АЛЕКСАНДР ИВАНОВ

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ НА PYTHON:

ОТ ИНЖЕНЕРА К РАЗРАБОТЧИКУ

- ОТ ТЕОРИИ К РАБОТАЮЩЕМУ КОДУ
- ВИЗУАЛИЗАЦИИ, КОТОРЫЕ ОБЪЯСНЯЮТ
- PYTHON, NUMPY, SCIPY, MATPLOTLIB
- ФИЛЬТРЫ, СПЕКТР, АДАПТИВНЫЕ АЛГОРИТМЫ
- ОТ АНАЛИЗА СИГНАЛОВ ДО СИНТЕЗА ЗВУКА

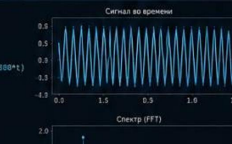
```
import numpy as np
import matplotlib.pyplot as plt

fs = 8000
t = np.arange(0, 1, 1/fs)
x = np.sin(2*np.pi*440*t) + 0.5*np.sin(2*np.pi*880*t)

X = np.fft.rfft(x)
freqs = np.fft.rfftfreq(len(x), 1/fs)

plt.figure(figsize=(10,4))
plt.plot(t, x)
plt.title('Сигнал во времени')
plt.xlabel('Время, с'); plt.ylabel('Амплитуда')
plt.grid(True); plt.tight_layout()

plt.figure(figsize=(10,4))
plt.stem(freqs, np.abs(X), use_line_collection=True)
plt.title('Спектр (FFT)')
plt.xlabel('Частота, Гц'); plt.ylabel('|X(f)|')
plt.grid(True); plt.tight_layout()
plt.show()
```



ДИСКРЕТИЗАЦИЯ И КВАНТОВАНИЕ → ПРЕОБРАЗОВАНИЕ ФУРЬЕ → ЦИФРОВЫЕ ФИЛЬТРЫ → АДАПТИВНАЯ ФИЛЬТРАЦИЯ → СИНТЕЗ И ГЕНЕРАЦИЯ

ПОНЯТЬ. УВИДЕТЬ. РЕАЛИЗОВАТЬ. СОЗДАТЬ СВОЁ.

БЕЗ СТРАХА ПЕРЕД ФОРМУЛАМИ. ТОЛЬКО ЖИВОЙ КОД, ВИЗУАЛИЗАЦИИ И ИНЖЕНЕРНАЯ ИНТУИЦИЯ.

Александр Иванов

Цифровая обработка сигналов на Python. От инженера к разработчику.

https://litres.ru/74004728
SelfPub; 2026

Аннотация

Цифровая обработка сигналов — это не магия. Это алгоритмы, которые можно понять, написать на Python и применить к реальному звуку. Эта книга доказывает это на каждой странице.

Вас ждёт сквозной путь: от дискретизации и теоремы Найквиста до синтеза звука и адаптивных фильтров. Вы напишете DFT и FFT с нуля — и увидите, почему FFT изменил мир. Вы спроектируете КИХ- и БИХ-фильтры и проверите их на своём голосе. Вы освоите свёртку и создадите реверберацию, которая превратит обычную комнату в концертный зал. Вы реализуете LMS-алгоритм для подавления нестационарного шума. Вы напишете детектор высоты голоса через кепстральный анализ. И соберёте свой синтезатор — от аддитивного до физического моделирования струны.

Для студентов, которые хотят освоить DSP без боли. Для разработчиков, которые хотят войти в мир аудио. Для создателей контента, которые устали просто нажимать кнопки и хотят управлять звуком по-настоящему.

Содержание

Цифровая обработка сигналов на Python. От инженера к разработчику.

***

Предисловие

Почему я написал эту книгу

Цифровая обработка сигналов — удивительная область. Она соединяет математику, физику и программирование. Она позволяет заглянуть внутрь сигнала и увидеть его структуру. Она даёт возможность не просто использовать готовые инструменты, а создавать свои. Но путь к этому знанию обычно лежит через толстые учебники с пугающими формулами и абстрактными доказательствами. Многие талантливые люди, которые могли бы применить цифровую обработку сигналов в своих проектах, так и не доходят до сути — они спотыкаются о математический язык и отступают.

Я написал эту книгу, потому что убеждён: сложные идеи можно объяснять простым языком. Не жертвуя глубиной. Не заменяя понимание запоминанием. А находя правильные метафоры, правильные визуализации, правильные примеры кода, которые делают абстрактное — конкретным, а непонятное — очевидным.

В основе книги лежит принцип: каждая идея должна быть не просто прочитана, а прожита. Читатель не просто узнаёт о преобразовании Фурье — он пишет его своими руками и применяет к своему голосу. Он не просто слышит о фильтрах — он создаёт фильтр, пропускает через него сигнал и сразу видит результат. Теория немедленно становится прак-

тикой. Практика немедленно подтверждает теорию.

Эта книга не отпугивает, а приглашает. Она говорит: «Это сложно, но мы разберёмся вместе». Она уважает читателя и верит в его способность понять глубокие идеи, если их правильно объяснить.

Чем эта книга отличается от других

Существуют отличные учебники по цифровой обработке сигналов. Они строги, полны и математически точны. Но они написаны для математиков и будущих учёных. Они предполагают, что читатель уже владеет высшей математикой на хорошем уровне и готов воспринимать информацию через формулы. Для большинства людей, которые хотят обрабатывать звук — музыкантов, звукорежиссёров, разработчиков аудиософта, — такой подход не работает. Формулы пугают. Абстрактные доказательства кажутся не имеющими отношения к реальности. Читатель бросает книгу на третьей странице и идёт искать туториал на YouTube.

Существуют отличные практические руководства. Они показывают, какую функцию вызвать и какие параметры передать. Они дают результат быстро. Но они не дают понимания. Когда что-то идёт не так — а в обработке звука что-то всегда идёт не так, — читатель остаётся беспомощным. Он знает рецепт, но не знает, почему рецепт работает. Он не может адаптировать его под свою ситуацию.

Эта книга идёт по третьему пути. Я объясняю теорию — но не как лекцию, а как диалог. Каждое новое понятие

вводится через вопрос: «Зачем это нужно? Какую реальную проблему это решает?» Затем я даю интуицию — метафору или аналогию из повседневной жизни, которая делает идею понятной до всяких формул. Затем я показываю формулу — но не как истину с неба, а как естественное развитие идеи. Затем мы пишем код, который реализует эту формулу, и смотрим на результат. Работает ли он? Похож ли результат на то, что мы ожидали? Если нет — почему?

Этот цикл — интуиция, теория, код, проверка — повторяется в каждой главе. Вы не просто читаете о преобразовании Фурье — вы пишете его с нуля и применяете к своему голосу. Вы не просто узнаёте о фильтрах — вы создаёте фильтр, пропускаете через него звук и слышите результат. Теория немедленно становится практикой. Практика немедленно подтверждает теорию.

Как устроена эта книга

Книга состоит из девяти глав. Мы начинаем с фундамента: в первой главе разбираем, как аналоговый звук превращается в цифровой, что такое дискретизация и квантование, и почему их параметры именно такие, какие есть. Это база, без которой невозможно понять всё остальное.

Со второй по четвёртую главу мы изучаем преобразование Фурье и его вариации. Это главный инструмент анализа звука. Мы пройдём путь от наивного дискретного преобразования Фурье, которое работает медленно, к быстрому алгоритму Кули-Тьюки, который изменил мир. Мы разберёмся

с оконными функциями, утечкой спектра и кратковременным преобразованием Фурье, которое лежит в основе спектрограмм.

В пятой и шестой главах мы займёмся фильтрами и свёрткой. Мы поймём, что фильтр и свёртка — это одно и то же, просто с разных точек зрения. Мы напишем свои КИХ- и БИХ-фильтры, научимся строить частотные характеристики и применять фильтры для реальных задач: эквализации, шумоподавления, создания эффектов.

В седьмой главе мы перейдём к адаптивным алгоритмам — фильтрам, которые подстраиваются под изменяющийся сигнал. Это уже уровень профессиональных систем шумоподавления и эхокомпенсации.

