



# Оглавление

<b>Введение .....</b>	<b>8</b>
Предисловие к русскому изданию .....	8
Для кого эта книга? .....	9
Работа с кодом .....	9
Условные обозначения в этой книге .....	11
Список корреспондентов .....	12
<b>Глава 1. Звуки и сигналы .....</b>	<b>14</b>
Периодические сигналы .....	14
Разложение в спектр .....	16
Сигналы .....	18
Чтение и запись сигналов .....	20
Спектры .....	20
Объекты Wave .....	21
Объекты Signal .....	22
Упражнения .....	24
<b>Глава 2. Гармоники .....</b>	<b>25</b>
Треугольный сигнал .....	25
Прямоугольный сигнал .....	27
Биения (алиасинг) .....	29
Вычисление спектра .....	31
Упражнения .....	33
<b>Глава 3. Аперiodические сигналы .....</b>	<b>35</b>
Линейный чирп .....	35
Экспоненциальный чирп .....	37
Спектр чирпа .....	38
Спектрограмма .....	39
Предел Габора .....	40
Утечка .....	41
Окна .....	42
Реализация спектрограмм .....	44

Упражнения.....	46
<b>Глава 4. Шум.....</b>	<b>48</b>
Некоррелированный шум .....	48
Интегральный спектр .....	51
Броуновский шум .....	52
Розовый шум.....	55
Гауссов шум .....	57
Упражнения.....	59
<b>Глава 5. Автокорреляция.....</b>	<b>61</b>
Корреляция .....	61
Последовательная корреляция .....	64
Автокорреляция .....	65
Автокорреляция периодических сигналов .....	66
Корреляция как скалярное произведение .....	70
Использование NumPy .....	71
Упражнения.....	72
<b>Глава 6. Дискретное косинусное преобразование ...</b>	<b>74</b>
Синтез .....	74
Синтез с массивами .....	75
Анализ .....	77
Ортогональные матрицы .....	78
ДКП-IV.....	80
Обратное ДКП.....	81
Класс Dct.....	82
Упражнения.....	83
<b>Глава 7. Дискретное преобразование Фурье .....</b>	<b>85</b>
Комплексные экспоненты.....	85
Комплексные сигналы .....	87
Задача синтеза .....	88
Синтез с матрицами.....	90
Задача анализа .....	92
Эффективный анализ .....	92
ДПФ.....	93
Периодичность ДПФ .....	95
ДПФ реальных сигналов.....	96
Упражнения.....	98

<b>Глава 8. Фильтрация и свертка .....</b>	<b>99</b>
Сглаживание .....	99
Свертка .....	102
Частотная область .....	103
Теорема о свертке .....	104
Гауссов фильтр .....	106
Эффективная свертка .....	108
Эффективная автокорреляция .....	109
Упражнения .....	111
<b>Глава 9. Дифференцирование и интегрирование ...</b>	<b>112</b>
Конечные разности .....	112
Частотная область .....	113
Дифференцирование .....	115
Интегрирование .....	117
Нарастающая сумма .....	119
Интегрирование шума .....	122
Упражнения .....	123
<b>Глава 10. Линейные стационарные системы .....</b>	<b>125</b>
Сигналы и системы .....	125
Окна и фильтры .....	127
Акустическая характеристика .....	128
Системы и свертка .....	131
Доказательство теоремы о свертке .....	134
Упражнения .....	136
<b>Глава 11. Модуляция и выборка (квантование) ....</b>	<b>138</b>
Свертка с импульсами .....	138
Амплитудная модуляция .....	139
Выборка .....	142
Биения .....	145
Интерполяция .....	148
Итог .....	150
Упражнения .....	151
<b>Предметный указатель .....</b>	<b>153</b>
<b>Об авторе .....</b>	<b>159</b>
<b>Об обложке .....</b>	<b>159</b>



# Введение

Обработка сигналов – одна из любимых тем автора. Она полезна во многих областях науки и техники – и, если постичь основы, то будет легче разобраться во многих вещах, видимых в мире, и тем более в слышимых.

Но с обработкой сигналов практически никто, кроме инженеров, не знаком. Проблема в том, что большинство книг (и использующие их курсы) представляют материал «снизу вверх», начиная с математических абстракций, например с фазоров. И эти книги обычно теоретические, с немногочисленными и малополезными примерами.

Цель этой книги – научить тех, кто умеет программировать, использовать свое умение для познания нового, да еще и с удовольствием.

