O'REILLY®

Цифровая обработка сигналов на языке Python

Аллен Б. Дауни

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

УДК 534:004.438Python
ББК 22.32с
Д21

Д21 Аллен Б. Дауни
Think DSP. Цифровая обработка сигналов на Python / пер. с англ. Бряндинский А. Э. – М.: ДМК Пресс, 2017. – 160 с.: ил.

ISBN 978-5-97060-454-0

Изучить обработку сигналов легко – достаточно знания основ математики и программирования на Python. Обычно изучение этой сложной темы начинают с теории, а в основу данной книги положены сугубо практические примеры. Уже в первой главе звук будет разложен на гармоники, которые модифицируются и создают новые звуки. Кроме того, в книге рассмотрены: периодические сигналы и их спектры; гармоническая структура простого сигнала; чирпы и иные звуки с изменяющимся во времени спектром; шумовые сигналы и естественные источники шума; дискретное косинусное преобразование (ДКП) для сжатия информации; дискретное и быстрое преобразования Фурье для спектрального анализа, а также многое другое.

Издание будет полезно всем, кто интересуется цифровой обработкой сигналов.

УДК 534:004.438Python
ББК 22.32с

ISBN 978-1-49193-845-4 (англ.)
ISBN 978-5-97060-454-0 (рус.)

Оглавление

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 просмотреть 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.

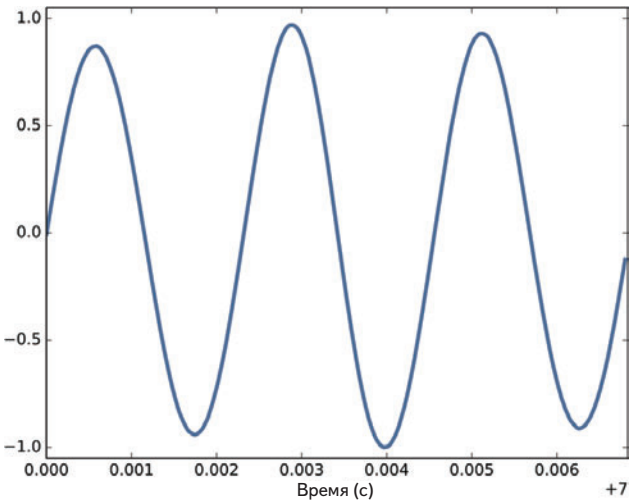

Рис. 1.1. Сегмент звукозаписи колокола

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 – A (ля), или, точнее, A4. В американской системе нотации числовой суффикс указывает, из какой октавы нота. A4 – это A выше среднего C. A5 – на октаву выше. Подробнее см. `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 квинтета № 5 ми-мажор Боккерини.

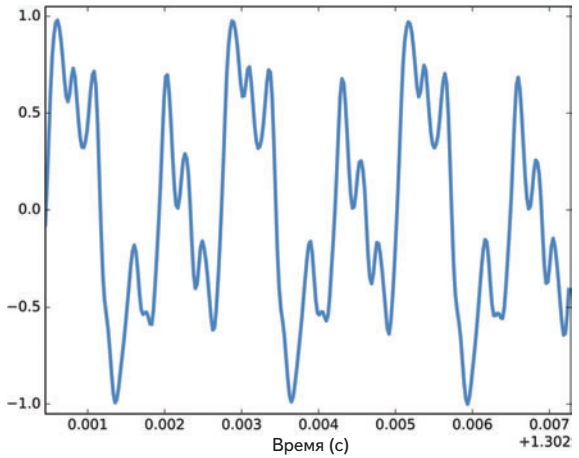

Рис. 1.2. Сегмент записи звука скрипки

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.

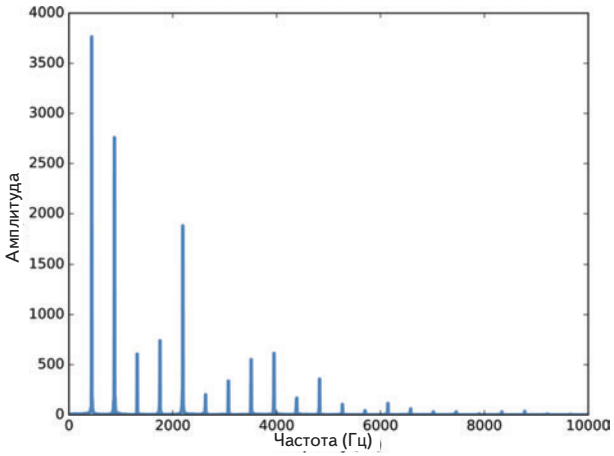

Рис. 1.3. Спектр сегмента записи звука скрипки

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

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 A5, на одну октаву выше основной. *Октава* – это удвоение частоты;
- 1320 – приблизительно E6, которая выше A5 ровно на квинту. Про музыкальные интервалы и про чистую квинту можно прочесть здесь: `https://ru.wikipedia.org/wiki/Интервал_(музыка)`;
- 1760 – это A6, на две октавы выше основной;
- 2200 – примерно C#7, на большую терцию выше A6.

Эти гармоники – ноты аккорда A-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`).

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 называет-

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.

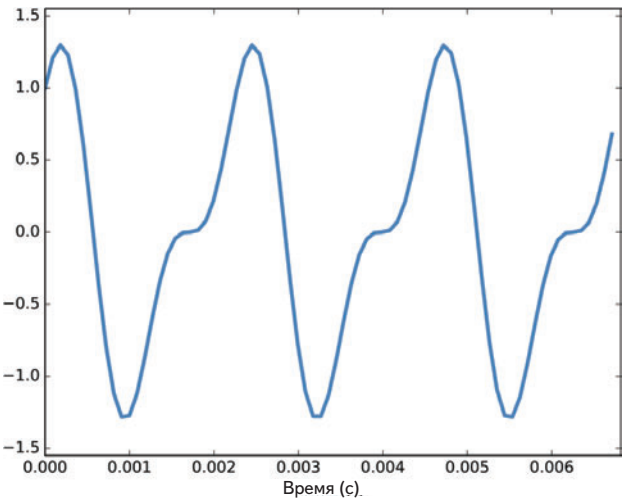

Рис. 1.4. Сегмент сигнала из смеси двух синусоид

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
```

Раскрутим 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 2π. Умножение на `PI2` преобразует циклы в фазу. Можно рассматривать фазу как «число радианов от начала». Каждый цикл равен 2π 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.

5. 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 ϕ_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 вебстранице `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.