество изображения.

Обратное проецирование с фильтрацией: Техника, используемая в компьютерной томографии для реконструкции изображения по его видам. Виды *фильтруются* с последующим *обратным проецированием*.

Обращение спектра: Метод изменения ядра фильтра таким образом, чтобы соответствующая частотная характеристика перевернулась слева направо. Это превращает фильтры нижних частот в фильтры верхних частот.

Обучающий алгоритм: Процедура, используемая для нахождения набора весов нейронной сети, основанная на примерах того, как должна работать сеть.

Ограниченная окном синк функция: Цифровой фильтр, используемый для отделения одной полосы частот от другой.

Одинарная точность: Система обозначений с плавающей запятой, использующая для представления каждого числа *32* бита. Смотри для сравнения *двойная точность*.

Однополюсный цифровой фильтр: Простой рекурсивный фильтр, имитирующий высокочастотный и низкочастотный *RC* фильтры в электронике.

Однородность: математическое свойство всех линейных систем. Если $x[n]$ на входе дает $y[n]$ на выходе, тогда $kx[n]$ на входе дает $ky[n]$ на выходе для любой постоянной k.

Окно Блэкмена: Плавная кривая, используемая при проектировании фильтров и в спектральном анализе, вычисляется по: $0,42-0,5cos(2\pi n/M)+0,08cos(4\pi n/M)$, где n изменяется от 0 до M.

Окно с плоской вершиной: Окно, используемое в спектральном анализе; обеспечивает точное измерение амплитуд спектральных составляющих. Должно использоваться ядро фильтра с ограниченной окном синк функцией.

Окно Хемминга: Плавная кривая, используемая при проектировании фильтров и в спектральном анализе, вычисляется по: $0,54-0,46cos(2\pi n/M)$, где n изменяется от 0 до M.

Октава: Множитель два по частоте.

Оптимальный фильтр: Фильтр, который в некоторых особых случаях является "наилучшим". Например, фильтры Винера дают

оптимальное соотношение сигнал/шум, а согласованные фильтры оптимальны для обнаружения цели.

Ослабление в полосе заграждения: Величина, посредством которой оценивается уменьшение по амплитуде частот в полосе заграждения, обычно выражается в децибелах. Используется для описания характеристик фильтра.

Основная мембрана: Небольшой орган в ухе, который действует, как анализатор спектра. Он позволяет разным частотам стимулировать различные волокна в улиточном нерве.

Основная частота: Частота, с которой повторяется форма периодической волны. Смотри для сравнения *гармоники*.

Отклик на прямоугольный импульс: Выходной сигнал системы, когда на ее вход подан прямоугольный импульс.

Отклик на край: В обработке изображений выходной сигнал системы, когда входным сигналом является край. Резкость отклика на край часто используется, как мера разрешающей способности системы.

Открытие: морфологическая операция, определенная, как операция расширения, сопровождаемая операцией эрозии.

Отрицательные частоты: Синусоиды могут быть записаны как с положительной частотой: $cos(\omega t)$, так и с отрицательной частотой: $cos(-\omega t)$. Отрицательные частоты включены в комплексное преобразование Фурье, делая его более мощным.

Ошибка квантования: Ошибка, появляющаяся в результате квантования сигнала. В большинстве случаев максимальная ошибка квантования составляет $\pm 1/2$ МЗР, а среднеквадратическая ошибка $1/\sqrt{12}$ МЗР. Также называется *шум квантования*.

Параллельные каскады: Комбинация двух или более каскадов с общим входом и просуммированными выходами.

Параметрическое пространство: Жаргон обнаружения цели. Один параметр может быть интерпретирован графически как *линия*, два параметра как *плоскость*, три параметра как *пространство*, а больше чем три параметра как *гиперпространство*.

Пара строк: Термин из области описания изображений для *периода*. Например, *5* периодов на миллиметр - то же самое, что и *5* пар строк на миллиметр.