С таким подходом, основанным на программировании, можно сразу понять самые важные идеи. Прочитав первую главу, вы научитесь анализировать звукозаписи или иные сигналы, а также генерировать новые звуки. В каждой главе вводятся новый прием и приложение, работающее с реальными сигналами. На каждом этапе сначала изучается применение приема, а затем – его работа.

Такой подход более практичен, и, вероятно, интересен.

## Предисловие к русскому изданию

Эта книга чрезвычайно необычна. Она аканонична во всем. «Оголтелый» прагматизм автора, вероятно, может вызвать легкий обморок у специалистов с академическим образованием.

И именно этим книга хороша! Автор использует практически все методы и алгоритмы, известные на сегодняшний день, причем в ясной и прозрачной манере. Читатель избавлен от скучной теории, от мрачных тенет аналитической геометрии, линейной алгебры и высшего анализа.

Автор сразу вооружает читателя готовыми инструментами, которые пригодятся в самых разных областях науки, техники и предпринимательства, где нужна и полезна цифровая обработка сигналов.

У книги есть один недостаток – не введено понятие аналитического сигнала. Но большинство читателей от этого только выигрывает! Ведь, изучив материал книги, им будет заметно проще перейти от обработки записи звука скрипки к обработке нестационарных случайных сигналов – от радиолокаторов или гидроакустических станций до ультразвуковых диагностических приборов или масс-спектрометров и газоанализаторов.

## Для кого эта книга?

Примеры и соответствующий код для этой книги написаны на Python. Надо знать основы Python и основы объектно-ориентированного программирования – хотя бы уметь применять объекты, пусть и не свои собственные.

Если язык Python вам еще не знаком, можно начать с книги Аллена Б. Дауни (Allen B. Downey) *Think Python* (это введение в Python для тех, кто никогда ранее не программировал) или с книги Марка Луца (Mark Lutz) *Learning Python* – она удобнее для людей с опытом программирования.

В книге широко использованы NumPy и SciPy, и все используемые в них функции и структуры данных подробно объясняются. А если они уже известны, то это хорошо.

Предполагается, что читатель знаком с основами математики, в том числе с комплексными числами. Весь матанализ помнить не нужно; достаточно понятий об интегрировании и дифференцировании. По мере использования линейной алгебры все будет подробно объяснено.

## Работа с кодом

Код и образцы звуков, используемые в этой книге, доступны в репозитории GitHub: <https://github.com/AllenDowney/ThinkDSP>. Для тех, кто не знаком с Git и GitHub, поясним: Git – это система управления версиями, в которой можно работать с входящими в проект файлами. Коллекция файлов под контролем Git называется «репозиторий». GitHub – это хостинг для хранения репозитория Git, и у него удобный веб-интерфейс.

На домашней странице репозитория автора в GitHub с кодом можно работать несколькими способами:

- создать копию репозитория на GitHub, нажав кнопку «вилка» (fork). Если аккаунта на GitHub еще нет, его надо завести. По-

сле разветвления (копии) появится новый, личный репозиторий GitHub; в нем удобнее отслеживать код при работе с этой книгой. Затем можно клонировать свой репозиторий, скопировав файлы на компьютер;

- клонировать репозиторий автора. Для этого аккаунт GitHub не нужен, но записать изменения обратно в GitHub уже нельзя;
- скачать файлы в архиве ZIP, нажав кнопку в правом нижнем углу страницы GitHub (если использовать Git нежелательно).

Весь код в книге работает без трансляции и в Python 2, и в Python 3.

Эта книга разработана с помощью Anaconda компании Continuum Analytics (Continuum Analytics, компания) – свободного дистрибутива Python, включающего все нужные для запуска кода пакеты (и многое другое). Устанавливается Anaconda легко. По умолчанию эта платформа ставится на уровне пользователя, а не на системном, так что права администратора не нужны. Она поддерживает и Python 2, и Python 3. Anaconda можно скачать из <http://continuum.io/downloads>.

Если использовать Anaconda нежелательно, то понадобятся следующие пакеты:

- NumPy для работы с числами (<http://www.numpy.org>);
- SciPy для научных вычислений (<http://www.scipy.org>);
- Matplotlib для визуализации (<http://matplotlib.org>).

Эти пакеты часто используются, но они не входят ни в один дистрибутив Python, и в некоторых средах их бывает трудно установить. Если будут проблемы с их установкой, рекомендуется прибегнуть к Anaconda или одному из других дистрибутивов Python, включающих эти пакеты.

