

Евгений Столов

Цифровая обработка сигналов с модулями на Python'e

Предисловие

Всеобщая цифровизация набирает обороты. В основном она сводится к модернизации различных сервисов. С программистской точки зрения это работа с базами данных, облачными репозиториями и пользовательскими интерфейсами. Это требует хороших навыков, но заложенная в продукт математика весьма элементарна. Однако существует область знаний, тесно связанная с практическим применением, в которой используется серьезная математика, это цифровая обработка сигналов и изображений.

Предлагаемая вашему вниманию книга посвящена обработке сигналов. Она основана на курсе лекций, прочитанных автором в Казанском федеральном университете в "Институте вычислительной математики и информационных технологий" и является переработанной версией первой главы книги автора [7]. Отличительная черта книги, которая выделяет ее из большого количества доступных учебников, это использование модулей на языке Python для решения рассматриваемых проблем. Применение специализированных модулей: `numpy`, `scipy`, `scikit-learn`, `pywt`, `bitarray`, `matplotlib` позволяет создавать короткие скрипты, решающие текущую проблему.

Книга содержит традиционный материал.

1. Преобразование Фурье от функций и последовательностей и дискретное преобразование Фурье. Свертки функций и последовательностей. Элементы теории обобщенных функций. Теорема Котельникова-Шеннона и способы восстановления функции по ее значениям в дискретных точках.
2. Теория линейных фильтров. Особенности реализаций фильтров в пакете `scipy.signal`. Проблема сдвига фазы в результате фильтрации и способы ее решения.
3. Примеры применения изложенной теории для решения конкретных задач. Сдвиг спектра сигнала с помощью модуляции. Преобразование Гильберта и его применение для сдвига спектра. Децимация и обратная децимация сигнала. Применение В-Сплайнов для изменения частоты стробирования сигнала. Линейное предсказание. Преобразования Хартли и дискретное косинус-преобразование. Сравнение эффективности различных базисов при сжатии звуковых файлов. Wavelet-преобразование и multiresolution. Метод главных компонент. Использование известной мелодии в качестве водного знака при внедрении в аудио файлы.

Отметим задачу восстановления сигнала по отфильтрованному образцу в случае, когда обратный фильтр не существует. Обычно этот вопрос не включают в учебные пособия.

Книга является учебным пособием, в ней нет новых результатов, поэтому, за редкими исключениями, в тексте отсутствуют ссылки на оригинальные статьи. В конце приведен список источников, которыми автор пользовался при написании текста.

Предполагается, что читатель обладает знаниями по математике в пределах первых двух курсов технического вуза. В тех случаях, когда используется более сложный материал, название параграфа помечается звездочкой. Изучение такого пункта можно опустить, но нужно будет ознакомиться с окончательными результатами. От читателя требуется обладание навыками программирования на Python. При использовании функций из модулей делаются необходимые пояснения. Представленные скрипты имеют малый размер, поэтому внешний источник программ не нужен. Для экспериментов с речевыми файлами и их записи на жесткий диск потребуется микрофон и программа для манипулирования звуковыми файлами. Можно воспользоваться для этой цели бесплатной программой audacity.

В книге отсутствуют задачи для читателя. Вместо этого предлагается проверить работоспособность приведенных скриптов и модифицировать их, чтобы понять сущность задачи.

С замечаниями и сообщениями об ошибках обращаться по адресу ystolov@list.ru

Оглавление

https://t.me/it_boooks/2

1	Основные понятия	1
1.1	Цифровой сигнал. Постановка задачи	1
1.2	Преобразование Фурье и обобщенные функции	3
1.2.1	Случай периодического сигнала	3
1.2.2	Сигнал заданный на всей оси	3
1.3	Свойства преобразования Фурье	7
1.3.1	Формула обращения преобразования Фурье	8
1.3.2	Соотношения между функцией и ее преобразованием Фурье	9
1.3.3	Свертка функций	9
1.3.4	Дифференцирование обобщенных функций. Функция Хевисайда *	10
1.4	Преобразование Фурье от последовательности	12
1.4.1	Свойства преобразования Фурье от последовательности	13
1.5	Дискретное преобразование Фурье	15
1.5.1	Определение ДПФ	16
1.5.2	Оценки спектра сигнала с помощью ДПФ	18
1.5.3	Быстрое преобразование Фурье	19
1.5.4	Пример оценки спектра сигнала с помощью ДПФ	19
1.5.5	Сглаживающие окна	20
1.5.6	Связь между спектром непрерывного сигнала и ДПФ для периодического сигнала	24
1.6	Дальнейшие свойства ДПФ	25
1.6.1	Сдвиг последовательности	26
1.6.2	Свертка последовательностей	26
1.6.3	Корреляция между двумя последовательностями	27
1.6.4	Примеры вычисления корреляции	28
1.6.5	автокорреляция	29
1.7	Восстановление функции	31
1.7.1	Преобразование Фурье от единичной последовательности *	31
1.7.2	Восстановление функции по дискретным значениям	33

1.7.3	Шум при переходе к целому формату в процессе оцифровки	35
2	Фильтрация	37
2.1	Линейные фильтры	37
2.1.1	Устойчивость фильтра	38
2.1.2	Фильтры с конечным временем отклика	39
2.1.3	Фильтры с бесконечным временем отклика	39
2.1.4	Соединение фильтров	40
2.1.5	Устойчивость фильтров с бесконечным временем отклика	41
2.1.6	Случай конечной последовательности	42
2.1.7	Фильтры низких и высоких частот. Идеальный фильтр	43
2.1.8	Реализация фильтров в пакете <code>scipy</code>	44
2.2	Примеры применения различных фильтров	45
2.2.1	Пример FIR фильтра	45
2.2.2	Представление IIR фильтра в виде последовательного соединения фильтров первого и второго порядка	49
2.2.3	Восстановление сигнала по отфильтрованному результату	50
2.3	Фаза сигнала	52
2.3.1	Фазовый сдвиг сигнала в результате фильтрации .	52
2.3.2	Двусторонний фильтр. Обратный проход фильтра .	55
2.3.3	All-pass фильтр	56
2.3.4	Фазовая и групповая задержка	58
3	Примеры обработки цифровых сигналов	60
3.1	Работа со звуковым файлом	60
3.1.1	Загрузка файла	60
3.2	Сдвиг спектра сигнала	61
3.2.1	Сдвиг с помощью модуляции и гребенки фильтров	62
3.2.2	Аналитический сигнал. Преобразование Гильберта	64
3.2.3	Пример работы с аналитическим сигналом	67
3.3	Изменение частоты стробирования сигнала	69
3.3.1	Децимация и обратная децимация сигнала	70
3.3.2	Пример реализации <code>upsampling</code> и <code>downsampling</code> . . .	72
3.3.3	B-Spline и сжатие звуковых файлов	73
3.4	Сжатое описание цифрового файла. Линейное предсказание	77
3.4.1	Линейное предсказание	78
3.5	Примеры использования линейного предсказания	79
3.5.1	Применение линейного предсказания для сжатия сигнала	79
3.5.2	Применение линейного предсказания для получения сглаженного спектра сигнала	80

3.5.3	Обнаружение схожих музыкальных произведений .	81
3.6	Другие преобразования на основе ДПФ, используемые в задачах обработки цифровых файлов	85
3.6.1	Преобразование Хартли (Hartley)	85
3.6.2	Дискретное косинус преобразование (DCT)	87
3.6.3	Сравнение эффективности трех преобразований . .	88
3.7	Преобразования не связанные с ДПФ	91
3.7.1	Дискретное wavelet преобразование	92
3.7.2	Реализация дискретного wavelet преобразования . .	94
3.7.3	Multiresolution	95
3.7.4	Сжатие файла и метод главных компонент	96
3.7.5	Реализация PCA	98
3.8	Внедрение водяных знаков в аудио файлы	100
3.8.1	Выбор водяного знака	100
3.8.2	Внедрение ВЗ с помощью Wavelet-преобразования .	102
3.8.3	Внедрение ВЗ с помощью dct-преобразования . . .	102
3.8.4	Внедрение ВЗ с помощью интерполяции	103

Глава 1

Основные понятия

1.1 Цифровой сигнал. Постановка задачи

В настоящее время методы цифровой обработки сигналов (ЦОС), или digital signal processing (DSP) находят все более широкое применение, вытесняя постепенно методы, основанные на аналоговой обработке. Хотя аналоговые схемы работают быстрее цифровых, исчезает проблема электрической стыковки двух блоков, разработанных независимо друг от друга, что существенно упрощает проектирование.

Исходный сигнал $x(t)$, подлежащий обработке, как правило, является аналоговым. С математической точки зрения это означает, что t может быть любым числом из заданного вещественного интервала, а область значений $x(t)$ – любые вещественные или комплексные числа. В общем случае имеется сигнал $x(t)$, заданный на интервале $[0, \infty)$. В таком виде сигнал непригоден для обработки в компьютере, поэтому нужна его оцифровка. При переходе к оцифровке осуществляются две операции. Сначала от сигнала с непрерывным временем переходят к сигналу с дискретным временем. Выбирается шаг дискретизации T , и вместо исходного сигнала получается последовательность $x[n] = x(nT)$, $n = 0, 1, \dots$. Далее осуществляется сама оцифровка. Для этого выбирается формат для хранения значений функции. Это может быть формат с плавающей точкой либо целый формат. Для хранения звуковых файлов обычно выбирается целый формат. Сначала выбирается формат оцифровки B (количество битов для хранения одного значения), и каждое значение $x[n]$ заменяется неким целым числом, содержащими B битов. Обычно B бывает кратным 8, хотя это не обязательно. Предположим, что существует такое число M , что выполнены неравенства: $-M \leq x[n] \leq M$ для всех n . Интервал $[-M, M]$ разбивается на 2^B частей. После этого каждое значение $x[n]$ заменяется номером интервала, в который попало соответствующее значение. В результате последовательность $x[n]$ заменяется новой последовательностью $y[n]$, но теперь каждый член новой

последовательности принимает значения из интервала $[0, 2^B - 1]$. При желании вместо указанного представления можно перейти к представлению сигнала целыми числами со знаком.