Пары преобразования Фурье: Соответствующие друг другу формы волн во временной и частотной областях. Например, прямоугольный импульс и синк функция.

Пассивная гидролокация: Обнаружение подводных лодок и других подводных объектов по издаваемым ими звукам. Используется для тайного наблюдения.

Первая разность: Операция для дискретных сигналов, имитирующая первую производную для непрерывных сигналов; также называется *дискретной производной*.

Передаточная функция: Выходной сигнал, деленный на входной сигнал. Она встречается в нескольких различных формах, в зависимости от того, каким образом представляются сигналы. Например, в *s*-области и *z*-области это будет один полином, деленный на другой полином, и может быть выражена в виде *полюсов* и *нулей*.

Перекрестная декомпозиция: Разбиение сигнала на отсчеты с четными и нечетными номерами. Используется в БПФ.

Переменная составляющая: Термин для части сигнала, испытывающей колебания вокруг среднего значения.

Переходная характеристика: Выходной сигнал системы, когда на ее вход подана ступенчатая функция.

Пиксель: сокращение от "picture element" (элемент картинки). Отдельный отсчет в цифровом изображении.

Пилюлеобразная: Форма ядра фильтра, используемая в обработке изображений: круглая область постоянного значения, окруженная нулями.

Плавающая запятая: Один из двух характерных способов, с помощью которого в компьютерах хранятся числа. Плавающая запятая использует форму научной системы обозначений, в которой мантисса возводится в степень. Смотри для сравнения *фиксированная запятая*.

Погрешность: Повторяемая от опыта к опыту ошибка в измерении (или предсказании). Погрешность ограничена систематическими (повторяемыми) ошибками. Смотри для сравнения *точность*.

Подстановка: Способ использования комплексных чисел для представления

физических задач, наподобие проектирования электрических схем. В этом методе для приведения физической задачи к комплексной форме добавляется член *j*, а затем, для того чтобы снова вернуться назад, удаляется. Смотри для сравнения *математическая эквивалентность*.

Полная ширина на половине максимума (ПШПМ): Обычный способ измерения ширины пика в сигнале. Ширина пика измеряется на половине максимальной амплитуды пика.

Полоса заграждения: Полоса частот, которую фильтр спроектирован не пропускать.

Полоса перехода: Жаргон из области фильтров; полоса частот, между полосой пропускания и полосой заграждения, в которой происходит завал.

Полоса пропускания: Полоса частот, для которой фильтр разработан так, чтобы пропускать ее без изменений.

Полукадр: В телевидении с чересстрочной разверткой сначала выводятся четные строки кадра (изображения), а затем нечетные строки. Четные строки называются *четным полукадром*, а нечетные строки – *нечетным полукадром*.

Полутон: Простой способ печати изображений на бумаге. Оттенки серого создаются разнообразными образцами маленьких черных точек. Цветные полутона создаются красными, зелеными и синими точками.

Полюс: Термин, используемый в преобразовании Лапласа и *z*-преобразовании. Когда передаточная функция *s*-области или *z*-области записана в виде одного полинома, деленного на другой полином, то корни знаменателя являются *полюсами* системы, в то время как корни числителя являются *нулями*.

Полярная форма: Представление синусоид с помощью их модуля и фазы: $M\cos(\omega t+\varphi)$, где M – модуль, а φ – фаза. Смотри для сравнения *прямоугольная форма*.

Последовательное соединение: Комбинация двух или более каскадов, где выход одного каскада становится входом следующего.

Постоянная составляющая: Термин для части сигнала, не изменяющейся во времени. Среднее значение или среднее. Смотри для сравнения *переменная составляющая*.

Преобразование: Процедура, уравнение или алгоритм, преобразующая одну группу данных в другую группу данных.

Преобразование Лапласа: Математический метод анализа систем, описываемых дифференциальными уравнениями. Основной инструмент проектирования электрических цепей, таких как аналоговые фильтры. Преобразует сигнал временной области в *s*-область.