Большинство упражнений – это Python-скрипты, но в некоторых также используется «блокнот Jupyter». Ознакомиться с Jupyter и изучить его можно на <http://jupyter.org>.

Есть три способа работы с блокнотами Jupyter:

1. Запустить Jupyter на своем компьютере.

Если Anaconda установлена, то и Jupyter, вероятно, установлен по умолчанию. Проверьте это, запустив сервер из командной строки:

```
$ jupyter notebook
```

Если он не установлен, его можно установить в Anaconda:

```
$ conda install jupyter
```

При запуске сервера он запустит веб-браузер по умолчанию или же создаст новую вкладку в уже открытом окне браузера.

## 2. Запустить Jupyter в Binder.

Binder – это служба, запускающая Jupyter на виртуальной машине. По ссылке <http://mybinder.org/repo/AllenDowney/ThinkDSP> открывается домашняя страница Jupyter с блокнотами для этой книги, а также дополнительными данными и скриптами.

Скрипты можно запускать и изменять их для запуска собственного кода, но виртуальная машина будет временной. Любые изменения пропадут, если пауза в работе с ней продлится более часа.

## 3. Просмотреть блокноты на nbviewer.

В книге также приводятся ссылки на nbviewer, с его помощью можно просматривать код и результаты. Это удобно для чтения блокнотов и прослушивания примеров, но ни изменить код, ни запустить его, ни использовать интерактивные виджеты нельзя.

Удачи в работе!

# Условные обозначения в этой книге

В этой книге используются следующие типографские соглашения:

### *Курсив*

Выделение новых терминов и особо значимых фрагментов текста.

### **Жирный шрифт**

Используется для обозначения сочетаний клавиш, элементов программного интерфейса.

### Моноширинный шрифт

Используется для обозначения URL-адресов и адресов электронной почты, листингов программ, имен файлов, расширений файлов, а также элементов программ, таких как имена переменных и функций, типов данных, инструкций и ключевых слов.

### **Моноширинный полужирный шрифт**

Отображает набираемые пользователем команды или иной текст.

## Список корреспондентов

Предложения или замечания по поводу книги направляйте на [downey@allendowney.com](mailto:downey@allendowney.com). Если на основе ваших сообщений будут сделаны изменения, то автора сообщения добавят в список корреспондентов (при отсутствии специальных условий).

Желательно указывать хотя бы часть предложения, в котором обнаружена ошибка, – так ее легче найти. Можно сослаться и на номер страницы или название раздела, хотя это усложняет поиск. Спасибо за понимание!

- Прежде чем автор приступил к написанию этой книги, идеи, которые легли в ее основу, развивались в общении с Булосом Хэрбом (Boulos Harb) из Google и с Аурелио Рамосом (Aurelio Ramos), ранее работавшим в Harmonix Music Systems.
- Во время осеннего семестра 2013 года Натан Линц (Nathan Lintz) и Ян Дэниэр (Ian Daniher) работали со автором над независимым учебным проектом и помогли с первым изданием этой книги.
- На DSP-форуме Reddit пользователь под ником Ramjetsoundwave помог решить проблему с реализацией броуновского шума. Другой участник форума, andodli, нашел опечатку.
- Весной 2015 года этот материал преподавался совместно с проф. Оскаром Мур-Мирандой (Oscar Mur-Miranda) и проф. Сиддхартаном Говиндасами (Siddharta Govindasamy). Оба внесли много предложений и исправлений.
- Сайлас Гигер (Silas Gyger) исправил арифметическую ошибку.
- Джузеппе Мазетти (Giuseppe Masetti) прислал ряд полезных замечаний.

Особая благодарность техническим рецензентам – Эрику Петерсу (Eric Peters), Брюсу Ливенсу (Bruce Levens) и Джону Винсенту (John Vincent) – за многие полезные замечания, разъяснения и исправления.

Также благодарность Freesound, источнику многих звуков, используемых в книге, и пользователям Freesound, предоставившим их. Некоторые из них включены в репозиторий GitHub этой книги, причем использованы оригинальные имена файлов, так что источники легко найти.

К сожалению, большинство пользователей Freesound не указывает своих имен, поэтому здесь можно привести только их ники. Образцы



звучков, используемые в книге, были предоставлены пользователями Freesound – iluppai, wcf10, thirsk, docquesting, kleeb, landup, zippi1, themusicalnomad, bcjordan, rockwehrmann, marcgascon7, jcveliz.

Спасибо им всем!