В восьмой главе мы займёмся анализом высоты голоса — задачей, которая на первый взгляд кажется простой, но на деле требует хитрых алгоритмов.

В девятой главе мы сменим направление и займёмся синтезом звука — созданием звуков с нуля с помощью Python.

Каждая глава следует единой структуре. Сначала я показываю проблему и даю интуицию. Затем мы разбираем теорию шаг за шагом, без пропусков и без «очевидно, что». Затем пишем код. Затем смотрим, что может пойти не так — и как это исправить. Затем я даю творческое задание для самостоятельного исследования. И в конце — чек-лист для самопроверки.

Что вам понадобится

Для работы с книгой вам нужен компьютер с Python версии 3.9 или выше. Все необходимые библиотеки — numpy, scipy, librosa, soundfile, matplotlib — бесплатны и устанавливаются одной командой. В первой главе я покажу, как всё настроить.

Вам понадобятся базовые знания Python. Вы должны понимать, что такое переменная, функция, список, цикл for и условный оператор if. Вы должны уметь открыть файл в редакторе кода и запустить скрипт из командной строки. Этого достаточно. Всему остальному я научу.

Вам понадобится готовность думать. Эта книга не для пассивного чтения. Вы будете писать код, экспериментировать, ошибаться и исправлять ошибки. Это нормально. На ошибках учатся быстрее, чем на успехах. Если что-то не получается с первого раза — вы на верном пути.

Вам понадобится любопытство. Желание заглянуть под капот и понять, как всё устроено. Если вы из тех людей, кто в детстве разбирал игрушки, чтобы посмотреть, что внутри, — эта книга для вас. Цифровая обработка сигналов — это разобранная на детали аудиовселенная. И она потрясающе красива.

Я благодарен вам, читатель. За то, что взяли эту книгу. За то, что готовы потратить время и усилия на понимание, а не только на применение. Это решение разделяет тех, кто просто использует технологии, и тех, кто ими управляет. После этой книги вы будете во второй группе.

Введение. Почему понимать — это выгодно

О чём эта книга

Возьмите любую программу для обработки звука. Откройте в ней эффект шумоподавления. Вы увидите ползунки с названиями: «Порог», «Соотношение», «Атака», «Релиз». Вы двигаете их туда-сюда, слушаете результат и в какой-то момент думаете: «Вроде неплохо». Но понимаете ли вы, что именно вы сделали? Можете ли вы предсказать, как поведёт себя этот эффект на другой записи, с другим типом шума, с другим голосом? Можете ли вы объяснить другому человеку, почему ползунок «Порог» нужно поставить именно в это положение, а не на пять делений левее?

Большинство создателей контента отвечают на эти вопросы «нет». Они действуют методом тыка. Они запоминают удачные положения ползунков для своей конкретной записи и молятся, чтобы в следующий раз сработало так же. Они не управляют звуком — они угадывают.

Эта книга написана для того, чтобы вы перестали угадывать и начали управлять. Чтобы вы понимали не только «что делать», но и «почему это работает». Чтобы вы могли не просто применить готовый алгоритм, но и написать свой, заточенный под вашу конкретную задачу. Чтобы вы перешли с

уровня пользователя на уровень создателя.

В первой книге — «Python для творческих: звук на твоей стороне» — мы научились использовать готовые инструменты. Мы загружали аудио, убирали шум, выравнивали громкость, меняли голос, микшировали с музыкой. Мы собрали большой автоматический конвейер, который за минуту делает то, на что раньше уходили часы ручного труда. Это был курс молодого бойца. Мы научились стрелять из готового оружия.

Теперь мы пойдём на оружейный завод. Мы разберёмся, как устроены патроны, порох и ствол. Мы поймём физику выстрела и баллистику пули. И в результате мы сможем не только стрелять — мы сможем создавать оружие под себя.

Эта книга — о цифровой обработке сигналов. Не пугайтесь названия. Цифровая обработка сигналов — это просто набор идей о том, как работать с числами, которые представляют звук. Никакой магии. Никаких недоступных простому смертному тайн. Всё, что делают профессиональные программы для обработки звука, можно понять, написать на Python и запустить на своём компьютере. И вы это сделаете.

Для кого эта книга

Эта книга для вас, если вы прочитали первую книгу и хотите идти дальше. Если вы уже умеете применять готовые функции и теперь хотите понимать, что у них внутри. Если вы чувствуете, что стандартных инструментов вам не хватает и вы хотите создавать свои.

Эта книга для вас, если вы не читали первую книгу, но уже немного знакомы с Python. Вы знаете, что такое переменная, функция и массив. Вы можете написать простой цикл. Вам не нужно быть профессиональным программистом — достаточно базовых навыков. Всё остальное я объясню.

Эта книга для вас, если вы интересуетесь обработкой звука не как пользователь, а как исследователь. Если вам мало нажать кнопку — вы хотите знать, что происходит после нажатия. Если вас раздражает, когда что-то работает, а вы не понимаете почему. Если вы из тех людей, которые в детстве разбирали игрушки, чтобы посмотреть, что внутри.

Эта книга для вас, если вы студент технической специальности и вам нужно освоить цифровую обработку сигналов, но учебники написаны сухим математическим языком и вызывают только желание закрыть их и никогда не открывать. Я пишу не учебник. Я пишу книгу, которую интересно читать и по которой можно учиться.

Почему существующие книги не решают проблему

Существует множество книг по цифровой обработке сигналов. Почти все они делятся на две категории. Первая — академические учебники. Они полны формул, интегралов, доказательств теорем. Они строги, точны и абсолютно непригодны для самостоятельного изучения обычным человеком. Если вы не сидите в университетской аудитории с профессором, который объясняет каждый шаг, вы застрянете на третьей странице. Я знаю это, потому что сам застревал.

Я инженер, я люблю математику, но академические учебники по ЦОС вызывали у меня желание биться головой о стол.

Вторая категория — практические руководства. Они говорят: «Вот библиотека, вот функция, вызовите её с такими-то параметрами, получите результат». Они не объясняют, что внутри функции. Они не дают понимания. Они дают рецепты. И когда рецепт перестаёт работать в ваших условиях — а он обязательно перестанет, потому что универсальных рецептов не бывает, — вы остаётесь беспомощны. Вы не можете адаптировать рецепт, потому что не понимаете, как он устроен.

Эта книга идёт по третьему пути. Она объясняет теорию — но не через формулы и доказательства, а через метафоры, визуализации и код. Она даёт практические навыки — но не через готовые рецепты, а через написание алгоритмов с нуля. Вы будете не просто читать о быстром преобразовании Фурье — вы напишете его сами. Вы будете не просто знать, что такое КИХ-фильтр — вы создадите свой фильтр и проверите его на реальном звуке. Теория и практика идут рука об руку, и ни одна не забегает вперёд.

Как устроена эта книга

Книга состоит из девяти глав, каждая из которых посвящена одному фундаментальному концепту цифровой обработки сигналов. Мы начинаем с самых основ — что такое сигнал, как он оцифровывается, почему частота дискретизации именно такая. Затем мы знакомимся с преобразованием

Фурье — главным инструментом анализа звука. Мы проходим путь от наивной реализации, которая работает медленно, до быстрого алгоритма, который изменил мир. Мы изучаем фильтры, свёртку, адаптивные алгоритмы и синтез звука.

Каждая глава построена по одному принципу. Сначала я показываю проблему — зачем вообще нужен этот алгоритм, какую реальную задачу он решает. Затем я даю интуицию — метафору или аналогию, которая делает идею понятной на человеческом уровне, до всяких формул. После этого мы погружаемся в теорию — но не как в лекцию, а как в диалог. Каждый новый концепт выводится из предыдущего шаг за шагом. Как только теория изложена, мы тут же проверяем её кодом. Мы пишем реализацию алгоритма с нуля, строчку за строчкой, и смотрим, работает ли она. Затем мы визуализируем результат — строим графики, спектрограммы, диаграммы, которые подтверждают или опровергают наши ожидания. После основного материала идёт раздел «За кулисами» — исторический контекст, связь с другими областями, неочевидные детали. Затем «Лаборатория ошибок» — что может пойти не так, почему и как это исправить. И наконец «Творческое задание» — эксперименты, которые выводят за пределы главы и побуждают исследовать самостоятельно. Каждая глава завершается чек-листом — списком того, что вы должны уметь после её прочтения.