Преобразование Фурье: Семейство математических способов, основанных на разложении сигналов на синусоиды. В комплексном варианте сигналы раскладываются на комплексные экспоненты.

Преобразование шкалы серого: Преобразующая функция между значением хранимого пикселя и его яркостью в показываемом изображении. Также называется *гамма кривая*.

Преобразователь Гильберта (квадратурный фильтр – прим. ред. перев.**):** Система, обладающая следующими частотными характеристиками: Модуль=1, Фаза=90^0 для всех частот. Используется в системах связи для осуществления модуляции. Может быть аналоговой или цифровой.

Прибор с зарядовой связью: датчик света в электронных камерах. Сформирован из тонкой пленки кремния, содержащего двумерный массив светочувствительных областей, называемых *ямами*.

Причинная система: Система, имеющая нулевой выходной сигнал до тех пор, пока на ее входе не появится ненулевое значение (т.е. вход является *причиной* выхода). Импульсным откликом причинной системы является причинный сигнал.

Причинный сигнал: Любой сигнал, который имеет значение нуля для всех отрицательных пронумерованных отсчетов.

Пространственная область: Сигнал, имеющий в качестве независимой переменной расстояние (интервал). Изображения являются сигналами в пространственной области.

Протяженность шкалы серого: Сильно увеличивающаяся контрастность цифрового изображения для обеспечения детальной проверки уровней квантования в небольшом диапазоне. Уровни квантования вне этого диапазона отображаются, как насыщенные черный или белый.

Прямое преобразование: Уравнение анализа преобразования Фурье, вычисляющее частотную область по временной области. Смотри для сравнения *обратное преобразование*.

Прямоугольное окно: Сигнал с группой смежных точек, имеющих значение равное единице, и равный нулю во всех других местах. Обычно умножается на другой сигнал для выбора участка сигнала для обработки.

Прямоугольная форма: Представление синусоиды в виде: $A\cos(\omega t)+B\sin(\omega t)$, где A называется *действительной частью*, а B называется *мнимой частью* (даже притом, что они не являются комплексными числами).

PAL: Телевизионный стандарт, используемый в Европе. Смотри для сравнения *NTSC*.

Радар: RAdio Detection And Ranging (радио обнаружение и определение дальности). Техника эхолокации, использующая для обнаружения самолетов радиоволны.

Конечно-разностное уравнение: Уравнение, связывающее предыдущие и текущие отсчеты выходного сигнала с предыдущими и текущими отсчетами входного сигнала. Также называется *рекурсивным уравнением*.

Равенство Парсеваля: Уравнение, связывающее энергию во временной области с энергией в частотной области.

Разрешение по частоте: Способность различать или отделять близко расположенные частоты.

Раскачивающий шум: Добавление шума к аналоговому сигналу перед аналого-цифровым преобразованием, для того чтобы не дать оцифрованному сигналу "застыть" на одном значении.

Распределение Пуассона: Изменения в значении сигнала, являющиеся результатом того, что он представлен конечным числом частиц, наподобие: рентгеновских лучей, фотонов света или электронов. Другое название *шум Пуассона* и *статистический шум*.

Растекание спектра: Термин, используемый в спектральном анализе. Поскольку ДПФ может быть взято только от сигнала конечной длины, частотным спектром синусоиды является пик с хвостами. Эти хвосты рассматриваются, как *растекание* от основного пика.

Расширение: Морфологическая операция. При применении к двоичным изображениям расширение делает объекты больше и может объединять разъединенные объекты в единый объект.

Рекурсивные коэффициенты: Взвешенные значения, используемые в рекурсивных уравнениях. Рекурсивные коэффициенты определяют характеристики рекурсивных (БИХ) фильтров.

Рекурсивное уравнение: Уравнение, связывающее предыдущие и текущие отсчеты выходного сигнала с предыдущими и текущими значениями входного сигнала. Также называется *конечно-разностным уравнением*.