На каждом из упомянутых шагов происходит огрубление сигнала. Первая задача цифровой обработки заключается в оценке искажения исходного сигнала в результате оцифровки. Последующая обработка состоит в извлечении из полученного сигнала нужной информации. Для этого могут понадобиться дальнейшие преобразования, например, подавление шумов, изменение спектра и другие. Преобразования цифровых сигналов принято называть цифровой фильтрацией. Особое место занимает теория линейных фильтров, которая к настоящему времени разработана достаточно глубоко. Для оценки результатов фильтрации часто используется аппарат теории вероятностей и математической статистики.

Даже в оцифрованной форме сигнал занимает много места, и для его передачи или хранения применяют сжатие. Прямое сжатие обычным архиватором без потери информации оказывается мало эффективным. Более интересным является сжатие с потерей информации, учитывающее особенность исходного сигнала. Здесь важно установить критерии допустимой потери информации. В зависимости от выбранного критерия выбирается способ сжатия.

В теоретических исследованиях предполагают, что последовательность бесконечна, но в реальных условиях мы имеем дело лишь с конечными последовательностями. В этой связи нужна оценка потерь, связанных с усечением последовательностей. Хотя внутри компьютера мы имеем дело с дискретным сигналом, как было отмечено выше, исходный сигнал зависит от непрерывного времени. Нужно научиться извлекать информацию о свойствах таких сигналов. Это делается с помощью применения к сигналу $x(t)$ различных преобразований. Суть этих преобразований состоит в проектировании исходного сигнала $x(t)$ на различные элементарные функции. Выбирается пространство со скалярным произведением, которому принадлежат исходный сигнал и элементарные функции, и вычисляются скалярные произведения сигнала и элементарных функций.

Следует сразу же заметить, что как правило, мы не рассматриваем тонкости, относящиеся к математическому обоснованию проводимых манипуляций (сходимость рядов, изменение порядка интегрирования в повторных интегралах, почленное интегрирование или дифференцирование элементов бесконечного ряда). Всегда предполагается законность проводимых операций.

На первом месте среди всех преобразований, используемых для анализа аналогового сигнала, стоит его преобразование Фурье, или спектр сигнала.

1.2 Преобразование Фурье и обобщенные функции

Для каждого класса функций имеется свое определение преобразования Фурье.

1.2.1 Случай периодического сигнала

Пусть сигнал описывается периодической функцией $f(t)$ с периодом равным 1 — $f(t+1) = f(t)$. Известно, что такой сигнал раскладывается в ряд Фурье

$$f(t) = \sum_{k=-\infty}^{\infty} a_k e^{2\pi k j t},$$

где $j^2 = -1$. Коэффициенты a_k составляют спектр сигнала. Содержательно каждый коэффициент указывает вклад заданной частоты в разложение функции. Справедлива формула

$$\int_0^1 \exp(2\pi j k t) dt = \begin{cases} 0 & \text{if } k \neq 0, \\ 1 & \text{if } k = 0. \end{cases} \quad (1.1)$$

Отсюда следует, что коэффициенты находятся по формуле

$$a_k = \int_0^1 f(t) e^{-2\pi k j t} dt.$$

Эта формула может быть интерпретирована следующим образом. Имеется набор функций вида $e^{2\pi k j t}$ и мы ищем проекции функции $f(t)$ на функции из этого набора, поскольку интеграл можем рассматривать как скалярное произведение.

1.2.2 Сигнал заданный на всей оси

Если исходный сигнал задан функцией $f(t)$, определенной на всей вещественной оси, то его преобразование Фурье задается формулой

$$F(w) = \int_{-\infty}^{\infty} \exp(-2\pi j w t) f(t) dt. \quad (1.2)$$

Интеграл рассматривается как скалярное произведение в пространстве L_2 , поэтому функция $F(w)$ трактуется как проекция функции $f(t)$ на функции вида $\exp(2\pi j w t)$, где w произвольное вещественное число. Содержательно $|F(w)|$ это интенсивность исходного сигнала на частоте

w . Формула (1.2) имеет смысл лишь при определенных ограничениях на функцию $f(t)$. Однако в любом случае при обычном понимании интегрирования необходимым условием является убывание этой функции на бесконечности. В реальных условиях это ограничение для стационарного сигнала не имеет места, поэтому предварительно нужно ознакомиться со специальным математическим аппаратом, позволяющим в некоторых случаях обойти данное ограничение. Следует отметить, что дальнейшее изложение отличается от строгих математических доказательств. В силу сказанного, размываются ограничения, при которых приводимые ниже формулы остаются справедливыми. Всегда будем предполагать, что функции, с которыми мы имеем дело, обладают нужными свойствами, и приводимые формулы справедливы.

Преобразование Фурье от единичной функции

Рассмотрим наиболее интересный случай преобразования Фурье, когда $f(t) = 1$. В этом случае интеграл (1.2) не существует. Однако существует математический аппарат, с помощью которого этому интегралу можно придать смысл. Решение заключается в переходе от функций к обобщенным функциям. Эти функции были предложены физиками и инженерами, которые пользовались ими без особых оснований. Позднее математики попытались найти строгое доказательство возможности их использования, но на практике эти функции используют и в тех случаях, когда строгое обоснование отсутствует. При первоначальном изучении следующие два пункта, где рассматривается математическая модель, можно опустить и сразу перейти к следующему пункту.

Обобщенные функции *

Обобщенные функции являются одним из замечательных изобретений, используемых в математике. Они были предложены физиком (Дирак) и инженером (Хевисайд). Значительность этого изобретения заключается в том, что используя данный аппарат можно получить правильный результат даже в тех случаях, когда строго доказать его не удастся. Тем не менее, математики пытаются дать обоснование получаемых результатов хотя бы в отдельных случаях. В частности, был предложен подход на основе функционалов в некотором пространстве. Следующий пункт является описанием этого подхода. Он требует математической подготовки, и при желании его можно опустить, просмотрев лишь представленные формулы.

Обозначим через K множество бесконечно дифференцируемых функций с компактным носителем, то есть существует замкнутое ограниченное множество, которому принадлежат все точки, в которых функция отлична от нуля. Примером такой функции является функция, заданная

равенством

$$\phi(x) = \exp\left(\frac{1}{x-b} - \frac{1}{x-a}\right), x \in [a, b],$$

и $\phi(x) = 0$ в остальных точках. По определению, последовательность $\psi_n(x) \in K$ стремится к функции $\psi(x) \in K$, если все эти функции имеют общий компактный носитель и в каждой точке имеет место обычная сходимость. Непрерывный функционал это линейное отображение $f : K \rightarrow C$ (C – множество комплексных чисел), причем из $\psi_n(x) \rightarrow \psi(x)$ следует $f(\psi_n) \rightarrow f(\psi)$. Если $u(x)$ – интегрируемая функция, то ей соответствует функционал $U(\psi) = \int_{-\infty}^{\infty} \psi(x)u(x)dx$ (в дальнейшем областью интегрирования является вся ось, если не указано в явной форме обратное). Однако существуют функционалы, не представимые в указанной форме. Например, $\Delta(\psi) = \psi(0)$ является непрерывным функционалом. Он записывают в форме $\psi(0) = \int \psi(x)\delta(x)dx$ и называется δ –функцией в честь Дирака. Хотя этот функционал нельзя представить с помощью обычной функции, можно ввести δ –образную последовательность. Положим $\delta_n(x) = n/2$ при $-1/n \leq x \leq 1/n$ и 0 в остальных точках. Интеграл от $\delta_n(x)$ равен 1. При больших n функция $\psi(x) \in K$ представима в виде $\psi(x) = \psi(0) + x\psi'(0) + O(n^{-2})$ при $-1/n \leq x \leq 1/n$, поэтому

$$\int_{-1/n}^{1/n} \psi(x)\delta_n(x)dx = \psi(0) + O(n^{-2})$$

(второе слагаемое исчезает в силу нечетности подынтегральной функции). Это означает, что функционал $\delta(x)$ можно представить в виде предела обычных функционалов.

Вернемся к задаче вычисления преобразование Фурье от единичной функции. Понадобятся несколько вспомогательных утверждений

Пусть $\psi(x), \psi'(x)$ интегрируемы. Тогда

$$\int_A^B \psi(x) \cos(nx)dx \rightarrow 0, \quad n \rightarrow \infty \quad (1.3)$$

для любого конечного интервала $[A, B]$.

Доказательство проводится интегрированием по частям. Аналогичное утверждение справедливо и для интеграла, содержащего $\sin(nx)$.