Главы можно читать последовательно — каждая следую-

щая опирается на предыдущие. Но можно и выборочно — если вас интересуют конкретно фильтры или конкретно синтез звука, открывайте нужную главу. Я постарался сделать изложение максимально связным, но при этом каждую главу можно изучать как самостоятельный модуль.

Что вам понадобится

Для работы с книгой вам нужен компьютер с установленным Python. Если вы читали первую книгу, всё уже установлено. Если нет — не переживайте, я проведу вас через установку в первой главе.

Вам понадобится базовое знакомство с Python. Вы должны знать, что такое переменная, функция, список, цикл for, условный оператор if. Если вы этого не знаете — прочитайте первые две главы нашей первой книги, там это объяснено с нуля. Этого достаточно.

Вам понадобится желание разбираться. Эта книга сложнее первой. В ней есть формулы. Не много, но есть. Я не оставляю их без объяснения — каждую формулу я перевожу на русский язык и проверяю кодом. Но вам придётся думать. Это не развлекательное чтение на ночь. Это учебник, написанный в развлекательном стиле. Разница существенна: вам будет интересно, но вам придётся работать головой.

Вам понадобится привычка экспериментировать. Код в этой книге — не священное писание. Это приглашение к игре. Меняйте параметры. Ломайте алгоритмы. Смотрите, что получается. Лучший способ понять, как что-то работает —

это сломать это и посмотреть, что изменилось. Я даю вам безопасную среду для поломок: вы всегда можете вернуться к исходному коду из книги.

Чего в этой книге нет

В этой книге нет строгих математических доказательств. Я не доказываю теорему Найквиста — я показываю, что будет, если её нарушить. Я не вывожу формулы из аксиом — я объясняю их смысл и проверяю их работу.

В этой книге нет полного охвата цифровой обработки сигналов. Это не энциклопедия и не университетский курс. Я выбрал те темы, которые наиболее важны для практической работы со звуком, и разобрал их глубоко. За пределами книги остались многие интересные вещи: нелинейная обработка, сжатие аудио, психоакустические модели, современные нейросетевые методы. Это темы для следующих книг.

В этой книге нет привязки к конкретным программам или плагинам. Мы работаем только с Python и бесплатными библиотеками. Всё, что вы узнаете, применимо в любом контексте — потому что вы понимаете принципы, а не запоминаете кнопки.

Краткое содержание глав

В первой главе мы начнём с фундамента. Что такое сигнал и как он становится цифровым. Мы разберём дискретизацию и квантование — процессы, которые превращают непрерывный звук в массив чисел. Мы напишем код, который симули-

рует эти процессы, и увидим, что происходит, когда частота дискретизации слишком низкая. Мы познакомимся с алиасингом — коварным эффектом, который возникает при нарушении теоремы Найквиста, и научимся его избегать.

Во второй главе мы познакомимся с преобразованием Фурье — главным инструментом анализа звука. Я расскажу, почему любой звук можно разложить на сумму синусоид, и мы проверим это утверждение кодом. Мы напишем наивную реализацию дискретного преобразования Фурье — она будет работать правильно, но очень медленно. Мы поймём, почему она медленная, и это понимание подготовит нас к следующей главе.

В третьей главе мы ускорим преобразование Фурье в сотни раз. Мы разберём идею быстрого преобразования Фурье — алгоритм Кули-Тьюки, который изменил мир цифровой обработки сигналов. Мы напишем рекурсивную реализацию и сравним её скорость с наивной версией. Мы познакомимся с оконными функциями и поймём, почему простое FFT даёт утечку спектра.

В четвёртой главе мы перейдём к анализу меняющихся во времени сигналов. Мы изучим кратковременное преобразование Фурье — технику, которая лежит в основе спектрограмм и большинства алгоритмов обработки звука. Мы напишем свою реализацию STFT и сравним её с библиотечной.

В пятой главе мы займёмся цифровыми фильтрами. Мы поймём, что фильтр — это просто формула, которая превра-

щает входной сигнал в выходной. Мы напишем несколько фильтров с нуля: скользящее среднее, разностный фильтр, фильтр нижних частот. Мы научимся строить частотную характеристику фильтра и понимать, что она говорит о его поведении.

В шестой главе мы изучим свёртку — математическую операцию, которая лежит в основе фильтрации, реверберации и многих других эффектов. Мы поймём, почему свёртка и фильтр — это одно и то же. Мы напишем быструю свёртку через FFT и применим её для создания реалистичной реверберации.

В седьмой главе мы перейдём к адаптивным алгоритмам — фильтрам, которые подстраиваются под изменяющийся сигнал. Мы реализуем LMS-алгоритм и применим его для подавления нестационарного шума и эхокомпенсации.

В восьмой главе мы займёмся анализом высоты голоса. Мы разберём несколько методов определения основного тона — от простой автокорреляции до кепстрального анализа — и напишем свой детектор pitch.

В девятой главе мы переключимся с анализа на синтез. Мы создадим звуки с нуля: аддитивный синтез, субтрактивный синтез, частотная модуляция, физическое моделирование струны. Мы напишем простой синтезатор на Python.

Как получить максимум от этой книги

Не просто читайте — программируйте. Открывайте редактор кода и набирайте примеры руками. Когда вы набира-

ете код, а не копируете его, ваш мозг работает иначе — он запоминает и понимает. Когда вы копируете — он расслабляется.

Меняйте параметры. Если я пишу np.sin(2 * np.pi * 440 * t), попробуйте 880 вместо 440. Что изменилось? Почему? Такие маленькие эксперименты дают понимание, которое невозможно получить из текста.

Делайте творческие задания. Они не обязательны — вы можете пропустить их и всё равно понять главу. Но если вы их сделаете, понимание станет глубже и останется с вами надолго.

Ведите конспект. Не просто читайте, а записывайте главные мысли своими словами. Когда вы переводите идею из текста книги в свои слова, вы присваиваете её. Она становится вашей.

Обсуждайте с другими. Если у вас есть друг или коллега, который тоже интересуется обработкой звука, читайте книгу вместе. Объясняйте друг другу сложные места. Объяснение другому — лучший способ проверить, действительно ли вы поняли.

Не сдавайтесь на сложных местах. В книге есть сложные концепты. Если вы застряли — отложите книгу на день, вернитесь и перечитайте. Часто понимание приходит после перерыва. Мозг продолжает работать над проблемой в фоновом режиме, даже когда вы спите или гуляете.

Пара слов перед стартом

Я писал эту книгу для вас. Для человека, который хочет не просто использовать инструменты, но понимать их. Для человека, которому мало нажать кнопку — ему нужно знать, что за ней. Для человека, который чувствует, что способен на большее, чем просто следовать инструкциям.

Цифровая обработка сигналов — это красивая область. Она соединяет математику, физику и программирование в единое целое. Она позволяет заглянуть внутрь сигнала и увидеть его структуру — будь то звук, радиоволна, биомедицинские данные или вибрации моста. Она даёт суперсилу: способность анализировать и менять сигналы так, как вы хотите, а не так, как позволяет готовая программа.

В этой книге мы будем работать в основном со звуком, потому что звук — это самый наглядный и доступный пример сигнала. Вы можете записать свой голос, обработать его и тут же услышать результат. Вы можете увидеть спектр своего голоса на графике и сказать: «Вот этот пик — это я». Но принципы, которые вы изучите, универсальны. Преобразование Фурье работает одинаково для звука, для сейсмических волн и для сигналов из космоса. Фильтры, свёртка, адаптивные алгоритмы — весь этот математический аппарат применим к любым данным, которые меняются во времени.

Эта суперсила доступна каждому. Она не требует доро-

гого оборудования или специального образования. Она требует только желания разобраться и готовности думать. Всё остальное я дам.