# Глава 1.

## Звуки и сигналы

*Сигнал* представляет собой изменяющуюся во времени величину. Это определение довольно абстрактно, так что начнем с конкретного примера – звука. *Звук* – это изменение давления воздуха. *Звуковой сигнал* – изменения давления воздуха во времени.

Микрофон – это устройство, воспринимающее такие изменения и генерирующее соответствующий звуку электрический сигнал. Динамик – это устройство, принимающее электрический сигнал и производящее звук. Микрофоны и динамики называются *преобразователями*, так как они преобразуют сигналы из одного вида в другой.

Эта книга посвящена обработке сигналов, то есть процессам их синтеза, преобразования и анализа. Для изучения наиболее удобны звуковые сигналы, но все рассматриваемые методы применимы и к электронным сигналам, и к механическим вибрациям, и к сигналам вообще.

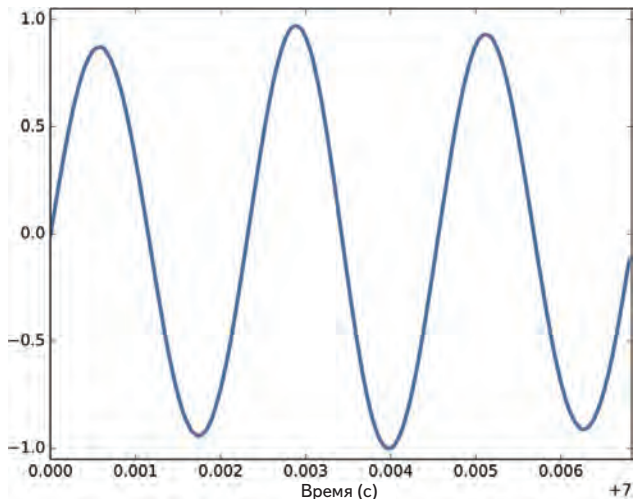
Они также применимы к сигналам, изменяющимся не во времени, а в пространстве, как, скажем, изменение высот профиля местности. Методы применимы и к многомерным сигналам, таким как изображения, то есть сигналам, изменяющимся в двумерном пространстве. Так, фильм – это сигнал, меняющийся в двумерном пространстве и во времени.

Начнем с простого одномерного звука.

Код для главы 1 находится в репозитории этой книги, в блокноте `chap01.ipynb` (см. раздел «Работа с кодом» на стр. 9). Его также можно посмотреть на <http://tinyurl.com/thinkdsp01>.

## Периодические сигналы

Начнем с *периодических сигналов*, то есть с сигналов, повторяющихся через некоторый период времени. Например, колокол после удара вибрирует, издавая звук. Если записать этот звук и построить график преобразованного сигнала, получится кривая, показанная на рис. 1.1.



**Рис. 1.1.** Сегмент звукозаписи колокола

Этот сигнал напоминает *синусоиду*, то есть он похож на тригонометрическую функцию синус.

Видно, что это периодический сигнал. Интервал выбран так, чтобы показать три полных повтора, или *цикла*. Длительность каждого цикла, называемая *периодом*, составляет около 2,3 мс.

*Частота* сигнала – число циклов в секунду, она обратна периоду. За единицу частоты приняты циклы в секунду, или *герц*, сокращенно «Гц». (Строго говоря, число циклов – безразмерная величина, поэтому герц – это «циклов в секунду».)

Частота этого сигнала около 439 Гц – чуть ниже 440 Гц, стандартной высоты тона для оркестровой музыки. Музыкальное имя этой ноты – А (ля), или, точнее, А4. В американской системе нотации числовой суффикс указывает, из какой октавы нота. А4 – это А выше среднего С. А5 – на октаву выше. Подробнее см. [https://ru.wikipedia.org/wiki/Американская\\_система\\_нотации](https://ru.wikipedia.org/wiki/Американская_система_нотации).

Колебания зубцов *камертона* – это форма простого гармонического движения, поэтому сигнал похож на синусоиду. Большинство музыкальных инструментов производят периодические сигналы, но форма этих сигналов не синусоидальная. Например, на рис. 1.2 показан сегмент записи звука скрипки, на которой играют 3-ю часть струнного квартета № 5 ми-мажор Боккерини.