Из курса анализа известна формула

$$\int_0^{\infty} \frac{\sin(x)}{x} dx = \pi/2.$$

Отсюда следует, что

$$\int_0^{\infty} \frac{\sin(Nx)}{x} dx = \pi/2.$$

при любом положительном N , или

$$\int_0^a \frac{\sin(Nx)}{x} dx \rightarrow \pi/2, \quad a > 0, \quad N \rightarrow \infty. \quad (1.4)$$

Действительно,

$$\int_0^{\infty} \frac{\sin(Nx)}{x} dx = \int_0^a + \int_a^{\infty}$$

Второй интеграл возьмем по частям. Получим

$$\frac{\cos(Na)}{Na} - \int_a^{\infty} \frac{\cos(Nx)}{Nx^2} dx,$$

то есть слагаемое стремится к 0 с ростом N .

Обозначим

$$F_N(w) = \int_{-N}^N e^{-2\pi jwx} dx = \frac{\sin(2\pi wN)}{\pi w}.$$

Не существует обычного предела у этой функции при $N \rightarrow \infty$. Построим функционал

$$\int_a^b \phi(w) F_N(w) dw = \int_a^b \phi(w) \frac{\sin(2\pi wN)}{\pi w} dw,$$

где интервал $[a, b]$ содержит носитель функции $\phi(w)$, а его предельное значение будем считать значением функционала (преобразования Фурье от единицы) от функции $\phi(w)$. Если 0 не попадает в интервал интегрирования, подынтегральная функция не имеет особенностей, и весь интеграл стремится к 0 (см. (1.3)). В противном случае, интеграл стремится к

$$\int_{-c}^c [\phi(0) + w\phi'(0) + w^2 O(c^2)] \frac{\sin(2\pi wN)}{\pi w} dw,$$

где c произвольное малое положительное число. Второе слагаемое исчезает в силу нечетности подынтегральной функции, третье слагаемое

стремится к 0 с ростом N , поскольку функция не имеет особенностей, и при $c \rightarrow 0$, $N \rightarrow \infty$ получаем, используя (1.4), что весь интеграл стремится к $\phi(0)$. Другими словами,

$$\delta(w) = \int \exp(-2\pi j w t) dt$$

это такая функция, что

$$\int \phi(w) \delta(w) dw = \phi(0) \quad (1.5)$$

для любой функции ϕ . Обычная функция не может обладать таким свойством. Для наглядности можно представить, что $\delta(w) = 0$ всюду, кроме нулевой точки, а в нулевой точке она равна бесконечности. Чтобы понять, как примерно выглядит $\delta(w)$, нарисуете график функции (Рис.1.1).

```
import numpy as np
from matplotlib import pyplot as plt
N = 100
TotalPoints = 110
Fun = lambda w: np.sin(2*np.pi*N*w)/(np.pi*w)
Arg = np.linspace(-3.11,3.01,TotalPoints)
Val = Fun(Arg)
fig,ax = plt.subplots()
ax.plot(Arg,Val,'k')
ax.set_yticks([])
```

Обратим внимание на способ задания множества аргументов для Fun с целью избежания деления на нуль. Мы видим всплеск в нулевой точке и осцилляцию в окрестности любой другой точки.

1.3 Свойства преобразования Фурье

Установим свойства, вытекающие из определения преобразования Фурье (1.2). Если преобразование Фурье от функции $f(t)$ равно $F(w)$, то будем записывать это в виде $f(t) \Leftrightarrow F(w)$. В частности, $1 \Leftrightarrow \delta(w)$. Запись применения функционала δ в интегральной форме упрощает вывод соответствующих формул. Все они могут быть доказаны путем подсчета соответствующих функционалов и предельного перехода. Интегральная запись подсказывает правильный ответ. Например подсчитаем

$$\int f(t) \delta(t - u) dt$$

Положим $t - u = v$. Продолжим равенство

$$\int f(v + u) \delta(v) dv = f(u)$$

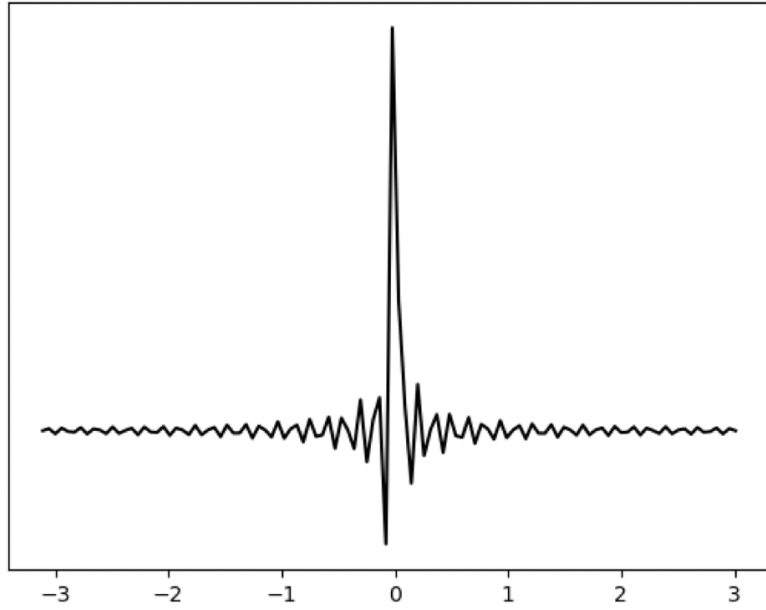


Рис. 1.1: Приближенная форма графика дельта-функции

Аналогично подсчитаем

$$\int \exp(2\pi j w_0 t) dt$$

после замены $t = -u$ получим равенство

$$\int \exp(-2\pi j w_0 u) du = \delta(w_0) = \delta(-w_0).$$

Найдем преобразование Фурье от $\exp(2\pi j w_0 t)$

$$\begin{aligned} \int e^{2\pi j w_0 t} e^{-2\pi j w t} dt &= \\ \int e^{-2\pi (w - w_0) j t} dt &= \delta(w - w_0). \end{aligned}$$

1.3.1 Формула обращения преобразования Фурье

Докажем формулу. Если

$$F(w) = \int f(t) e^{-2\pi j w t} dt$$

то

$$f(t) = \int \exp(2\pi j w t) F(w) dw \quad (1.6)$$

Имеем

$$\int \exp(2\pi j\omega t) F(\omega) d\omega = \int \exp(2\pi j\omega t) \left(\int \exp(-2\pi j\omega u) f(u) du \right) d\omega =$$

Меняем порядок интегрирования

$$\int f(u) du \int \exp(-2\pi j\omega(u-t)) d\omega =$$

$$\int f(u) \delta(u-t) du = f(t).$$

1.3.2 Соотношения между функцией и ее преобразованием Фурье

Перечислим наиболее важные формулы, связанные с преобразованием Фурье

1. Если $f(t) \Leftrightarrow F(\omega)$, $g(t) \Leftrightarrow G(\omega)$, то $af(t) + bg(t) \Leftrightarrow aF(\omega) + bG(\omega)$.
2. $f(t-a) \Leftrightarrow \exp(-2\pi j\omega a) F(\omega)$
3. $f(at) \Leftrightarrow \frac{1}{a} G\left(\frac{\omega}{a}\right)$
4. $f'(t) \Leftrightarrow 2\pi j\omega F(\omega)$

Первые три формулы вытекают из определения. Докажем последнюю из формул. Имеем

$$f(t) = \int F(\omega) e^{2\pi j\omega t} d\omega$$

Возьмём производную по t от обеих частей равенства. Получим

$$f'(t) = \int F(\omega) 2\pi j\omega e^{2\pi j\omega t} d\omega$$

Нужное соотношение получается из формулы обращения.

Обратим внимание на изменение преобразования Фурье при сдвиге функции (№ 2 в списке). Теория линейной фильтрации основана на этой формуле.

1.3.3 Свертка функций

Сверткой двух функций $f(t)$ и $g(t)$ называется функция $h(t) = f(t) * g(t)$, заданная формулой:

$$h(t) = \int f(t-x)g(x)dx.$$

Сразу же отметим, что операция свертки является коммутативной. Имеет место соотношение $f(t) * g(t) \Leftrightarrow F(w)G(w)$. Действительно

$$\begin{aligned} \int f(t) * g(t) \exp(-2\pi j w t) dt &= \int \exp(-2\pi j w t) \left(\int f(t-x)g(x)dx \right) dt = \\ \int g(x)dx \int \exp(-2\pi j w t) f(t-x)dt &= \int g(x)dx \int \exp(-2\pi j w (t-x+x)) f(t-x)dt = \\ \int g(x) \exp(-2\pi j w x)dx \int \exp(-2\pi j w (t-x)) f(t-x)d(t-x) &= G(x)F(w). \end{aligned}$$

Двойственное соотношение имеет вид $f(t)g(t) \Leftrightarrow F(w) * G(w)$. Все эти формулы легко выводятся формальным изменением порядка интегрирования так же, как и при выводе формулы обращения. Вообще говоря, не предполагается, что функция $f(t)$ - вещественная. Если же это так, то $F(-w) = \bar{F}(w)$. Обратно, выполнение данного равенства обеспечивает вещественность исходной функции.

$$\bar{f}(t) = \int \overline{F(w) \exp(2\pi j w t)} dw = \int \bar{F}(w) \exp(-2\pi j w t) dw = f(t).$$

Кроме того, преобразование Фурье является ортогональным преобразованием, то есть сохраняет норму функции в пространстве L_2 :

$$\begin{aligned} \int F(w)\bar{F}(w)dw &= \int dw \int f(u)e^{-2\pi j w u} du \int \bar{f}(v)e^{2\pi j v w} dv = \\ \iint f(u)\bar{f}(v)dudv \int e^{-2\pi j w(u-v)} dw &= \int f(u)du \int \bar{f}(v)\delta(u-v)dv = \int f(u)\bar{f}(u)du \end{aligned}$$