Поехали.

Глава 1. Звук как математика: сигналы, отсчёты и теорема, которую нельзя нарушать

«Когда ты понимаешь, как работает алгоритм, ты перестаёшь зависеть от того, кто его написал»

О чём эта глава

В первой книге мы работали со звуком как с готовым материалом. У нас был файл, мы его загружали, и внутри Python он превращался в массив чисел. Мы знали, что эти числа — измерения громкости, сделанные много тысяч раз в секунду. Мы знали, что если изменить эти числа, изменится звук. Но мы не задавались вопросом: а почему, собственно, это работает? Почему последовательность чисел, записанных с определённой частотой, может воспроизвести человеческий голос, музыку, шум ветра — любой звук, который мы слышим в реальном мире? Как вообще непрерывный, плавный, аналоговый звук превращается в набор дискретных, прерывистых чисел? И что теряется — если теряется — при таком превращении?

Эта глава даёт ответы на эти вопросы. Мы начнём с самого фундамента: что такое звук с физической точки зрения, как он распространяется в воздухе и как микрофон превращает его в электрический сигнал. Затем мы разберём два клю-

чевых процесса, которые делают возможной цифровую запись: дискретизацию и квантование. Мы узнаем, почему частота дискретизации для компакт-дисков равна 44 100 Гц, а для телефонных разговоров — 8 000 Гц. Мы поймём теорему Найквиста-Шеннона — не как абстрактную математическую истину, а как практическое правило, нарушение которого приводит к слышимым и неприятным последствиям. Мы познакомимся с алиасингом — эффектом, который возникает, когда частота дискретизации слишком низкая, и научимся его избегать.

Вся теория будет немедленно проверяться кодом. Мы напишем несколько скриптов, которые симулируют дискретизацию и квантование на простых сигналах, а затем и на реальном звуке. Мы увидим своими глазами — и услышим своими ушами — что происходит, когда параметры оцифровки выбраны правильно, и что — когда неправильно. К концу этой главы у вас будет не просто знание терминов «дискретизация» и «частота Найквиста», а глубокое понимание того, как аналоговый мир превращается в цифровой и какие ограничения накладывает этот процесс.

Что такое звук

Прежде чем говорить об обработке, давайте поймём, что мы обрабатываем. Звук — это колебания воздуха. Когда вы говорите, ваши голосовые связки вибрируют. Эти вибрации толкают молекулы воздуха, создавая области повышенного и пониженного давления. Эти области распространяются от

вашего рта во все стороны, как круги на воде от брошенного камня. Когда волна давления достигает уха слушателя, она колеблет барабанную перепонку. Колебания барабанной перепонки через систему крошечных косточек передаются в улитку внутреннего уха, где специальные волосковые клетки превращают их в электрические импульсы, которые мозг интерпретирует как звук.

Микрофон работает похоже. Внутри микрофона есть мембрана — тонкая плёнка, которая колеблется под воздействием звуковых волн. Эти колебания превращаются в электрический сигнал: напряжение на выходе микрофона меняется пропорционально звуковому давлению. Если подключить микрофон к осциллографу — прибору, который рисует график напряжения во времени, — мы увидим волну. Эта волна и есть аналоговый звуковой сигнал: непрерывная, плавная линия, которая поднимается и опускается в такт колебаниям воздуха.

У этой волны есть несколько важных характеристик. Первая — амплитуда. Это высота волны, то есть насколько сильно звуковое давление отклоняется от среднего значения. Амплитуда определяет воспринимаемую громкость звука: чем выше волна, тем громче звук. Вторая — частота. Это количество полных колебаний в секунду. Частота определяет высоту звука: чем больше колебаний в секунду, тем выше звук. Частота измеряется в герцах (Гц). Один герц — одно колебание в секунду. Человеческое ухо в среднем слышит звуки от

20 Гц (очень низкий бас) до 20 000 Гц (очень высокий писк). С возрастом верхняя граница снижается — многие взрослые не слышат звуки выше 15 000-16 000 Гц. Третья характеристика — фаза. Это положение волны в конкретный момент времени относительно начала цикла. Две волны одинаковой частоты и амплитуды могут быть сдвинуты друг относительно друга, и этот сдвиг влияет на то, как они складываются.

В реальном мире звук редко состоит из одной частоты. Когда вы говорите, ваши голосовые связки создают сложную волну, в которой смешано множество частот. Когда играет оркестр, звуковое давление в каждой точке пространства — это сумма волн от всех инструментов, отражённых от стен, пола и потолка. Получается очень сложная, но по-прежнему непрерывная волна. И задача цифровой записи — превратить эту непрерывную волну в последовательность чисел, по которой её можно будет точно восстановить.

Дискретизация: делаем непрерывное прерывистым

Цифровой компьютер не может работать с непрерывными сигналами. Компьютер — существо дискретное. Он оперирует отдельными числами в отдельные моменты времени. Чтобы записать звук, нам нужно взять непрерывную звуковую волну и измерить её значение в определённые моменты времени. Этот процесс называется дискретизацией. Мы как бы фотографируем волну много раз в секунду, и каждая фотография — это одно число.

Количество таких измерений в секунду называется частотой дискретизации. Частота дискретизации 44 100 Гц означает, что мы измеряем звуковое давление 44 100 раз в секунду. Почему именно 44 100, а не круглые 40 000 или 50 000? Ответ даёт теорема, открытая Гарри Найквистом и Клодом Шенноном. Она гласит: чтобы точно записать и потом восстановить сигнал, частота дискретизации должна быть как минимум в два раза выше, чем самая высокая частота в этом сигнале.

Человеческое ухо слышит звуки примерно до 20 000 Гц. Значит, чтобы записать всё, что слышит человек, нужна частота дискретизации не менее 40 000 Гц. Разработчики компакт-дисков взяли 44 100 Гц — чуть больше, чем 40 000, с небольшим запасом. Почему именно 44 100, а не 44 000? Это связано с тем, что первые цифровые записи делались на видеомагнитофоны, и частота дискретизации выбиралась так, чтобы быть кратной частоте кадров видео. В Европе это было 25 кадров в секунду, и число 44 100 хорошо ложилось в этот формат. Так техническая необходимость определила стандарт, которым мы пользуемся до сих пор.

Для речи высокая частота дискретизации не нужна. Человеческий голос редко поднимается выше 8 000 Гц, а основные частоты, важные для разборчивости, находятся ниже 4 000 Гц. Поэтому телефонные разговоры оцифровывают с частотой 8 000 Гц — этого достаточно, чтобы понять собеседника, хотя качество заметно хуже, чем у музыки. Для подка-

стов и аудиокниг часто используют 22 050 Гц — половину от CD-качества. Такой выбор экономит место на диске и ускоряет обработку, не жертвуя качеством восприятия речи.

Давайте проверим всё это кодом. Создадим простой синусоидальный сигнал — чистый тон определённой частоты — и попробуем оцифровать его с разной частотой дискретизации.

```
python
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
# Параметры сигнала
duration = 1.0 # одна секунда
freq = 440.0 # нота "ля" первой октавы
# Создаём непрерывное время с очень высокой частотой дискретизации
# Это наша "идеальная" модель аналогового сигнала
sr_continuous = 441000 # в 10 раз выше, чем CD
t_continuous = np.linspace(0, duration, int(sr_continuous * duration), endpoint=False)
y_continuous = np.sin(2 * np.pi * freq * t_continuous)
print(f"Идеальный сигнал: {len(y_continuous)} отсчётов")
print(f"Частота сигнала: {freq} Гц")
# Теперь симулируем дискретизацию с разными частотами
sample_rates = [8000, 11025, 22050, 44100]
```

```
    for sr in sample_rates:
    # Берём каждый n-ный отсчёт из идеального сигнала
    step = sr_continuous // sr
    y_sampled = y_continuous[::step]
    t_sampled = t_continuous[::step]
    print(f"\nЧастота дискретизации: {sr} Гц")
    print(f" Количество отсчётов: {len(y_sampled)}")
    print(f" На один период сигнала приходится {sr / freq:.1f}
отсчётов")
    # Сохраняем для прослушивания
    sf.write(f'sine_{freq}hz_{sr}.wav', y_sampled, sr)
```

Этот код создаёт синусоидальный тон частотой 440 Гц — это нота «ля» первой октавы, стандартный камертон. Сначала мы создаём «идеальный» аналоговый сигнал, используя очень высокую частоту дискретизации — в десять раз выше CD. Затем мы имитируем процесс дискретизации: берём из этого идеального сигнала каждый n-ный отсчёт, чтобы получить сигнал с нужной частотой дискретизации. Мы пробуем четыре варианта: 8 000 Гц (телефонное качество), 11 025 Гц (четверть CD), 22 050 Гц (половина CD) и 44 100 Гц (полное CD-качество).