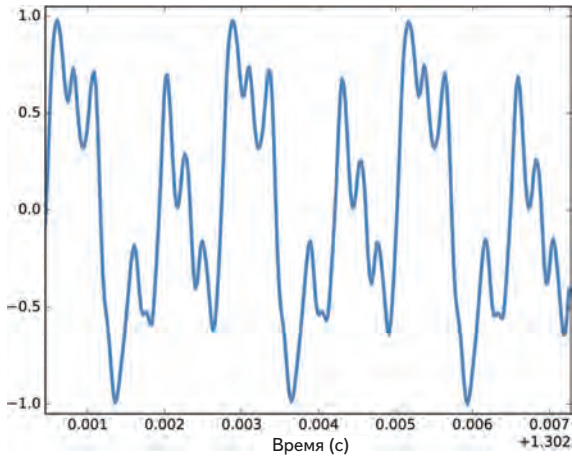


Рис. 1.2. Сегмент записи звука скрипки

Видно, что сигнал периодический, но форма сигнала заметно сложнее. В англоязычной литературе форму периодического сигнала называют *waveform*. В русском научно-техническом лексиконе такого понятия нет. Ведь сигнал – он и есть сигнал. И он может иметь самую разную форму.

Сигналы большинства музыкальных инструментов сложнее синусоиды. От формы сигнала зависит музыкальный *тембр*, определяющий восприятие качества звука. Люди обычно воспринимают сложные сигналы как богатые, теплые, более интересные, чем синусоидальные.

## Разложение в спектр

Самая важная тема этой книги – *разложение в спектр*. Суть ее в том, что любой сигнал можно представить суммой синусоид с разными частотами.

Самая важная математическая идея в этой книге – *дискретное преобразование Фурье* (ДПФ). Оно берет сигнал и выдает его спектр. *Спектр* – это набор синусоид, составляющих сигнал.

А самый важный алгоритм в этой книге – *быстрое преобразование Фурье* (БПФ) – эффективнейший способ вычисления ДПФ.

Например, на рис. 1.3 показан спектр записи звука скрипки по рис. 1.2. Ось  $x$  – область частот, составляющих сигнал. Ось  $y$  показывает размахи, или *амплитуды*, составляющих спектра.

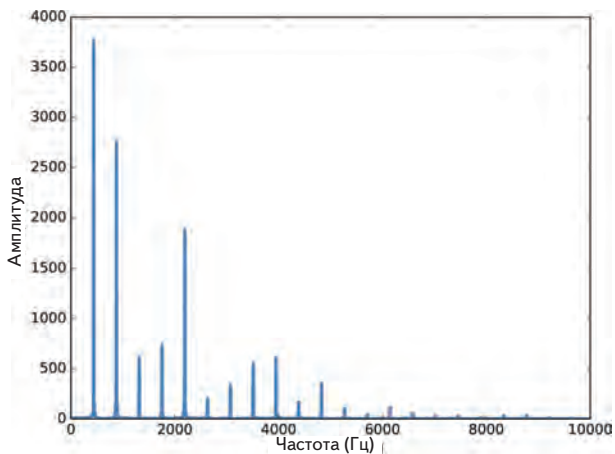


Рис. 1.3. Спектр сегмента записи звука скрипки

Компонента с самой низкой частотой называется *основной частотой*. Основная частота этого сигнала – около 440 Гц (на самом деле чуть ниже, или «плосче»).

В данном сигнале у основной частоты самая большая амплитуда, так что это еще и *доминирующая частота*. Обычно воспринимаемая высота звука определяется основной частотой, даже если она и не доминирующая.

Другие пики в спектре находятся на частотах 880, 1320, 1760 и 2200, и это целые кратные основной. Эти компоненты называются *гармониками*, потому что они музыкально гармоничны с основной:

- 880 – частота А5, на одну октаву выше основной. *Октава* – это удвоение частоты;
- 1320 – приблизительно Е6, которая выше А5 ровно на квинту. Про музыкальные интервалы и про чистую квинту можно прочесть здесь: [https://ru.wikipedia.org/wiki/Интервал\\_\(музыка\)](https://ru.wikipedia.org/wiki/Интервал_(музыка));
- 1760 – это А6, на две октавы выше основной;
- 2200 – примерно С#7, на большую терцию выше А6.

Эти гармоники – ноты аккорда А-мажор, хоть они и не все в одной октаве. Некоторые из них не совсем точны, поскольку ноты, составляющие «западную» музыку, были нормализованы для *равномерной темперации* (см. [https://ru.wikipedia.org/wiki/Равномерно\\_темперированный\\_строй](https://ru.wikipedia.org/wiki/Равномерно_темперированный_строй)).

Зная гармоники и их амплитуды, сигнал можно восстановить сложением синусоид. Далее будет показано, как.