1.3.4 Дифференцирование обобщенных функций. Функция Хевисайда *

Предположим, что некоторый функционал A , примененный к функции $\phi(t)$ и записанный в интегральной форме, имеет вид

$$A(\phi) = \int \alpha(t)\phi(t)dt$$

Его производная определяется формулой

$$A'(\phi) = - \int \alpha(t)\phi'(t)dt$$

Эта формула получается из формального интегрирования по частям с учетом того, что функция $\phi(t)$ равна нулю на бесконечности. В качестве

примера рассмотрим функцию (функционал) Хевисайда. Функционал определяется с помощью функции Хевисайда $\eta(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$.

$$H(\phi) = \int \eta(t)\phi(t)dt = \int_0^{\infty} \phi(t)dt$$

Производная от функционала

$$H'(\phi) = - \int_0^{\infty} \phi'(t)dt = \phi(0)$$

Это означает, что $H' = \delta$. Найдем преобразование Фурье от $\eta(t)$. Пусть это X . Тогда преобразование Фурье от производной должно равняться $2\pi jwX$. Но производная от функционала равна δ , преобразование Фурье последней есть 1, то есть имеет место формула

$$2\pi jwX = 1 \quad (1.7)$$

Однако вывод $X = \frac{1}{2\pi jw}$ оказывается неверным. Напомним стандартную процедуру вычисления значений обобщенной функции. Положим

$$X_N(w) = \int_0^N e^{-2\pi jw t} dt = \frac{1}{2\pi jw} (1 - e^{-2\pi jw N})$$

Надо найти предел $\int X_N(w)\phi(w)dw$, $N \rightarrow \infty$. Сразу выделим член, не зависящий от N /

$$\int \frac{1}{2\pi jw} \phi(w)dw$$

Займемся вторым слагаемым

$$\int \frac{1}{2\pi jw} \phi(w) (-\cos(2\pi w N) + j \sin(2\pi w N)) dw$$

У подынтегральной функции особенность в 0. Интеграл разбивается на три слагаемых

$$\int = \int_{-\infty}^{-\epsilon} + \int_{-\epsilon}^{\epsilon} + \int_{\epsilon}^{\infty},$$

где ϵ произвольное положительное число. Первое и третье слагаемые не имеют особенностей и они интегрируемы на бесконечности, поэтому они

стремятся к 0 с ростом N . Рассмотрим среднее слагаемое. Для малых ϵ это

$$\int_{-\epsilon}^{\epsilon} \frac{1}{2\pi jw} (\psi(0) + w\psi'(0) + w^2 O(\epsilon)) (-\cos(2\pi wN) + j \sin(2\pi wN)) dw$$

Функция $\cos(w)/w$ является нечетной, поэтому

$$\int_{-\epsilon}^{\epsilon} \frac{1}{2\pi jw} \phi(0) \cos(2\pi wN) dw = 0$$

Ранее было доказано, что

$$\int_{-\epsilon}^{\epsilon} \sin(2\pi wN)/w dw \rightarrow \pi \cdot N \rightarrow \infty$$

Все остальные интегралы не имеют особенностей в подынтегральной функции, поэтому они стремятся к 0 с ростом N . Окончательно получаем, что

$$X = \frac{1}{2\pi jw} + \delta(w)/2$$

Обратим внимание, что (1.7) является верной, поскольку $w\delta(w) = 0$.

1.4 Преобразование Фурье от последовательности

Рассмотренное выше преобразование Фурье от функции с непрерывным аргументом имеет теоретическое значение, поскольку в компьютере мы имеем дело с сигналом с дискретным временем. Для таких сигналов вводят специальное преобразование Фурье – преобразование от последовательности. Это другое преобразование, хотя идейно оно связано с преобразованием Фурье от функций.

Преобразованием Фурье от последовательности y называется

$$Y(w) = \sum_{n=-\infty}^{\infty} y[n] \exp(-2\pi jwn) \quad (1.8)$$

Отметим, что функция $Y(w)$ является периодической с периодом 1. Это принципиальное различие между преобразованиями Фурье от функции и от последовательности. Следует заметить, что часто преобразование Фурье от последовательности задают формулой

$$Y(w) = \sum_{n=-\infty}^{\infty} y[n] \exp(-jwn).$$

В этом случае период функции $Y(w)$ равен 2π . Это не вносит принципиальных различий в теорию, но существенно при использовании функций из пакетов цифровой обработки. Будем пользоваться первым определением.

Указанное преобразование возникает естественным образом в цифровой обработке. Пусть имеется сигнал $x(t)$, и выбран шаг дискретизации T . Подынтегральная функция заменяется ступенчатой функцией $y[n] = x(nT)$, постоянной на отрезках длины T , включая экспоненту, а ее интегральное преобразование Фурье превращается в сумму

$$Y_T(w) = T \sum_{n=-\infty}^{\infty} y[n] \exp(-2\pi j T w n) \quad (1.9)$$

Это периодическая функция с периодом $1/T$. Положив $T = 1$, получим указанное выше определение. Формула (1.9) имеет важное значение при интерпретации дискретного преобразования Фурье при заданном T .

1.4.1 Свойства преобразования Фурье от последовательности

Теперь установим свойства преобразования Фурье от последовательности аналогичные свойствам преобразования от функции. Как обычно, преобразования типа почленного интегрирования ряда, перестановки порядка суммирования и тому подобные будут проводиться без какого-либо обоснования. Предполагается, что соответствующие функции обладают необходимыми свойствами. Ограничимся случаем, когда $T = 1$.

Сдвиг последовательности

Пусть x последовательность, тогда x^p – сдвинутая последовательность, которая задается формулой $x^p[n] = x[n - p]$. Имеем

$$X(w) = \sum_{n=-\infty}^{\infty} x[n] e^{-2\pi j w n}$$

Положим $y = x^p$. Тогда

$$Y(w) = \sum_{n=-\infty}^{\infty} y[n] e^{-2\pi j w n} = \sum_{n=-\infty}^{\infty} x[n - p] e^{-2\pi j w (n - p + p)} = e^{-2\pi j w p} X(w) \quad (1.10)$$

Формула обращения

Как уже отмечалось, преобразование от последовательности является периодической функцией. Обратное преобразование получается почлен-

ным интегрированием ряда (1.8). Заметив, что

$$\int_0^1 e^{-2\pi jw(n-p)} dw = \begin{cases} 1 & n = p \\ 0 & n \neq p \end{cases},$$

получим, что обратное преобразование задается формулой

$$\int_0^1 X(w) e^{2\pi jwp} dw = \sum_{n=-\infty}^{\infty} x[n] \int_0^1 e^{-2\pi jw(n-p)} dw = x[p] \quad (1.11)$$

Свертка последовательностей

Следующие свойства преобразования Фурье формулируются с помощью понятия свертки. Для последовательностей очевидным образом определяются арифметические операции сложения и умножения на число. Если $z = x + y$, то $z[n] = x[n] + y[n]$. Если A число, а $y = A \cdot x$, то $y[n] = Ax[n]$.

Свертка z двух последовательностей x и y обозначается символом $z = x * y$ и определяется формулой :

$$z[n] = \sum_{k=-\infty}^{\infty} x[n-k]y[k] \quad (1.12)$$

В дальнейшем используются важные свойства свертки.

1. Коммутативность. $x * y = y * x$.
2. Линейность.

$$x * (A_1 \cdot y + A_2 \cdot y) = A_1 \cdot x * y + A_2 \cdot x * y$$

3. Инвариантность. Если $z = x * y$, то $x^p * y = z^p$.

Докажем инвариантность. Пусть $z_1 = x^p * y$.

$$z_1[n] = \sum_{k=-\infty}^{\infty} z_1[n-k]y[k] = \sum_{k=-\infty}^{\infty} x[n-p-k]y[k] = z[n-p].$$

Как будет показано далее, это свойство, несмотря на его тривиальность, имеет принципиальное значение в теории линейных систем.

Свертка преобразований Фурье от последовательностей

Свертка двух преобразований Фурье от последовательностей обозначается символом $Z(w) = X(w) * Y(w)$ и определяется формулой

$$Z(w) = \int_0^1 X(w-t)Y(t)dt. \quad (1.13)$$

Следующее утверждение является аналогом соответствующих свойств преобразования Фурье от функций. Преобразование от свертки двух последовательностей равняется произведению их преобразований Фурье, а преобразование от произведения двух последовательностей есть свертка их преобразований Фурье.

Найдем преобразование Фурье от произведения последовательностей. Имеем

$$x[n]y[n] = \int_0^1 X(u)e^{2\pi j u n} du \int_0^1 Y(v)e^{2\pi j v n} dv = \iint X(u)Y(v)e^{2\pi j n(u+v)} dudv$$

Сделаем замену переменных, положив $u+v=w$. В силу периодичности подынтегральных функций с периодом 1, интеграл по любому интервалу длины 1 равен этому интегралу по интервалу $[0,1]$. Двойной интеграл переписывается в виде повторного в следующем виде

$$\int_0^1 e^{2\pi j n w} dw \int_0^1 Y(w-u)X(u)du.$$

Теперь из (1.11) вытекает первая часть утверждения. Найдем преобразование Фурье от свертки. Перемножая эти ряды

$$X(w) = \sum_{n=-\infty}^{\infty} x[n]e^{-2\pi j w n} \quad Y(w) = \sum_{n=-\infty}^{\infty} y[n]e^{-2\pi j w n}$$

и собирая коэффициенты при одинаковых степенях, получим

$$X(w)Y(w) = \sum_n z[n]e^{-2\pi j w n},$$

где $z[n] = x[n] * y[n]$. Из доказанного соотношения вытекает ассоциативность операции свертки, $(x*y)*z = x*(y*z)$, поскольку указанным свойством обладает произведение функций (преобразований Фурье в данном случае).