Запустите скрипт и прослушайте полученные файлы. Файл с частотой 44 100 Гц звучит как чистый, гладкий тон. Файл с 22 050 Гц — почти так же, разницу на слух уловить трудно. Файл с 11 025 Гц — звук заметно грубее, появляется лёгкое дребезжание. Файл с 8 000 Гц — тон всё ещё разли-

чим, но качество сильно страдает. Это потому, что на каждый период сигнала при частоте 8 000 Гц приходится всего около 18 отсчётов, а при 44 100 Гц — около 100. Чем больше отсчётов на период, тем точнее мы можем восстановить форму волны.

Теорема Найквиста и алиасинг

Теперь давайте проверим саму теорему Найквиста на практике. Теорема утверждает: чтобы точно оцифровать сигнал с частотой F, частота дискретизации должна быть строго больше 2F. Что будет, если попытаться оцифровать сигнал с частотой, которая выше половины частоты дискретизации? Возникнет алиасинг — ложные частоты, которых не было в исходном сигнале.

Представьте себе колесо автомобиля в кино. Вы наверняка замечали, что иногда колеса на экране крутятся в обратную сторону или очень медленно, хотя машина едет быстро. Это и есть алиасинг. Камера снимает с частотой 24 кадра в секунду. Если колесо делает чуть меньше 24 оборотов в секунду, каждый кадр захватывает спицы чуть раньше, чем они завершили полный оборот. В результате нам кажется, что колесо вращается назад. Частота вращения колеса выше половины частоты съёмки — и возникает ложное изображение.

В звуке то же самое. Если частота сигнала выше половины частоты дискретизации, после оцифровки она «отражается» и появляется на более низкой частоте. Эта частота-призрак

и называется алиасом.

```
python
import numpy as np
import soundfile as sf
def demonstrate_aliasing(signal_freq, sample_rate, duration=1.0):
"""
Демонстрирует эффект алиасинга.
signal_freq - частота исходного сигнала
sample_rate - частота дискретизации
"""
nyquist = sample_rate / 2
print(f"Частота сигнала: {signal_freq} Гц")
print(f"Частота Найквиста: {nyquist} Гц")
if signal_freq <= nyquist:
print(" Частота сигнала <= частота Найквиста. Алиасинга нет.")
else:
# Вычисляем частоту алиаса
alias_freq = abs(signal_freq - sample_rate * round(signal_freq / sample_rate))
print(f" Частота сигнала > частота Найквиста! Возникает алиасинг.")
print(f" Ложная частота (алиас): {alias_freq:.1f} Гц")
# Генерируем сигнал на высокой частоте
sr_high = 441000
```

```
    t = np.linspace(0, duration, int(sr_high * duration),
endpoint=False)
    y = np.sin(2 * np.pi * signal_freq * t)
    # Дискретизируем
    step = sr_high // sample_rate
    y_sampled = y[::step]
    # Сохраняем
    sf.write(f'alias_{signal_freq}hz_at_{sample_rate}.wav',
y_sampled, sample_rate)
    print(f"                Файл                сохранён:
alias_{signal_freq}hz_at_{sample_rate}.wav\n")
    # Проверяем на нескольких примерах
    demonstrate_aliasing(signal_freq=5000,
sample_rate=44100) # должно быть чисто
    demonstrate_aliasing(signal_freq=25000,
sample_rate=44100) # алиасинг!
    demonstrate_aliasing(signal_freq=40000,
sample_rate=44100) # сильный алиасинг
```

Разберём, что делает этот код. Функция demonstrate_aliasing принимает частоту сигнала и частоту дискретизации. Она вычисляет частоту Найквиста — половину частоты дискретизации. Если частота сигнала ниже или равна частоте Найквиста, всё в порядке. Если выше — вычисляется частота алиаса по формуле: abs(signal_freq - sample_rate * round(signal_freq / sample_rate)). Эта формула показывает, на какой частоте появится ложный сигнал.

Первый пример: сигнал 5 000 Гц, дискретизация 44 100 Гц. Частота Найквиста — 22 050 Гц. Сигнал ниже, алиасинга нет. Второй пример: сигнал 25 000 Гц при той же дискретизации. Частота сигнала выше частоты Найквиста. По формуле алиас будет на частоте 19 100 Гц. То есть мы пытаемся записать звук частотой 25 000 Гц, а после дискретизации слышим 19 100 Гц — совсем другой тон! Третий пример: сигнал 40 000 Гц. Алиас будет на частоте 4 100 Гц — высокий тон превратился в низкий.

Запустите скрипт и прослушайте файлы. Первый файл звучит как высокий, но чистый тон. Второй — тон заметно ниже, хотя мы генерировали более высокую частоту. Третий — тон стал басовитым, что совершенно не соответствует исходным 40 000 Гц. Это алиасинг в действии.

Борьба с алиасингом: антиалиасинговый фильтр

Как с этим бороться? Решение простое: перед оцифровкой нужно удалить из сигнала все частоты выше частоты Найквиста. Для этого используется фильтр нижних частот — антиалиасинговый фильтр. Он пропускает частоты ниже частоты Найквиста и подавляет всё, что выше. В реальных аудиоустройствах — звуковых картах, аудиоинтерфейсах — такой фильтр встроен в аппаратуру. При записи звука на компьютер антиалиасинговый фильтр автоматически применяется до дискретизации, и вы никогда не сталкиваетесь с алиасингом в обычной жизни.

Но мы можем смоделировать этот фильтр в коде, чтобы

понять, как он работает. В следующих главах мы будем подробно изучать фильтры, а пока используем готовый из библиотеки scipy.

```
python
from scipy.signal import butter, sosfilt
def anti_alias_filter(y, sr, cutoff=None):
"""
Применяет антиалиасинговый фильтр (фильтр нижних частот).
cutoff - частота среза. По умолчанию = частота Найквиста * 0.9
"""
nyquist = sr / 2
if cutoff is None:
cutoff = nyquist * 0.9 # небольшой запас
# Создаём фильтр Баттерворта 8-го порядка
sos = butter(8, cutoff / nyquist, btype='lowpass', output='sos')
y_filtered = sosfilt(sos, y)
return y_filtered
# Демонстрируем работу фильтра
sr_high = 441000
t = np.linspace(0, 1.0, int(sr_high * 1.0), endpoint=False)
# Создаём сигнал с частотами 5000 и 25000 Гц одновременно
y_mixed = np.sin(2 * np.pi * 5000 * t) + 0.5 * np.sin(2 * np.pi * 25000 * t)
```

```
# Дискретизируем без фильтра
step = sr_high // 44100
y_bad = y_mixed[::step]
# Применяем антиалиасинговый фильтр, затем дискретизируем
y_filtered = anti_alias_filter(y_mixed, sr_high, cutoff=20000)
y_good = y_filtered[::step]
sf.write('aliasing_bad.wav', y_bad, 44100)
sf.write('aliasing_good.wav', y_good, 44100)
print("Файлы сохранены. Сравните aliasing_bad.wav и aliasing_good.wav")
print("В плохом файле слышен алиас на ~19100 Гц от сигнала 25000 Гц")
print("В хорошем файле сигнал 25000 Гц подавлен фильтром до дискретизации")
```

Этот код создаёт смесь двух частот: 5 000 Гц (безопасная) и 25 000 Гц (опасная, выше частоты Найквиста для 44 100 Гц). Без фильтра после дискретизации мы услышим алиас от 25 000 Гц на частоте около 19 100 Гц. С фильтром частота 25 000 Гц подавляется до дискретизации, и мы слышим только чистый тон 5 000 Гц. Прослушайте оба файла и убедитесь сами.