Ряд старшего порядка: Образец для преобразования изображения в последовательную форму. Действует так же, как написание на Английском (Русском - прим. ред. перев.): слева направо на первой строке, слева направо на второй строке и т.д.

Ряды Фурье: Член семейства преобразований Фурье связанный с *непрерывными* и *периодическими* сигналами временной области.

RGB кодирование: Представление цветного изображения путем задания для каждого пикселя уровня красного, зеленого и синего цветов.

RISC: Reduced Instruction Set Computer (компьютер с сокращенным набором команд), также называемый микропроцессором ЦОС. Меньшее количество команд программирования, допускающих намного более высокую скорость математических вычислений. Противоположностью является Complex Instruction Set Computer (компьютер с полным набором команд) наподобие Pentium.

Свертка в частотной области: Свертка, выполненная при помощи перемножения частотных спектров сигналов.

Сверточная сумма: Математическое уравнение, определяющее свертку для дискретных систем.

Свойство ассоциативности свертки: Записывается как $(a[n]*b[n])*c[n]=a[n]*(b[n]*c[n])$. Оно важно в обработке сигнала, поскольку описывает, как ведут себя последовательно соединенные каскады.

Свойство коммутативности свертки: Записывается как: $a[n]*b[n] = b[n]*a[n]$.

Сдвиг и вычитание: Операция по обработке изображения, создающая трехмерный или рельефный эффект.

Сейсмология: Раздел геофизики, имеющей дело с механическими свойствами земли.

Сепарабельное: Изображение, которое может быть представлено, как произведение его вертикальных и горизонтальных сечений. Используется для улучшения скорости свертки изображения.

Сжатие без потерь: Техника сжатия данных, которая точно восстанавливает первоначальные данные, наподобие *LZW* сжатия.

Сжатие преобразованием: Техника сжатия данных, основанная на назначении высоким частотам меньшего количества битов. Наилучшим примером является *JPEG*.

Сжатие с потерями: Метод сжатия данных, который восстанавливает только приближение к первоначальным данным. Это позволяет достигнуть более высоких коэффициентов сжатия. Пример - *JPEG*.

Сигмоида: "S" образная кривая, используемая в нейронных сетях.

Сигнал: Описание того, как один параметр изменяется в зависимости от другого параметра. Пример: *напряжение*, изменяющееся во *времени*.

Синк функция: Формально определяется соотношением: $sinc(a) = sin(\pi a)/\pi a$. Член π, часто спрятан в других переменных, приводя соотношение к общей форме: $sin(x)/x$. Важна, поскольку она является преобразованием Фурье прямоугольного импульса.

Синтез: Обратное преобразование Фурье, вычисляющее временную область по частотной области. Смотри для сравнения *анализ*.

Синусоидальное качество: Важное свойство линейных систем. Синусоидальный сигнал на входе может дать на выходе только синусоидальный сигнал; амплитуда и фаза могут изменяться, но частота останется такой же.

Система: Любой процесс, дающий выходной сигнал в ответ на входной сигнал.

Систематическая ошибка: Повторяемые от опыта к опыту ошибки в измерении или предсказании. Систематические ошибки определяют *погрешность*. Смотри для сравнения *случайная ошибка*.

Система чириканья: Используется в радио- и гидролокаторах. Перед передачей импульс превращается в более длительный сигнал, а после приема обратно сжимается в импульс.

Системы без памяти: Системы, где текущее значение на выходе зависит только от текущего значения на входе, и не зависит от предыдущих значений.

Случайная ошибка: Не повторяющиеся от опыта к опыту ошибки в измерении или предсказании. Определяет *точность*. Смотри для сравнения *систематическая ошибка*.

Совмещение имен: Процесс, при котором в результате дискретизации или другого нелинейного процесса синусоида изменяется от одной частоты до другой (стробоскопический эффект – прим. ред. перев.). Обычно приводит к потере информации в сигнале.