1.5 Дискретное преобразование Фурье

В компьютере мы имеем дело с последовательностями конечной длины. Для работы с такими последовательностями используют специальное

преобразование Фурье, которое называется 'Дискретное преобразование Фурье' (ДПФ). Это преобразование связано с преобразованиями Фурье от функций или последовательностей, но формально от них не зависит. С его помощью удастся получить оценки спектра исходных бесконечных сигналов. При использовании ДПФ значения спектра подсчитываются лишь для конечного множества значений. Этим обстоятельством объясняется название данного метода. Теория ДПФ является самостоятельной дисциплиной.

1.5.1 Определение ДПФ

Пусть $f(x)$ - некоторая функция $x = 0, 1, \dots, N-1$, принимающая вещественные или комплексные значения. Функция считается периодической с периодом N , $f(x+N) = f(x)$. С этой функцией связана последовательность вида $f(0), f(1), \dots, f(N-1)$. Таким образом, с каждой функцией с областью определения $0, 1, 2, \dots, N-1$ связывается последовательность длины N . Множество указанных последовательностей образует линейное пространство размерности N . В этом пространстве определено скалярное произведение двух функций:

$$\langle f, g \rangle = \sum_{k=0}^{N-1} f(k) \bar{g}(k).$$

Рассмотрим функции χ_k , определенные равенствами

$$\chi_k(x) = \frac{1}{\sqrt{N}} e^{2\pi j k x / N}, k = 0, 1, \dots, N-1.$$

Значения этих функций можно отобразить с помощью скрипта

```
import numpy as np
from matplotlib import pyplot as plt
N = 17
Arr = np.zeros(N).astype('complex')
for k in range(N):
    Arr[k] = np.exp(2*np.pi*1j*k/N)
fig, ax = plt.subplots()
ax.set_xticks([])
ax.set_yticks([])
ax.scatter(Arr.real, Arr.imag)
```

Результат представлен на рисунке Рис.1.2. Точки находятся в вершинах правильного N -угольника. Подсчитаем скалярное произведение

$$\langle \chi_m, \chi_n \rangle = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi j k (m-n)/N}.$$

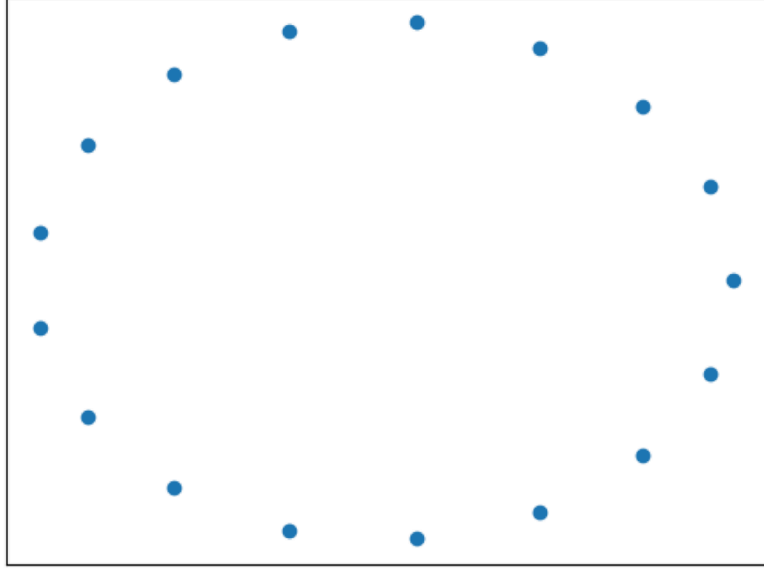


Рис. 1.2: Графическое представление функций базиса

Это геометрическая прогрессия со знаменателем $Z = e^{2\pi j(m-n)/N}$. Если $Z = 1$, то и сумма равна 1, в противном случае сумма равна

$$\frac{1 - Z^N}{1 - Z} = 0,$$

поскольку $Z^N = 1$. Другими словами, имеет место равенство

$$\langle \chi_m, \chi_n \rangle = \delta_{m,n}, \quad (1.14)$$

где

$$\delta_{m,n} = \begin{cases} 1 & \text{if } m = n, \\ 0 & \text{othercase.} \end{cases}$$

Это означает, что функции χ_k составляют ортонормированный базис пространства. При этом для произвольной строки $f = (f(0), f(1), \dots, f(N-1))$ координаты разложения по этому базису равны $F(k) = \langle f, \chi_k \rangle$, то есть

$$(f(0), f(1), \dots, f(N-1)) = \sum F(k) \chi_k.$$

Эти формулы обычно записывают в виде

$$f(x) = \frac{1}{\sqrt{N}} \sum_k F(k) e^{2\pi j k x / N}, \quad F(k) = \frac{1}{\sqrt{N}} \sum_x f(x) e^{-2\pi j k x / N} \quad (1.15)$$

и называют дискретным преобразованием Фурье. В большинстве пакетов, содержащих ДПФ, множитель $1/\sqrt{N}$ исключают из прямого преобразования Фурье и включают $1/N$ в обратное преобразование. Формулы

приобретают следующий вид

$$F(k) = \sum_x f(x)e^{-2\pi j k x / N}, \quad f(x) = \frac{1}{N} \sum_k F(k)e^{2\pi j k x / N}$$

Из формулы (1.15) следует, что $F(k)$ есть аналог значения преобразования Фурье исходной функции, вычисленного в точке k/N . Ниже приводится уточнение этого утверждения. В дальнейшем условимся обозначать связь между последовательностью и ее ДПФ в форме $(f(0), f(1), \dots, f(N-1)) \Leftrightarrow (F(0), F(1), \dots, F(N-1))$.

1.5.2 Оценки спектра сигнала с помощью ДПФ

В данном пункте остановимся на эффектах, которые сопутствуют исследованию сигнала с помощью ДПФ. Следует различать сигнал с непрерывным временем и его спектр от последовательности с дискретным временем, полученную из сигнала с некоторым шагом T , и ее спектр. Более подробно о связи этих понятий будет сказано далее, а пока ограничимся простыми примерами. В основе всех вычислений лежат две формулы: если дискретные значения получены в результате выборки с шагом T из сигнала $x(t)$, то спектр этого сигнала оценивается суммой

$$X(w) = \sum_{n=-\infty}^{\infty} x(nT)e^{-2\pi j n T w}.$$

Когда подсчитываются коэффициенты ДПФ от $x[n] = x(nT)$, по N точкам, то это делается с помощью

$$F(k) = \sum_0^{N-1} x[n]e^{-2\pi j n k / N} = \sum x(nT)e^{-2\pi j n T k / (TN)}.$$

Это означает, что $F(k)$ есть оценка спектра исходного сигнала в точке

$$k/(TN). \quad (1.16)$$

Если исходная функция f вещественная, то в дискретном преобразовании Фурье присутствует избыточность, так как из N вещественных чисел получаются N комплексных чисел. Из определения следует, что

$$\begin{aligned} \bar{F}(k) &= \sum_{x=0}^{N-1} f(x)e^{2\pi j k x / N} = \sum_{x=0}^{N-1} f(x)e^{2\pi j x (k-N) / N} = \\ &= \sum_{x=0}^{N-1} f(x)e^{-2\pi j x (N-k) / N} = F(N-k). \end{aligned}$$

Отсюда получается, что $\bar{F}(k) = F(N - k)$ и $|F(k)| = |F(N - k)|$, поэтому при вычислении модуля спектра рассматривают только коэффициенты $F(k), k = 0, 1, \dots, [N/2]$ (последний индекс - целая часть $N/2$). Таким образом возникает первое ограничение на соотношение спектра и длину N для вещественных функций – $k < N/2$.

1.5.3 Быстрое преобразование Фурье

ДПФ имеет огромную область применения, при этом часто вычислительные ресурсы являются ограниченными. В этой связи большое внимание уделяется эффективным способам вычисления. Согласно традиции, сложность алгоритма вычисления определяется количеством операций умножения с плавающей точкой. По определению, для вычисления одного коэффициента требуется N перемножений (экспоненты подсчитываются заранее). Это означает, что для подсчета всех коэффициентов потребуется N^2 перемножений. Любой способ решения данной задачи, когда удастся уменьшить это число, называется быстрым. Программы, осуществляющие такие вычисления, обычно, имеют имя Fast Fourier Transform (fft). Самая быстрая известная схема вычислений относится к $N = 2^m$, где m любое натуральное число. После некоторых ухищрений ее удастся приспособить для произвольных N . Поскольку на практике используют готовые программы, мы не будем вдаваться в тонкости реализации. Самые популярные программы доступные для вызова из скриптов на Python это `numpy.fft.fft` и `scipy.fftpack.fft`

1.5.4 Пример оценки спектра сигнала с помощью ДПФ

Рассмотрим пример

```
import numpy as np
from matplotlib import pyplot as plt
N = 512
Freq = 32
T = 1/(2*N)
Arg = np.arange(N)*(2*np.pi*T*Freq)
Sn = np.sin(Arg)
Spect = np.fft.fft(Sn)[:N//2]
fig,ax = plt.subplots()
ax.set_yticks([])
ax.plot(range(20),abs(Spect[range(20)]))
```

Результат представлен на Рис.1.3. Выполнив этот фрагмент, получим, что спектр $|F(k)|$ имеет единственный максимум при $k = 16$.