Квантование: превращаем высоту волны в число

Дискретизация решает проблему времени: мы знаем, как

часто измерять сигнал. Но есть ещё проблема амплитуды. Звуковая волна непрерывна не только во времени, но и по амплитуде. В каждый момент времени звуковое давление может принимать любое значение в некотором диапазоне. Компьютер не может хранить «любое» значение — он может хранить только конечное количество различных чисел. Процесс округления измеренного значения до ближайшего допустимого числа называется квантованием.

Представьте, что вы измеряете рост человека. Вы можете записать «175 сантиметров», но не «175,384927... сантиметров». Вы округляете до целых сантиметров. Это и есть квантование. В цифровом аудио амплитуда округляется до ближайшего значения, которое может быть представлено заданным количеством бит.

Количество бит определяет, сколько различных уровней громкости мы можем различить. При 8 битах у нас есть 2 в 8-й степени = 256 различных уровней. При 16 битах — 65 536 уровней. При 24 битах — более 16 миллионов уровней. Чем больше бит, тем точнее мы можем записать амплитуду и тем меньше шум квантования — разница между истинным значением и его квантованным приближением.

Шум квантования звучит как тихое шипение, добавленное к сигналу. При 16 битах этот шум очень тихий — около -96 децибел относительно максимального сигнала. При 8 битах шум отчётливо слышен. Давайте смоделируем квантование и послушаем разницу.

```
python
def quantize(y, bits):
"""
Квантует сигнал до указанного количества бит.
bits - количество бит (например, 8 или 16)
"""
levels = 2 ** bits
# Диапазон от -1 до 1 делим на levels уровней
y_normalized = y / np.max(np.abs(y)) # нормализуем к диапазону -1..1
y_quantized = np.round(y_normalized * (levels / 2 - 1)) / (levels / 2 - 1)
return y_quantized
# Загружаем реальный голос
import librosa
y, sr = librosa.load('voice_sample.wav', sr=44100)
# Квантуем с разной разрядностью
for bits in [16, 12, 8, 6, 4]:
y_q = quantize(y, bits)
sf.write(f'voice_{bits}bit.wav', y_q, sr)
# Вычисляем шум квантования
noise = y - y_q[:len(y)]
noise_power = np.mean(noise ** 2)
signal_power = np.mean(y ** 2)
snr = 10 * np.log10(signal_power / noise_power) if noise_power > 0 else float('inf')
```

```
    print(f"{bits} бит: отношение сигнал/шум = {snr:.1f} дБ")
print("Файлы сохранены. Прослушайте и сравните качество.")
```

Прослушайте файлы. 16 бит звучит идеально чисто — это стандарт CD-качества. 12 бит — почти незаметная разница. 8 бит — слышно лёгкое шипение, особенно в паузах. 6 бит — шум становится неприятным, голос искажается. 4 бита — речь ещё можно разобрать, но качество ужасное, как через старую рацию.

Интересный факт: отношение сигнал/шум при квантовании можно оценить по простой формуле: примерно 6 децибел на каждый бит. 16 бит × 6 = 96 дБ — именно столько составляет динамический диапазон CD. 24 бита × 6 = 144 дБ — это уже больше, чем способно различить человеческое ухо. Именно поэтому 24-битная запись считается стандартом для профессиональной студийной работы: она даёт огромный запас по динамическому диапазону, позволяя записывать и очень тихие, и очень громкие звуки без риска клиппинга или потери деталей в шуме.

Связь дискретизации и квантования

Дискретизация и квантование работают вместе. Дискретизация решает, как часто мы измеряем сигнал во времени. Квантование решает, с какой точностью мы записываем каждое измерение. Вместе они определяют качество цифровой записи.

Стандартный аудиофайл формата WAV с параметрами 44

100 Гц, 16 бит, моно хранит 44 100 измерений в секунду, каждое из которых занимает 2 байта (16 бит = 2 байта). Одна минута такой записи занимает примерно 5 мегабайт. Час — около 300 мегабайт. Если использовать стерео вместо моно, размер удваивается. Если использовать 24 бита вместо 16 — увеличивается в полтора раза.

Форматы со сжатием, такие как MP3, используют психоакустические модели, чтобы уменьшить размер файла без заметной потери качества. Они выбрасывают частоты, которые человеческое ухо всё равно плохо слышит, особенно в присутствии более громких звуков на соседних частотах. Это называется маскировкой. Благодаря маскировке MP3 может быть в десять раз меньше WAV, но звучать почти так же. Однако для промежуточной обработки всегда лучше использовать несжатые форматы, потому что каждое пережатие MP3 добавляет новые артефакты.

Практический эксперимент: слышим ли мы разницу

Давайте проведём практический эксперимент, который закрепит всё, что мы узнали о дискретизации и квантовании. Мы возьмём одну и ту же запись, оцифруем её с разными параметрами и сравним.

```
python
def resample_audio(y, orig_sr, target_sr):
"""Передискретизирует аудио на новую частоту."""
duration = len(y) / orig_sr
new_length = int(duration * target_sr)
```

```
    t_old = np.linspace(0, duration, len(y), endpoint=False)
    t_new = np.linspace(0, duration, new_length,
endpoint=False)
    y_new = np.interp(t_new, t_old, y)
    return y_new, target_sr
    # Загружаем качественную запись
    y, sr = librosa.load('voice_sample.wav', sr=44100)
    print(f"Исходный файл: {sr} Гц, {len(y) / sr:.1f} сек")
    # Варианты обработки
    configs = [
    {'label': 'CD качество', 'target_sr': 44100, 'bits': 16},
    {'label': 'Подкаст', 'target_sr': 22050, 'bits': 16},
    {'label': 'Телефон', 'target_sr': 8000, 'bits': 8},
    {'label': 'Рация', 'target_sr': 8000, 'bits': 4},
    {'label': 'Алиасинг (без фильтра)', 'target_sr': 8000, 'bits':
16, 'no_filter': True},
    ]
    for cfg in configs:
    y_resampled, new_sr = resample_audio(y, sr, cfg['target_sr'])
    # Для варианта с алиасингом не применяем антиалиасин-
говый фильтр
    if not cfg.get('no_filter'):
    y_resampled = anti_alias_filter(y_resampled, new_sr)
    y_quantized = quantize(y_resampled, cfg['bits'])
    filename = f"experiment_{cfg['label'].replace(' ', '_')}.wav"
    sf.write(filename, y_quantized, new_sr)
```

```
    print(f"Создан: {cfg['label']} -> {filename}")
    print("\nПрослушайте все файлы и обратите внимание:")
    print("- Как меняется качество с уменьшением частоты дискретизации")
    print("- Как появляется шум при уменьшении разрядности")
    print("- Как алиасинг искажает высокие частоты")
```

Этот код создаёт пять версий одного и того же аудио. Первая версия — эталонная, CD-качество. Вторая — качество подкаста, половина частоты дискретизации, но 16 бит. Третья — телефонное качество, 8 000 Гц и 8 бит. Четвёртая — качество рации, 8 000 Гц и всего 4 бита. Пятая — умышленно испорченная версия: низкая частота дискретизации без антиалиасингового фильтра. Прослушайте все пять и обратите внимание на характер искажений в каждом случае.

За кулисами: история теоремы Найквиста

Гарри Найквист работал в компании Bell Labs в 1920-х годах. Он занимался проблемой передачи телеграфных сигналов по проводам и обнаружил, что для точной передачи сигнала нужно делать измерения с частотой как минимум вдвое выше, чем самая высокая частота в сигнале. Его работа 1928 года «Certain Topics in Telegraph Transmission Theory» заложила основу для всей цифровой связи.