Совмещение имен во временной области: Совмещение имен, которое происходит во временной области в ответ на действие, предпринятое в частотной области. Примером является круговая свертка.

Совмещение имен в частотной области: Происходящее совмещение имен, случающееся в частотной области в ответ на действие, предпринятое во временной области. Примером является совмещение имен при дискретизации.

Согласованная фильтрация: Метод, используемый для определения, встречается ли и в каком месте в сигнале известный образец. Согласованная фильтрация основана на корреляции, но осуществляется при помощи свертки.

Сонар: Sound Navigation And Ranging (звуковая навигация и определение дальности). Использование звука для обнаружения подводных лодок и других подводных объектов. *Активный сонар* использует эхолокацию, в то время как *пассивный сонар* только слушает.

Сортировка перевернутых битов: Алгоритм, используемый в БПФ для достижения перекрестной декомпозиции сигнала. Выполняется посредством переворота при помощи двоичных вычислений битов номеров отсчетов слева направо.

Спектральный анализ: Осмысление сигнала, путем исследования амплитуды, частоты и фазы составляющих его синусоид. Первостепенным инструментом спектрального анализа является преобразование Фурье.

Спектрограмма: Измерение того, как изменяется во времени частотный спектр звукового сигнала. Обычно отображается в виде

изображения. Также называется *отпечатком голоса*.

Среднее значение: среднее значение сигнала или другой группы данных.

Среднеквадратичное: Используется для выражения колебаний сигнала вокруг *нуля*. Часто используется в электронике. Определяется как корень квадратный от квадрата среднего значения. Смотри для сравнения *стандартное отклонение*.

Стандартное отклонение: Способ выражения флуктуации сигнала вокруг его среднего значения. Определяется как корень квадратный из среднего квадратов отклонений, где отклонение является разницей между отсчетом и средним значением. Смотри для сравнения *среднеквадратичное*.

Статическая линейность: Относится к тому, как ведет себя линейная система тогда, когда сигналы не изменяются (т.е. они являются постоянной величиной или *статическими*). В этом случае, сигнал на выходе равен сигналу на входе, умноженному на константу.

Статистический шум: Изменения в значении сигнала, являющиеся результатом того, что он представлен конечным числом частиц, наподобие рентгеновских лучей, электронов или фотонов света. Также называется *распределением Пуассона* и *шумом Пуассона*.

Суммирование перекрытий: Метод, используемый для разбиения длинных сигналов на сегменты для обработки.

Сходимость: Термин, используемый в итеративных методах для указания того, что продвижение идет в направлении решения ("алгоритм сходится") или, что решение было достигнуто ("алгоритм сошелся").

C: Типичный язык программирования, используемый в науке технике и ЦОС. Он также поставляется в виде более совершенной версии *C++*.

CVSD: Дельта модуляция с непрерывно изменяемым наклоном, техника, используемая для преобразования голосового сигнала в непрерывный двоичный поток.

s-область: Область, определяемая с помощью преобразования Лапласа. Также называется *s-плоскость*.

SECAM: Телевизионный стандарт, используемый в Европе. Смотри для сравнения *NTSC*.

Тембр: Восприятие человеком гармоник в звуке. Смотри для сравнения *высота звука*.

Теорема о дискретизации: Если непрерывный сигнал, состоящий из частот, меньше чем f подвергается дискретизации с частотой в $2f$, вся информация, содержащаяся в непрерывном сигнале, будет присутствовать в дискретном сигнале. Часто называется теорема *Шеннона* о дискретизации или теорема *Найквиста* о дискретизации (теорема *Котельникова* – прим. перев.).

Точность: Не повторяющаяся от опыта к опыту ошибка в измерении или предсказании. Точность определена случайными ошибками. Смотри для сравнения *погрешность*.

Трансформация: Постепенное деформирование изображения от одной формы до другой. Используется для спецэффектов, наподобие человека превращающегося в оборотня.