Поясним полученный результат. Исходная последовательность порождена функцией синус с периодом $1/Freq$. Длина ДПФ равна N . ДПФ

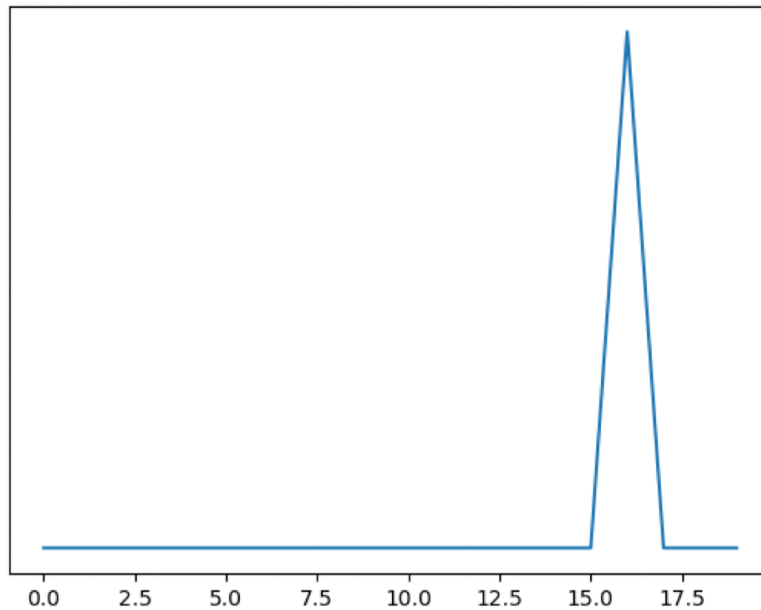


Рис. 1.3: Оценка спектра с помощью ДПФ

https://t.me/it_boooks/2

подсчитывается с помощью функций из пакета `numpy.fft`. Поскольку функция вещественная, отображается только первая половина спектра. Значения функции выбраны с шагом T . Согласно (1.16), величине $k = 16$ отвечает частота `Freq`.

1.5.5 Сглаживающие окна

При работе с конечными последовательностями возникают дополнительные эффекты, один из которых будет рассмотрен в данном пункте. Рассмотрим скрипт

```
N = 64
Arg = np.arange(N)/N
Sn = np.sin(2*np.pi*7*Arg)
Arr = Sn[6:58]
Ln = len(Arr)
plt.plot(Arr)
```

График функции представлен на Рис.1.4. Параметры подобраны таким образом, чтобы значения в начале и в конце массива сильно различались. Оценим спектр этого сигнала.

```
FArr = np.fft.fft(Arr)
plt.plot(abs(FArr[:Ln//2]))
```

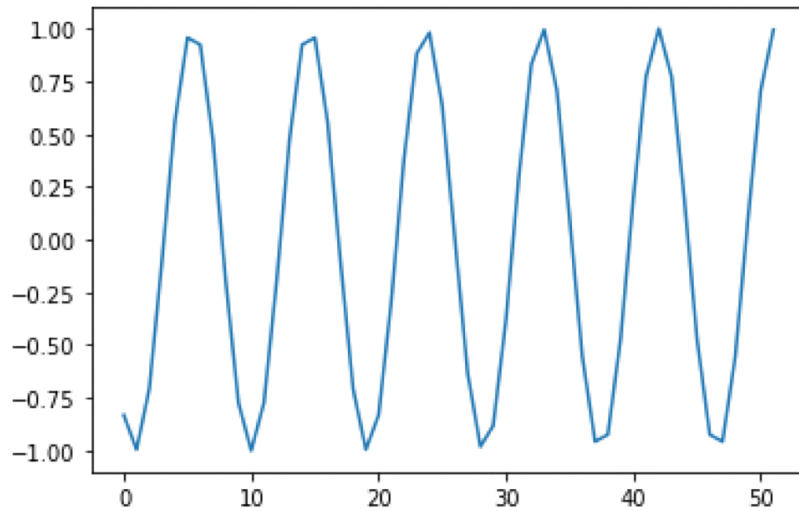


Рис. 1.4: Пример отрезка функции с разрывом на концах

По условию, все последовательности считаются периодическими, поэтому последовательность Arr имеет разрыв (можно начать ее с элемента отличного от 0, и разрыв становится явным). График на Рис. 1.5 имеет максимум при $k = 6$. Воспользовавшись (1.16), где $T = 1/64$, $N = Ln$, получим правильную оценку спектра. Однако, наряду с правильной оценкой спектра будут присутствовать и другие числа, хотя в исходном сигнале присутствует единственный синус. Это следствие наличия разрыва. Такая ситуация обычно имеет место при работе с ДПФ, и для ее смягчения была предложена процедура сглаживания. Ее суть заключается в выборе специальной последовательности $window$, с нулями на концах интервала. Длина функции выбирается равной длине исследуемой последовательности, после чего последовательность заменяется на произведение $window$ и этой последовательности. Выбор $window$ является непростой задачей, было предложено несколько вариантов, но наиболее популярными оказались две последовательности, предложенные Hamming'ом и Hanning'ом. Обе последовательности состоят из полупериода функции $\cos$ и имеют минимальное различие. Ниже будет сказано о критерии, по которому следует выбирать эти последовательности.

Применим сглаживание в нашем примере

```
from scipy import signal as sgn
Wn= sgn.hann(Ln)
WArr = Arr*Wn
FWArr = np.fft.fft(WArr)
plt.plot(abs(FWArr[:Ln//2]))
```

Результат представлен на Рис. 1.6. График получился более сжатый,

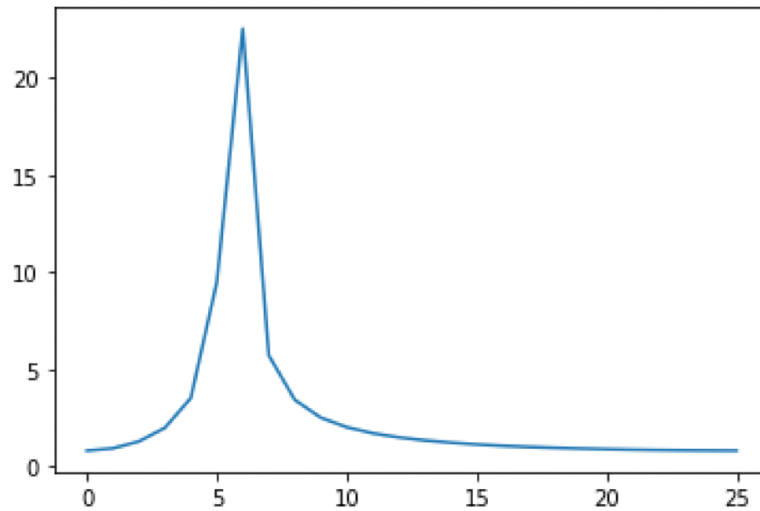


Рис. 1.5: График преобразования Фурье функции с разрывом

а максимальное значение также достигается при $k = 6$. Практическую пользу от сглаживания можем оценить следующим образом. Найдем отношение всех коэффициентов к максимальному.

```
Rel1 = abs(FArr[:Ln//2]/FArr[6])
Rel2 = abs(FWArr[:Ln//2]/FWArr[6])
fig,ax = plt.subplots()
ax.plot(Rel1,'k',label = 'break')
ax.plot(Rel2,'r',label = 'window',)
ax.legend()
```

На Рис. 1.7 видно, что использование сглаживания лучше подавляет фантомные частоты.

Выбор сглаживающего окна

Сделаем несколько замечаний о выборе сглаживающего окна. Это делается в рамках следующей модели. Имеются бесконечная последовательность и ее преобразование Фурье $x[n] \Leftrightarrow X(w)$. Требуется оценить $X(w)$ по конечному отрезку длины N . Строим новую бесконечную последовательность, состоящую из окна $window$ и нулей, $window \Leftrightarrow W(w)$. Перемножаем эти последовательность и вычисляем преобразование Фурье $\tilde{X}(w)$ полученной последовательности.