Клод Шеннон, также работавший в Bell Labs, в 1948 году опубликовал знаменитую работу «A Mathematical Theory of Communication», в которой обобщил и строго доказал

результаты Найквиста. Теорема стала называться теоремой Найквиста-Шеннона. Шеннон пошёл дальше и показал, что теорема верна не только для телеграфных сигналов, но и для любых сигналов, включая звук. Эта работа стала фундаментом теории информации и сделала возможной цифровую революцию.

Интересно, что сам Найквист не думал о звуке, когда формулировал свою теорему. Он думал о точках и тире в телеграфных линиях. Но математика оказалась универсальной. Теорема, придуманная для передачи сообщений азбукой Морзе, сегодня обеспечивает работу всего цифрового аудио — от телефонных звонков до стриминговых сервисов.

Лаборатория ошибок

После передискретизации звук стал глухим, без высоких частот. Вы использовали слишком низкую частоту дискретизации и не применили антиалиасинговый фильтр. Высокие частоты отразились и превратились в шум. Всегда применяйте фильтр нижних частот с частотой среза чуть ниже частоты Найквиста перед понижением частоты дискретизации.

После квантования в паузах слышно шипение. Это шум квантования. Вы использовали слишком мало бит. Для чистой записи нужно минимум 16 бит. Если вы работаете с 8-битным аудио — например, из старой игры или архивной записи, — примените шумоподавление после квантования, чтобы убрать шипение в паузах.

Файл после обработки стал занимать неожиданно много места. Проверьте параметры сохранения. Возможно, вы сохранили 8-битный файл с 16-битным заголовком или наоборот. Библиотека soundfile по умолчанию сохраняет WAV с теми параметрами, которые вы указали при создании массива, но лучше явно указывать разрядность.

При попытке загрузить файл с нестандартной частотой дискретизации библиотека выдаёт ошибку или предупреждение. Некоторые библиотеки ожидают только стандартные частоты: 8 000, 11 025, 16 000, 22 050, 44 100, 48 000, 96 000 Гц. Если вы создали файл с нестандартной частотой, некоторые плееры могут отказаться его проигрывать. Придерживайтесь стандартных значений.

Алиасинг не всегда звучит плохо. В некоторых музыкальных жанрах алиасинг используют как творческий эффект — например, в чиптюне и lo-fi музыке. Но для речи и большинства видов контента алиасинг нежелателен. Если вы слышите странные свистящие или звенящие призвуки, которых не было в оригинале, — проверьте частоту дискретизации.

Творческое задание

Запишите свой голос с частотой 44 100 Гц и 16 бит. Затем создайте версии с разными параметрами: 22 050 Гц, 16 бит; 8 000 Гц, 16 бит; 8 000 Гц, 8 бит. Найдите минимальные параметры, при которых ваш голос остаётся разборчивым и приятным на слух. Это знание пригодится, когда нужно будет экономить место или передавать аудио по медленному

каналу.

Проведите слепой тест. Дайте другу или коллеге прослушать несколько версий одной записи с разными параметрами дискретизации и квантования. Попросите оценить качество по шкале от 1 до 5. Постройте график зависимости оценки от частоты дискретизации и разрядности. На каком уровне параметров качество перестаёт улучшаться? Это ваш личный порог чувствительности к цифровым артефактам.

Сгенерируйте сигнал с частотой, плавно возрастающей от 1 000 до 30 000 Гц. Оцифруйте его с частотой 44 100 Гц без антиалиасингового фильтра и прослушайте. Вы услышите, как тон сначала повышается, а потом, после пересечения частоты Найквиста, начинает понижаться и уходит вниз. Это классическая демонстрация алиасинга, которая даёт интуитивное понимание эффекта.

В следующей главе мы познакомимся с преобразованием Фурье — главным инструментом анализа звука. Мы узнаем, почему любой звук можно разложить на сумму простых синусоид, и проверим это утверждение кодом. Мы напишем наивное дискретное преобразование Фурье с нуля — оно будет работать правильно, но очень медленно. Мы поймём, почему медленно, и это понимание подготовит нас к третьей главе, где мы ускорим алгоритм в сотни раз с помощью быстрого преобразования Фурье.

Глава 2. Синусоиды, из которых состоит всё

О чём эта глава

В предыдущей главе мы разобрались с тем, как звук становится цифровым. Мы узнали, что микрофон превращает колебания воздуха в электрический сигнал, а аналого-цифровой преобразователь измеряет этот сигнал много тысяч раз в секунду, создавая массив чисел. Но мы пока не ответили на главный вопрос: что с этими числами делать дальше? Как из последовательности измерений громкости понять, какой перед нами звук — голос, музыка, шум ветра или тишина? Как отличить высокий тон от низкого, если оба записаны в виде чисел?

Ответ на этот вопрос даёт преобразование Фурье. Это математический инструмент, который берёт сигнал — любой сигнал, каким бы сложным он ни был — и раскладывает его на составные части. На элементарные кирпичики, из которых построен звук. И эти кирпичики — синусоиды. Чистые, гладкие, математически идеальные волны. Преобразование Фурье говорит нам: «Вот из каких частот состоит твой звук и насколько громка каждая из них». Это как если бы вы взяли оркестровую запись и волшебным образом разделили её на отдельные инструменты — скрипки, виолончели, флейты,

барабаны, — показав, кто и когда играл.

В этой главе мы начнём с самого простого: поймём, что такое синусоида и почему именно она является универсальным строительным блоком для любого звука. Затем мы познакомимся с идеей разложения сигнала на частоты — спектральным анализом. Мы узнаем, что такое комплексные числа, зачем они нужны в обработке звука, и поймём их через простую метафору с вращающейся стрелкой. После этого мы напишем наше первое дискретное преобразование Фурье — DFT — с нуля, на чистом Python. Оно будет работать правильно, но очень медленно. Мы поймём, почему медленно, и это понимание подготовит нас к следующей главе, где мы ускорим алгоритм в сотни раз.

Синусоида: простейший звук во вселенной

Самый простой звук, который можно себе представить, — это чистый тон. Звук камертона. Звук телефонного гудка. Гладкое, ровное колебание без резких скачков и изломов. В математике такой звук описывается функцией синуса — отсюда и название «синусоида».

Почему именно синус? Потому что синус описывает идеальное колебание. Представьте себе грузик на пружине. Если оттянуть его вниз и отпустить, он начнёт колебаться вверх-вниз. График его движения — это синусоида. Представьте себе точку на ободе вращающегося колеса. Если смотреть на неё сбоку, её движение вверх-вниз — тоже синусоида. Представьте себе маятник старинных часов. Его отклонение

от центра — синусоида. Колебания пронизывают природу, и все они, от вибрации атомов до орбит планет, описываются синусоидами или их комбинациями.

У синусоиды есть три главных параметра. Первый — частота. Это количество полных колебаний в секунду, измеряется в герцах. Частота определяет, насколько высоким или низким мы слышим звук. Синусоида с частотой 440 Гц — это нота «ля» первой октавы, стандартный камертон. Синусоида с частотой 220 Гц — «ля» малой октавы, вдвое ниже. Синусоида с частотой 880 Гц — «ля» второй октавы, вдвое выше. Зависимость между частотой и воспринимаемой высотой звука логарифмическая: увеличение частоты вдвое всегда воспринимается как повышение на одну октаву, независимо от того, с какой частоты мы начали.

Второй параметр — амплитуда. Это размах колебаний, максимальное отклонение от нуля. Амплитуда определяет громкость звука. В цифровом аудио амплитуда измеряется в условных единицах от -1 до 1, где -1 и 1 — это максимально возможная громкость без искажений. Когда мы делаем звук громче в коде, мы просто умножаем все числа в массиве на коэффициент больше единицы — увеличиваем амплитуду. Когда делаем тише — умножаем на коэффициент меньше единицы.

Третий параметр — фаза. Это начальное положение колебания в момент времени ноль. Две синусоиды одинаковой частоты и амплитуды, но с разными фазами, будут сдвинуты

друг относительно друга. Если сдвинуть синусоиду ровно на половину периода, она превратится в косинусоиду — ту же форму волны, но начинающуюся не с нуля, а с максимума. Фаза важна, когда несколько звуков складываются вместе: две синусоиды в фазе усиливают друг друга, в противофазе — гасят.