TIFF: Типичный формат файла изображения, используемый в обработке текстов и в подобных программах. Обычно не сжат, хотя LZW сжатие является доступным. Смотри для сравнения *GIF* и *JPEG*.

Указатель: переменная, чье значение является адресом другой переменной.

Улитка: Орган в ухе, где входящий звук преобразуется в нервный сигнал.

Улиточный нерв: Нерв, передающий звуковую информацию от уха к мозгу.

Уравнивание гистограммы: Обработка изображения с помощью использования в качестве преобразования шкалы серого интегрированной гистограммы изображения. Работает посредством придания большим областям изображения более высокой контрастности, чем маленьким областям.

Усиление фронтов: Любой алгоритм обработки изображения, делающий фронты более очевидными. Называется также операцией *заострения*.

Фазорное преобразование: Способ использования комплексных чисел для нахождения частотных характеристик *R-L-C* цепей. Резисторы конденсаторы и катушки индуктивности становятся соответственно R, $-j/\omega C$ и $j\omega L$.

Фиксированная запятая: Один из двух характерных способов, с помощью которого в компьютерах хранятся числа; обычно

используется для хранения целых. Смотри для сравнения *плавающая запятая*.

Фильтр антисовмещения: Низкочастотный аналоговый фильтр, расположенный перед аналого-цифровым преобразователем. Удаляет частоты, лежащие выше половины частоты дискретизации, которые давали бы совмещение имен во время преобразования.

Фильтр Баттерворта: Отделяет одну полосу частот от другой; наиболее быстрый завал при сохранении пологости полосы пропускания; может быть аналоговым или цифровым. Также называется *максимально пологим* фильтром.

Фильтр Бесселя: оптимизированный для линейной фазы. У него почти нет никакого перерегулирования в переходной характеристике, а нарастающий и падающий фронты подобны. Используется для сглаживания сигналов, закодированных во временной области.

Фильтр Винера: Оптимальный фильтр для увеличения соотношения сигнал/шум, базируется на частотных спектрах сигнала и шума.

Фильтр на переключаемых конденсаторах: Аналоговый фильтр, использующий для замены резистора быстрые переключения. Выполнен в виде просто используемой интегральной схемы. Часто используется как фильтр антисовмещения для АЦП и восстанавливающий фильтр для ЦАП.

Фильтр нечетного порядка: Аналоговый или цифровой фильтр, имеющий нечетное число полюсов.

Фильтр скользящего среднего: Каждый отсчет в выходном сигнале является средним от многих смежных отсчетов во входном сигнале. Может быть выполнен с помощью свертки или рекурсии.

Фильтр Чебышева: Используется для отделения одной полосы частот от другой. Достигается более быстрый завал, чем у фильтра Баттерворта при допущении пульсаций в полосе пропускания. Может быть аналоговым или цифровым.

Фильтр четного порядка: Аналоговый или цифровой фильтр, имеющий четное число полюсов.

Фон Неймановская архитектура: Внутренняя организация компьютера, при которой и программа и данные находятся в одной и той же памяти; очень простая. Смотри для сравнения *Гарвардскую Архитектуру*.

Формат изображения: отношение ширины изображения к его высоте. Стандартное телевидение имеет формат изображения 4:3, в то время как кинематограф имеет формат изображения 16:9.

Формула Эйлера: Наиболее важное уравнение в комплексной математике, связывающее косинусную и синусную волны с комплексной экспонентой.

Фрикативные: Звуки речи человека, которые образуются за счет турбулентности воздушного потока как случайный шум, наподобие: *с*, *ф*, *ш*, *з*, *в* и *х*. Смотри для сравнения *звонкие*.

Функция массы вероятности: Показывает вероятность того, что *дискретная* переменная будет принимать определенное значение. Смотри для сравнения *функция плотности вероятности*.

Функция плотности вероятности: Показывает вероятность того, что *непрерывная* переменная будет принимать определенное значение.