$$\tilde{X}(w) = \int_0^1 W(w-t)X(t)dt$$

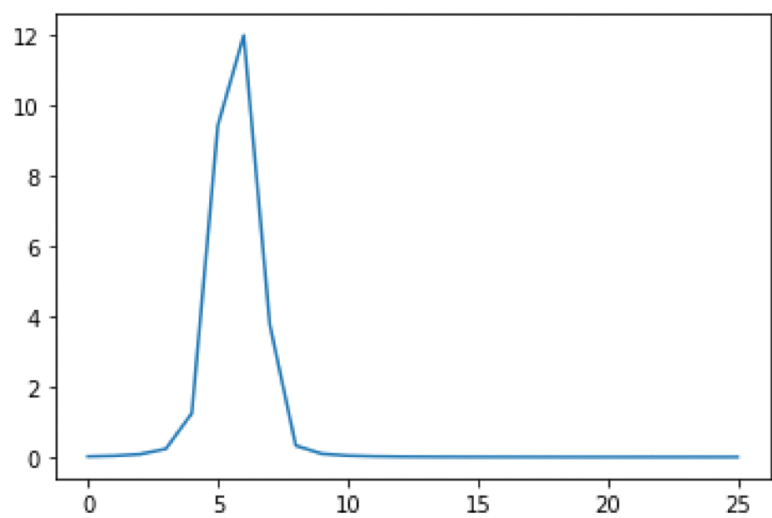


Рис. 1.6: График преобразования Фурье функции с разрывом

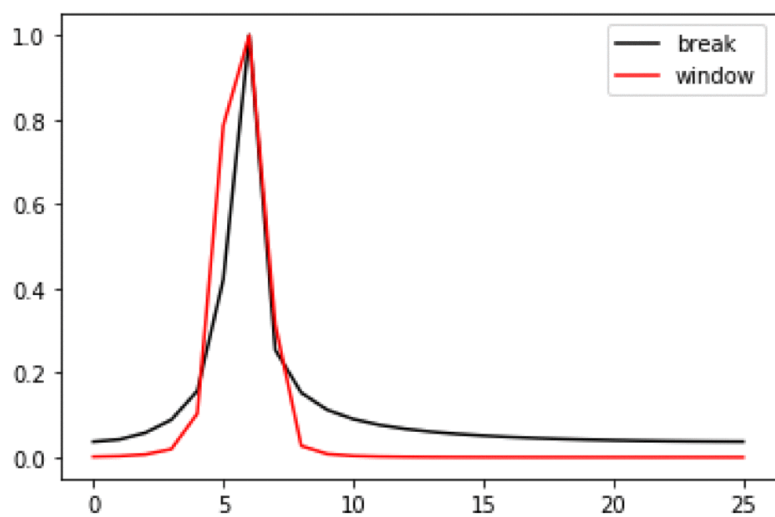


Рис. 1.7: Подавление фантомных частот после сглаживания

Это означает, что для лучшей оценки исходного спектра в выбранной области частот функция $W(w)$ должна быть похожа на дельта-функцию. В этой связи были предложены и другие окна, но они оказались полезными только в некоторых приложениях.

1.5.6 Связь между спектром непрерывного сигнала и ДПФ для периодического сигнала

Как уже отмечалось выше, с помощью ДПФ в общем случае получается только оценка спектра. Истинный спектр определяется разложением сигнала в ряд Фурье.

Пусть периодическая на $[0,1]$ функция задана формулой

$$f(x) = \sum_{n=-\infty}^{\infty} f_n e^{2\pi j n x}$$

Выберем N и найдем ДПФ, используя значения функции в точках m/N , $m = 0, 1, \dots, N-1$. Напомним, что символ $N|p-m$ означает, что N делится на $p-m$. Теперь

$$f(m/N) = \sum_{n=-\infty}^{\infty} f_n e^{2\pi j n m/N},$$

$$F(k) = \sum_{m=0}^{N-1} f(m/N) e^{-2\pi j k m/N} = \sum_{n=-\infty}^{\infty} f_n \sum_{m=0}^{N-1} e^{2\pi j m(n-k)/N}.$$

Внутренняя сумма есть сумма геометрической прогрессии. В ней останутся лишь те слагаемые, для которых $n-k|N$, другими словами, $n = k + pN$. Отсюда следует, что коэффициент $F(k)$ в формуле (1.15), найденный по последовательности $f(m/N)$, $m = 0, 1, \dots, N-1$, равен

$$F(k) = N \sum_{p=-\infty}^{\infty} f_{k+Np} \quad (1.17)$$

Этот эффект называют эффектом подмены частот, поскольку вместе с ожидаемой частотой в этот коэффициент вносят вклад и другие частоты. Таким образом, если нужно оценить коэффициенты Фурье f_k с помощью $F(k)$, нужно обеспечить отсутствие или малые значения для f_{k+Np} , $p \neq 0$.

Рассмотрим пример

```
N=512
ArgCom = np.arange(N)/N*2*np.pi
Freq1 = 32
Sin1 = np.sin(ArgCom*Freq1)
Freq2 = 40 + N
```

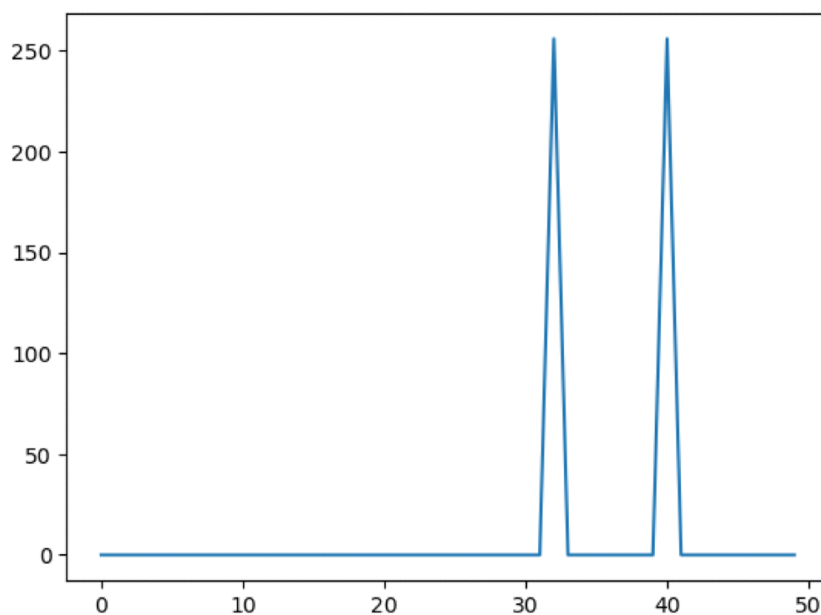



Рис. 1.8: Пример подмены частот

```
Sin2 = np.sin(ArgCom*Freq2)
Fun = Sin1 + Sin2
Spect = np.fft.fft(Fun)[:N//2]
plt.plot(range(50),abs(Spect[range(50)]))
```

Выполнив данный скрипт, получим два максимума спектра в точках 32 и 40 (см. Рис.1.8).

Непрерывный сигнал, порождающий последовательность, имеет 4 ненулевые точки спектра: $\pm 32, \pm 552$, а значения 40 в этом списке нет. Он возник в результате подмены частот. Из-за эффекта подмены частот при оценке спектра с помощью ДПФ появляется дополнительная частота 40, которая будет прослушиваться при воспроизведении дискретного сигнала. Для нивелирования этого явления перед переходом к дискретному сигналу надо удалить в исходном сигнале все ненужные частоты. Это делается с помощью фильтрации, о которой речь пойдет в дальнейших параграфах.

1.6 Дальнейшие свойства ДПФ

Найдем аналоги свойств преобразований Фурье последовательностей на случай ДПФ. Согласно определению, мы имеем дело с конечными последовательностями вида

$$a_0, a_1, \dots, a_{N-1},$$

однако, всегда предполагается, что эта последовательность бесконечная периодическая с периодом N . Обратим внимание, что преобразования $F(k)$ автоматически получаются периодическими с тем же периодом.

1.6.1 Сдвиг последовательности

Пусть $b_n = a_{n-m}$, $n = 0, 1, \dots, N-1$. Найдем ДПФ от новой последовательности.

$$B(k) = \sum_{n=0}^{N-1} b_n \exp(-2\pi jkn/N) = \sum_{n=0}^{N-1} a_{n-m} \exp(-2\pi jk(n-m+m)/N)$$

Сумма элементов периодической функции по периоду не зависит от начала отсчета. Обозначим $n-m$ за новую переменную. Имеем

$$B(k) = e^{-2\pi jm/N} \sum_{n=0}^{N-1} a_n \exp(-2\pi jkn/N) = e^{-2\pi jm/N} F(k). \quad (1.18)$$

1.6.2 Свертка последовательностей

Свертка двух последовательностей f_n, g_n задается формулой

$$h_n = f_n * g_n = \sum_{k=0}^{N-1} f_{n-k} g_k.$$

Найдем ДПФ от свертки и поменяем порядок суммирования

$$\begin{aligned} H(m) &= \sum_{n=0}^{N-1} h_n \exp(-2\pi jmn/N) = \sum_{k=0}^{N-1} g_k \sum_{n=0}^{N-1} f_{n-k} \exp(-2\pi jm(n-k+k)/N) = \\ &= \sum_{k=0}^{N-1} g_k \exp(-2\pi jmk/N) \sum_{n=0}^{N-1} f_{n-k} \exp(-2\pi jm(n-k)/N). \end{aligned}$$

Это означает, что

$$H(m) = G(m)F(m) \quad (1.19)$$

Обратим внимание, что из доказанной формулы вытекает коммутативность операции свертки. Аналогично доказывается, что преобразование Фурье от произведения последовательностей равно свертке преобразований. Пусть $u_n = f_n g_n$, $n = 0, 1, \dots, N-1$.

$$U(k) = \sum_n u_n \exp(-2\pi jkn/N) = \frac{1}{N^2} \sum_n \exp(-2\pi jkn/N) \cdot$$

$$\sum_m F(m) \exp(2\pi jmn/N) \sum_p G(p) \exp(2\pi jpn/N) =$$

$$\frac{1}{N^2} \sum_{p,m} G(p)F(m) \sum_n \exp(2\pi j n(p+m-k)/N)$$

Внутренняя сумма равна N , если $p+m-k|N$, в остальных случаях эта сумма равна 0. Это означает, во внешней сумме надо оставить слагаемые с p и $m = k - p \bmod N$, то есть справедлива формула

$$u_n = f_n g_n \Leftrightarrow U(k) = \frac{1}{N} F(k) * G(k) \quad (1.20)$$

1.6.3 Корреляция между двумя последовательностями

Дадим очень важные приложения формулы (1.19).