## Сигналы

Автор написал Python-модуль, `thinkdsp.py`, содержащий классы и функции для работы с сигналами и спектрами. Он помещен в репозиторий этой книги (см. раздел «Работа с кодом» на стр. 9).

Для представления сигналов в `thinkdsp` есть класс, называемый `signal`. Это родительский класс для нескольких типов сигналов, включая `Sinusoid`, представляющий сигналы синус и косинус.

`thinkdsp` предоставляет функции для создания сигналов синус и косинус:

```
cos_sig = thinkdsp.CosSignal(freq=440, amp=1.0, offset=0)
sin_sig = thinkdsp.SinSignal(freq=880, amp=0.5, offset=0)
```

`freq` – частота в герцах, `amp` – амплитуда в относительных единицах, причем 1,0 определяется как наибольшая возможная записываемая или воспроизводимая амплитуда.

`offset` – это *фазовый сдвиг*, или просто *фаза*, в радианах. Фазовый сдвиг определяет, в каком месте периода начинается сигнал. Например, сигнал синус с параметром `offset = 0` начинается в  $\sin 0$ , то есть в «0». При `offset = Pi/2` начало в  $\sin \pi/2$ , то есть в «1».

У сигналов есть метод `__add__`, так что можно использовать оператор `+` при их сложении:

```
mix = sin_sig + cos_sig
```

Результатом будет `SumSignal`, представляющий собой сумму двух или более сигналов.

Обычно `signal` – это Python-представление математической функции. Большинство сигналов определены для всех значений  $t$  от минус бесконечности до бесконечности.

Более ничего с `signal` не сделать, если не начать его обработку. В этом контексте «обработка» означает последовательность моментов времени, `ts`, и получение соответствующего им значения сигнала, `ys`. Представлены `ts` и `ys` в виде массивов NumPy, инкапсулированных в объект, называемый `wave`.

`wave` – это сигнал, обрабатываемый в последовательности моментов времени. Каждый момент времени называется *кадром* (`frame`) – термин заимствован из кино и видео. Результат измерения называется

ся *выборкой*, или *отсчетом*, хотя понятия «кадр» и «выборка» иногда взаимозаменяемы.

`signal` дает `make_wave`, возвращающий новый объект `wave`:

```
wave = mix.make_wave(duration=0.5, start=0, framerate=11025)
```

`duration` – длина `wave` в секундах.

`start` – время старта в секундах.

`framerate` – число (целое) кадров в секунду, а также число выборок в секунду.

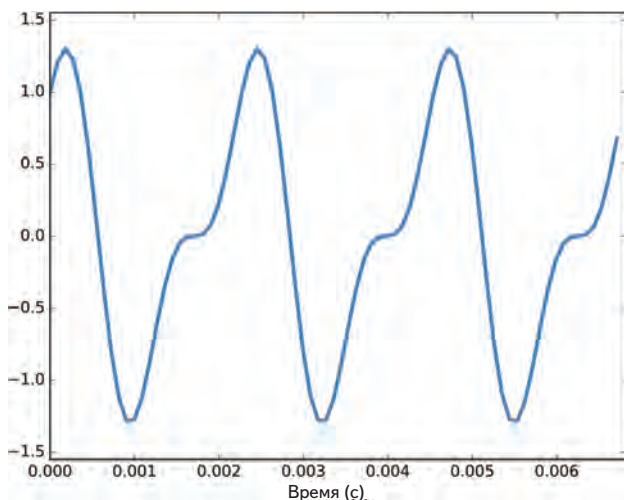
11 025 кадров в секунду – одна из многих частот выборки, обычно используемых в звуковых файлах разных форматов, включая Waveform Audio File (WAV) и MP3.

В этом примере обрабатывается сигнал от  $t = 0$  до  $t = 0,5$  в 5513 кадрах с равным интервалом (поскольку 5513 – это половина от 11 025). Время между кадрами, или *шаг*, составляет  $1/11\,025$  секунд (около 91 мкс).

`wave` поддерживает метод `plot`, использующий `pyplot`. Сигнал можно вывести на печать вот так:

```
wave.plot()  
pyplot.show()
```

`pyplot` – это часть `matplotlib`; он включен во многие дистрибутивы Python, и его всегда можно доустановить.