Давайте создадим синусоиду в коде и посмотрим на неё. Заодно послушаем, как звучит чистый тон.

```
python
import numpy as np
import soundfile as sf
# Параметры
duration = 2.0 # две секунды
sample_rate = 44100 # CD-качество
freq = 440.0 # нота "ля"
# Создаём массив моментов времени
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)
# Создаём синусоиду
y = np.sin(2 * np.pi * freq * t)
# Сохраняем в файл
sf.write('sine_440.wav', y, sample_rate)
print(f"Создана синусоида: частота {freq} Гц, длительность {duration} сек")
print(f"Количество отсчётов: {len(y)}")
print(f"Диапазон значений: от {np.min(y):.4f} до
```

{np.max(y):.4f}")

Запустите этот код и откройте полученный файл в любом аудиоплеере. Вы услышите чистый, ровный тон. Не очень интересный музыкально, но очень важный для понимания. Именно из таких тонов, только разных частот и с разными амплитудами, состоит любой звук.

Сложение синусоид: как рождается сложный звук

В реальном мире чистые синусоиды встречаются редко. Разве что камертон или генератор сигналов издают что-то близкое к чистому тону. Большинство звуков гораздо сложнее. Но вот ключевая идея: любой сложный звук можно представить как сумму простых синусоид. Эту идею впервые высказал французский математик Жозеф Фурье в начале XIX века, и она произвела революцию в науке.

Сам Фурье занимался проблемой распространения тепла, а не звука. Он показал, что любое периодическое колебание — неважно, насколько сложное, — можно разложить на сумму синусоид с частотами, кратными основной частоте. Основная частота определяет высоту звука, а дополнительные частоты — обертоны — определяют тембр. Благодаря обертонам мы отличаем скрипку от флейты, даже когда они играют одну и ту же ноту. У скрипки один набор обертонов, у флейты — другой.

Давайте проверим эту идею на практике. Создадим несколько синусоид с кратными частотами и сложим их. Посмотрим, что получится.

```
python
import numpy as np
import soundfile as sf
duration = 2.0
sample_rate = 44100
t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)
# Основная частота — 220 Гц (нота "ля" малой октавы)
f0 = 220.0
# Создаём основную синусоиду и обертоны
fundamental = np.sin(2 * np.pi * f0 * t) # основная частота
harmonic_2 = 0.5 * np.sin(2 * np.pi * f0 * 2 * t) # второй обертон (вдвое выше)
harmonic_3 = 0.33 * np.sin(2 * np.pi * f0 * 3 * t) # третий обертон (втрое выше)
harmonic_4 = 0.25 * np.sin(2 * np.pi * f0 * 4 * t) # четвёртый обертон
harmonic_5 = 0.2 * np.sin(2 * np.pi * f0 * 5 * t) # пятый обертон
# Складываем все синусоиды
y_complex = fundamental + harmonic_2 + harmonic_3 + harmonic_4 + harmonic_5
# Сохраняем основной тон и сложный звук для сравнения
sf.write('fundamental_220.wav', fundamental, sample_rate)
sf.write('complex_tone_220.wav', y_complex, sample_rate)
print("Созданы файлы: fundamental_220.wav и
```

complex_tone_220.wav")

print("Прослушайте оба. Основной тон звучит ровно и скучно.")

print("Сложный тон — богаче, с характером, похож на простой музыкальный инструмент.")

Прослушайте оба файла. Основной тон — чистый, гладкий, sterile. Сложный тон звучит иначе — он теплее, богаче, более музыкально. Похоже на простой синтезатор или орган. Но обратите внимание: высота звука та же самая. И в том, и в другом случае это нота «ля». Потому что высоту определяет основная частота, а тембр — набор и соотношение обертонов.

Спектр: портрет звука в частотной области

Когда мы смотрим на аудиосигнал как на массив чисел — то есть во временной области, — мы видим, как громкость меняется со временем. Это полезное представление, но оно не говорит нам, из каких частот состоит звук. Глядя на волновую форму сложного тона, вы не сможете сказать, сколько в нём обертонов и какой они амплитуды. Для этого нужно другое представление — частотное.

Спектр — это график, который показывает, какая энергия приходится на каждую частоту. По горизонтальной оси откладывается частота в герцах, по вертикальной — амплитуда или энергия. Спектр чистого тона 440 Гц — это один пик на частоте 440 Гц и больше ничего. Спектр сложного тона, который мы только что создали, — это несколько пиков

на частотах 220, 440, 660, 880 и 1100 Гц, убывающих по амплитуде.

Спектр — это не просто красивая картинка. Это мощнейший диагностический инструмент. Глядя на спектр записи, вы можете увидеть фоновый гул на 50 Гц (наводка от электросети), шипение на высоких частотах (шум микрофона), резонансы помещения (усиленные частоты, на которых комната «гудит»). Вы можете понять, почему голос звучит глухо — не хватает высоких частот — или резко — слишком много верхней середины. Спектр показывает то, что ухо слышит, но не может выразить в числах.

Как же получить спектр из временного сигнала? Для этого и нужно преобразование Фурье. Оно переводит сигнал из временной области в частотную.

Дискретное преобразование Фурье: основная идея

Дискретное преобразование Фурье, или DFT, — это алгоритм, который берёт массив из N чисел, представляющих сигнал во времени, и выдаёт массив из N комплексных чисел, представляющих сигнал в частоте. Каждое число в выходном массиве соответствует определённой частоте и говорит нам: «На этой частоте сигнал имеет такую-то амплитуду и такую-то фазу».

Как DFT это делает? Идея проста и гениальна одновременно. Представьте, что вы хотите узнать, есть ли в сигнале частота 100 Гц. Вы берёте синусоиду с частотой 100 Гц, умножаете её на ваш сигнал и суммируете результат. Если в

сигнале есть 100 Гц, синусоида будет с ним совпадать, и после умножения и суммирования получится большое число. Если 100 Гц нет — синусоида будет не в такт с сигналом, умножение даст то положительные, то отрицательные значения, и в сумме они погасят друг друга. Повторите эту процедуру для всех частот, которые вас интересуют — и вы получите спектр.

Формально DFT записывается так. Для каждой частоты k мы вычисляем:

X[k] = сумма по n от 0 до N-1 от x[n] умножить на e в степени минус j умножить на 2π умножить на k умножить на n, делённое на N.

Не пугайтесь этой формулы. Сейчас мы разберём её по кусочкам и превратим в работающий код. Вы увидите, что за страшными символами скрывается простая и красивая идея.

Комплексные числа: вращающаяся стрелка

В формуле DFT есть загадочная буква j и экспонента. Это комплексная экспонента, и она пугает многих новичков. Но комплексные числа — это не что-то сверхъестественное. Это просто удобный математический инструмент для описания вращения.

Представьте себе стрелку на плоскости. У неё есть длина и направление. Обычные числа могут описать длину стрелки, но не направление. Комплексные числа описывают и то, и другое. Комплексное число состоит из двух частей: действительной и мнимой. Действительная часть — это проек-

ция стрелки на горизонтальную ось. Мнимая — проекция на вертикальную ось. Длина стрелки — это модуль комплексного числа. Угол, который стрелка образует с горизонтальной осью, — это фаза.

Когда мы умножаем сигнал на комплексную экспоненту e в степени минус j умножить на 2π умножить на k умножить на n, делённое на N, мы делаем вот что. Мы берём стрелку единичной длины и начинаем её вращать. Частота вращения зависит от k. Для k=0 стрелка стоит на месте. Для k=1 стрелка делает один полный оборот за N отсчётов. Для k=2 — два оборота. И так далее. Умножая сигнал на эту вращающуюся стрелку, мы проверяем, насколько сигнал похож на вращение с данной скоростью. Если сигнал колеблется с той же частотой, что и вращение стрелки, они будут синхронны, и после суммирования получится большое число. Если нет — они будут расходиться, и сумма будет близка к нулю.

Конец ознакомительного фрагмента.