Функция размыва линии (ФРЛ): Отклик системы изображения на тонкую линию во входном изображении.

Функция размыва точки (ФРТ): Жаргон из области обработки изображений для импульсной характеристики.

Фурье реконструкция: Один из методов, используемый в компьютерной томографии для вычисления изображения по его видам.

Хранение нулевого порядка: Термин, используемый в ЦАП для описания того, что между преобразованиями на выходе ЦАП удерживается постоянное значение аналогового сигнала, что приводит к появлению лестницы.

Целые: Целые числа: ..., *-2*, *-1*, *0*, *1*, *2*, … . Также относится к числам, хранимым в форме представления с фиксированной запятой. Смотри для сравнения *плавающая запятая*.

Центральная предельная теорема: Важная теорема в статистике. Одна из форм: сумма большого числа случайных чисел будет иметь Гауссову функцию плотности вероятности, в независимости от функции плотности вероятности индивидуальных случайных чисел.

Частота Найквиста, скорость Найквиста: Эти термины относятся к теореме о дискретизации, но разными авторами

применяются по-разному. Они могут быть использованы для обозначения четырех различных вещей: самой высокой частоты, содержащейся в сигнале, частоты вдвое большей, чем эта, частоты дискретизации или половины частоты дискретизации.

Частота среза: В аналоговых и цифровых фильтрах, частота, отделяющая полосу пропускания от полосы перехода. Часто измеряется на уровне, где амплитуда снижается до 0,707 (-3dB).

Частота среза -3dB: раздел между полосой пропускания и полосой заграждения фильтра. Определяется как частота, на которой частотная характеристика снижается на -3dB (0,707 по амплитуде).

Частотная область: Сигнал, имеющий частоту в качестве независимой переменной. Результат преобразования Фурье.

Частотная характеристика: Изменения амплитуды и фазы, которые претерпевают синусоиды при прохождении через линейную систему. Обычно выражаются как функции частоты. Часто находятся взятием от импульсной характеристики преобразования Фурье.

Чересстрочное видео: Видеосигнал, в котором отображаются четные строки изображения, а затем нечетные строки. Используется в телевидении; разработано для снижения мерцания экрана.

Четная/нечетная декомпозиция: Способ разложения сигнала на два других сигнала, один из которых обладает четной симметрией, а другой обладает нечетной симметрией.

Шум округления: Ошибка, вызванная округлением результата математических вычислений к ближайшему уровню квантования.

Шум 1/f: Вид случайного шума с увеличивающейся амплитудой на более низких частотах. Он не совсем понятен, но часто наблюдается в физических системах. Смотри для сравнения *белый шум*.

Эллиптический фильтр: Используется для отделения одной полосы частот от другой. Достигается быстрый завал при допущении пульсаций в полосе пропускания и полосе заграждения. Может использоваться, как в аналоговых, так и в цифровых устройствах.

Эрозия: Морфологическая операция. При применении к двоичным изображениям эрозия делает объекты меньше и может разбить объект на два или более кусков.

Эффект Гиббса: Когда сигнал усекается в одной области, в другой области на фронтах и углах появляется звон и перерегулирование.

Ядро: Импульсная характеристика фильтра, реализованного с помощью свертки. Также известно, как *ядро свертки* и *ядро фильтра*.

Ядро свертки: Импульсный отклик фильтра, реализованного с помощью свертки. Также известно, как *ядро фильтра* и *ядро*.

Ядро фильтра: Импульсная характеристика фильтра, реализованного с помощью свертки. Известно также, как *ядро свертки* или просто *ядро*.

Язык высокого уровня: Язык программирования наподобие C, Бейсика и Фортрана.

Яма: Сокращение от *потенциальная яма*; чувствительная к свету область в приборе с зарядовой связью.

Яркость: Общая освещенность или затемненность изображения. Смотри для сравнения *контрастность*.

Алфавитный указатель