**Рис. 1.4.** Сегмент сигнала из смеси двух синусоид

При `freq = 440` за 0,5 секунды пройдет 220 периодов, так что при печати этот участок будет представлен сплошной заливкой. Меньшее число периодов можно вывести, используя `segment` – он копирует некий сегмент `wave` и возвращает новый вариант:

```
period = mix.period
segment = wave.segment(start=0, duration=period*3)
```

`period` – это свойство `signal`; он возвращает период в секундах. `start` и `duration` выражены в секундах. В этом примере из `mix` копируются первые три периода. Результат – объект `wave`.

На графике `segment` выглядит как на рис. 1.4. В этом сигнале две частотных компоненты; он сложнее сигнала камертона, но проще скрипичного.

## Чтение и запись сигналов

`thinkdsp` предоставляет `read_wave`, читающий WAV-файл и возвращающий `wave`:

```
violin_wave = thinkdsp.read_wave('input.wav')
```

А `wave` поддерживает `write`, записывающий WAV-файл:

```
wave.write(filename='output.wav')
```

Прослушать `wave` можно на любом мультимедиа-проигрывателе, воспроизводящем WAV-файлы. На Unix-системах это `aplay`, простой и надежный, включенный во многие дистрибутивы Linux.

В `thinkdsp` также есть `play_wave`, запускающий мультимедийный проигрыватель как подпроцесс:

```
thinkdsp.play_wave(filename='output.wav', player='aplay')
```

По умолчанию он использует `aplay`, но можно указать и другой проигрыватель.

## Спектры

`wave` поддерживает `make_spectrum`, возвращающий спектр – `spectrum`:

```
spectrum = wave.make_spectrum()
```

А `Spectrum` поддерживает `plot`:



```
spectrum.plot()
thinkplot.show()
```

`thinkplot` – написанный автором модуль, и это обертка для некоторых функций в `pyplot`. Он есть в репозитории этой книги (см. раздел «Работа с кодом» на стр. 9).

`Spectrum` предоставляет три метода, изменяющих спектр:

- `low_pass` применяет фильтр нижних частот (ФНЧ), то есть компоненты выше частоты среза ослабляются на некую величину.
- `high_pass` применяет фильтр верхних частот (ФВЧ), то есть ослабляются компоненты ниже частоты среза.
- `band_stop` применяет полосно-заграждающий фильтр (ФПЗ); он ослабляет компоненты в полосе частот между двумя частотами среза.

В этом примере все частоты выше 600 ослабляются на 99%:

```
spectrum.low_pass(cutoff=600, factor=0.01)
```

Фильтр НЧ удаляет «яркие» высокочастотные звуки, и результат звучит глуше, «темнее». Услышать это можно, если преобразовать `Spectrum` обратно в `wave` и воспроизвести:

```
wave = spectrum.make_wave()
wave.play('temp.wav')
```

Метод `play` записывает `wave` в файл, а затем воспроизводит его. В блокноте Jupyter можно использовать `make_audio`, создающий аудиовиджет для проигрывания звука.

## Объекты Wave

В `thinkdsp.py` нет ничего сложного. Большинство функций в нем – лишь тонкие обертки на функциях из NumPy и SciPy.

Основные классы в `thinkdsp` – это `signal`, `wave` и `Spectrum`. Имея `signal`, можно сделать `wave`. Имея `wave`, можно сделать `Spectrum`, и наоборот. Эти соотношения показаны на рис. 1.5.



Рис. 1.5. Соотношения классов в `thinkdsp`

У объекта `wave` есть три атрибута: `ys` – NumPy-массив, содержащий значения сигнала; `ts` – массив моментов выборки или преобразования сигнала; `framerate` – число выборок в единицу времени. За единицу времени обычно принимаются секунды, хотя допустимы и другие варианты. В одном из примеров это будут дни.

`wave` также имеет три свойства, доступные только для чтения: `start`, `end` и `duration`. Если изменить `ts`, эти свойства изменятся соответственно.

Изменить сигнал можно, явно изменив `ts` и `ys`. Например:

```
wave.ys *= 2
wave.ts += 1
```

Первая строка вдвое увеличивает размах, делая сигнал громче. Вторая строка сдвигает сигнал во времени, замедляя начало на 1 секунду.

Но `wave` дает методы, выполняющие множество общих операций. Например, можно записать эти же два преобразования так:

```
wave.scale(2) wave.shift(1)
```

Эти и другие методы описаны на веб-странице <http://greenteapress.com/thinkdsp.html>.

## Объекты Signal

`signal` – это родительский класс, предоставляющий функции, общие для всех типов сигналов, таких как `make_wave`. Дочерние классы наследуют эти методы и дают `evaluate`, оценивающий сигнал в заданные моменты времени.

Например `Sinusoid` – это дочерний класс `signal` с таким определением:

```
class Sinusoid(signal):
    def __init__(self, freq=440, amp=1.0, offset=0, func=np.sin):
        signal.__init__(self)
        self.freq = freq
        self.amp = amp
        self.offset = offset
        self.func = func
```

Параметрами `__init__` будут:

`freq`

Частота в циклах в секунду, или герцах.

`amp`

Амплитуда. Единицы амплитуды произвольны: обычно их выбирают так, что 1,0 соответствует максимуму входного уровня с микрофона или максимальному уровню на выходе.

`offset`

Указывает, в какой части своего периода сигнал начинается; `offset` указывается в радианах.

`func`

Функция Python, используемая для оценки сигнала в определенный момент времени. Обычно это `np.sin` или `np.cos`, то есть синусоидальный или косинусоидальный сигнал.

Как и многие `init`-методы, этот просто «откладывает» параметры «на потом».

`signal` дает `make_wave` следующим образом:

```
def make_wave(self, duration=1, start=0, framerate=11025):
    n = round(duration * framerate)
    ts = start + np.arange(n) / framerate
    ys = self.evaluate(ts)
    return wave(ys, ts, framerate=framerate)
```

`start` и `duration` – время начала и длительность в секундах.

`framerate` – число кадров (выборок) в секунду.

`n` – число выборок, а `ts` – массив времен выборки NumPy.

Для расчета `ys` сигнал `make_wave` вызывает функцию `evaluate`, получаемую из `Sinusoid`:

```
def evaluate(self, ts):
    phases = PI2 * Self.freq * ts + self.offset
    ys = self.amp * self.func(phases)
    return ys
```

Раскроем ее до одного шага по времени:

1. `Self.freq` – частота в циклах в секунду, а каждый элемент `ts` дает время в секундах, поэтому их произведение будет числом циклов от начала.
2. `PI2` – константа, хранящая  $2\pi$ . Умножение на `PI2` преобразует циклы в фазу. Можно рассматривать фазу как «число радианов от начала». Каждый цикл равен  $2\pi$  радиан.
3. `self.offset` – это фаза в момент  $t = 0$ . На графике это выглядит как сдвиг сигнала по оси времени влево или вправо.
4. Если `self.func` есть `np.si` или `np.cos`, то результатом будет значение в интервале между  $-1$  и  $+1$ .

- Умножение на `self.amp` дает сигнал, меняющийся от `-self.amp` до `+self.amp`.

Математически `evaluate` записывается следующим образом:

$$y = A \cos(2\pi ft + \phi_0)$$

Здесь  $A$  – амплитуда,  $f$  – частота,  $t$  – время и  $\phi_0$  – начальная фаза. Может показаться, что кода слишком много для вычисления одного простого выражения, но далее будет видно, что на этом коде основана обработка всех видов сигналов, а не только синусоиды.

## Упражнения

Для нижеследующих упражнений полезно скачать код для этой книги, следуя инструкциям в разделе «Работа с кодом» на стр. 9. Решения этих упражнений находятся в блокноте `chap01soln.ipynb`.

### Упражнение 1.1

Для Jupyter надо загрузить `chap01.ipynb`, прочитать пояснения и запустить примеры. Этот блокнот можно также просмотреть на веб-странице <http://tinyurl.com/thinkdsp01>.

### Упражнение 1.2

Скачайте с сайта <http://freesound.org> образец звука, включающий музыку, речь или иные звуки, имеющие четко выраженную высоту. Выделите примерно полусекундный сегмент, в котором высота постоянна. Вычислите и распечатайте спектр выбранного сегмента. Как связаны тембр звука и гармоническая структура, видимая в спектре?

Используйте `high_pass`, `low_pass` и `band_stop` для фильтрации тех или иных гармоник. Затем преобразуйте спектры обратно в сигнал и прослушайте его. Как звук соотносится с изменениями, сделанными в спектре?

### Упражнение 1.3

Создайте сложный сигнал из объектов `SinSignal` и `CosSignal`, суммируя их. Обработайте сигнал для получения `wave` и прослушайте его. Вычислите `spectrum` и распечатайте. Что произойдет при добавлении частотных компонент, не кратных основным?

### Упражнение 1.4

Напишите функцию `stretch`, берущую `wave` и коэффициент изменения. Она должна ускорять или замедлять сигнал изменением `ts` и `framerate`. Подсказка: должно получиться всего две строки кода.