Пусть даны две функции: $f(0), f(1), \dots, f(N-1)$ и $g(0), g(1), \dots, g(M-1)$, со значениями в комплексном поле, $M < N$. Требуется найти в первой последовательности участок, который больше всего похож на вторую последовательность.

Из линейной алгебры известно, что косинус угла между двумя векторами $\alpha = (a_0, a_1, \dots, a_k), \beta = (b_0, b_1, \dots, b_k)$ равен

$$\frac{\sum_n a_n \bar{b}_n}{|\alpha||\beta|}.$$

Если длины векторов совпадают, а косинус угла равен 1, то совпадают и сами векторы. Примерное решение упомянутой задачи сводится к отысканию такого значения n , для которого корреляционная функция

$$c(n) = \sum_{k=0}^{M-1} f(n+k) \bar{g}(k) \quad (1.21)$$

принимает наибольшее значение. Мы говорим о примерном решении, поскольку здесь остается постоянной только длина второго вектора. На практике, когда имеем дело с большим стационарным сигналом, условие примерного постоянства длины отрезка фиксированного размера выполнено для первого сигнала. Чтобы свести ситуацию к случаю обычной свертки, прежде всего, надо сделать обе последовательности равными по длине. Для этого вторую из них дополним нулями таким образом, чтобы она имела длину N . Для функций оставим прежние обозначения и введем функцию $u(k) = \bar{g}(-k)$. Тогда

$$c(n) = \sum_{k=0}^{N-1} f(n+k) u(k) = w(n),$$

где $w(n)$ свертка функций $w = f * u$. Найдем ДПФ от u

$$U(k) = \sum_n u(n) e^{-2\pi j k n / N} = \overline{\sum_n g(-n) e^{-2\pi j k (-n) / N}} = \bar{G}(k)$$

Напомним, что все функции считаются периодическими с периодами N . Хотя $c(n)$ определена для всех n , в задаче имеет смысл рассматривать значения от 0 до $N - M$. Теперь для подсчета корреляции вычисляют преобразование Фурье от f и g , перемножают их $F(k)\bar{G}(K)$, после чего находят обратное преобразование Фурье. Предложенный подход для вычисления свертки имеет смысл лишь в том случае, если имеется быстрый способ вычисления самого преобразования Фурье с помощью функции `fft`.

1.6.4 Примеры вычисления корреляции

Прежде всего, рассмотрим прямое решение задачи.

```
import numpy as np
from matplotlib import pyplot as plt
N = 512
M = 128
Freq = 32
Shift = 25
Arg = 2*np.pi*Freq *np.arange(N)/N
F = np.sin(Arg)
G = F[Shift:Shift+M]
Corr = np.zeros(N - M + 1)
for I in range( N -M +1):
    Part = F[I:I+M]
    Corr[I] = sum(Part*G)
plt.plot(Corr[:30])
```

Исходная функция F является периодической с периодом $N/Freq = 16$, а функция G – интервал значений F . $Shift = 25$ по модулю 16 равен 9. С этим шагом получим максимумы корреляционной функции (см. Рис.1.9).

Существует функция в пакете `numpy`, реализующая те же вычисления.

```
Corr1 = np.correlate(F,G)
print(Corr1[:15])
```

Распечатав массивы $Corr$, $Corr1$, убедимся, что они совпадают. Наконец, реализация с помощью ДПФ

```
GPlus = np.zeros(N)
GPlus[:M] = G
FF = np.fft.fft(F)
FGplus = np.fft.fft(GPlus)
Prod = FF*FGplus.conjugate()
```

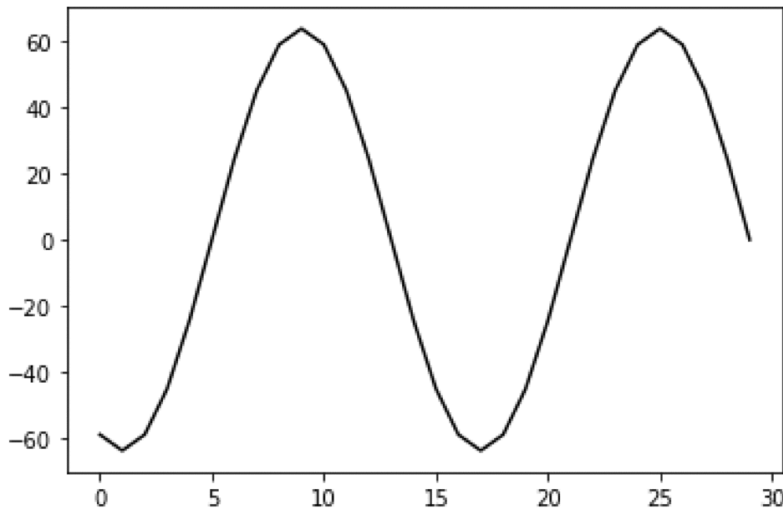


Рис. 1.9: График корреляционной функции

```
Corr2 = np.fft.ifft(Prod).real
print(Corr2[:15])
```

Преобразование Фурье всегда рассматривается как комплексный массив, а свойство `real` представляет вещественную часть комплексного числа. Из постановки задачи следует, что результат должен быть вещественным числом. В результате ошибок округления получается число, содержащее и мнимую часть. Переходя к реальной части результата, получаем правильный ответ.

1.6.5 автокорреляция

Важным примером использования корреляции является автокорреляционная функция (АКФ). Реальный сигнал редко бывает периодическим, но часто он обладает свойством похожим на периодичность. С целью вычисления этого псевдопериода используют АКФ. Для этого посчитывается корреляция функции f со сдвигами этой же функции. В этом случае $g(k) = f(k)$.

$$U(k) = \sum_{x=0}^{N-1} \bar{f}(-x) e^{-2\pi j k x / N} = \sum_{x=0}^{N-1} \bar{f}(x) e^{2\pi j k x / N}$$

поэтому

$$\bar{U}(k) = \bar{F}(k)$$

Это означает, что значения АКФ получаются с помощью обратного преобразования Фурье от последовательности $|F(k)|^2$. Рассмотрим пример.

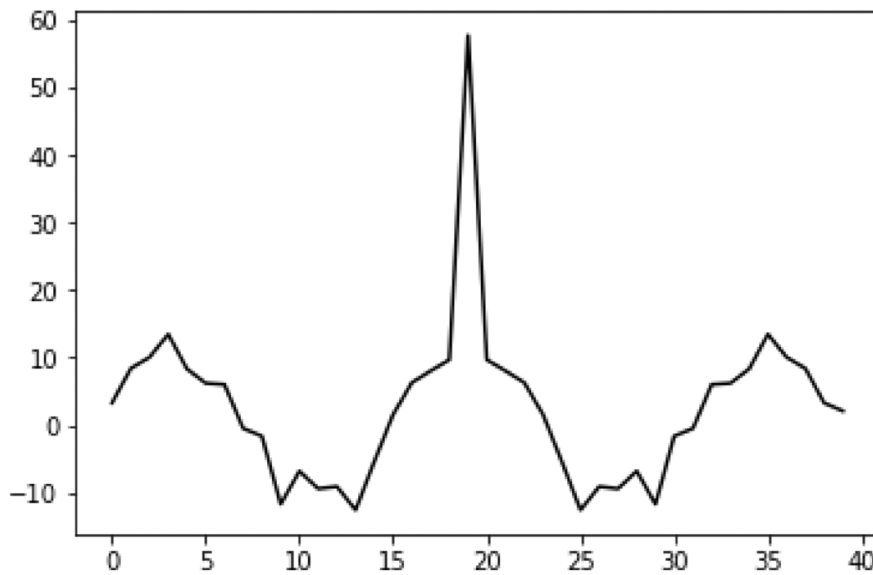


Рис. 1.10: График атокорреляционной функции

```
import numpy as np
N = 512
Freq = 32
Arg = 2*np.pi*Freq *np.arange(N)/N
F = np.sin(Arg)
Ran = np.random.random(N) - .5
Sum = .2*F+Ran
Corr3 = np.correlate(Sum,Sum,mode = 'full')
plt.plot(Corr3[N-20:N+20], 'k')
```

Функция *Sum* не будет периодической из-за наличия случайных слагаемых из массива *Ran*, однако, какая-то периодичность с шагом 16 должна наблюдаться. Опция 'full' означает следующее. Копия массива *Sum* длины *N* движется по оригиналу в обе стороны до тех пор, пока есть непустое пересечение, и производится вычисление скалярного произведения, дополняя недостающие элементы нулями. В результате получается массив длины $2N - 1$, в котором максимальный элемент находится в позиции $N - 1$. Все остальные локальные максимумы отстоят от главного максимума с шагом примерно равным 16. Поясним некоторые детали. Функция `np.random.random(N)` порождает псевдослучайную последовательность равномерно распределенную на интервале $[0,1]$ (см. Рис.1.10). Если требуется произвести вычисления автокорреляции самостоятельно с помощью `fft`, то для получения правильного результата надо добавить к исходному массиву нулевой массив той же длины.

1.7 Восстановление функции, заданной на всей оси, по ее значениям в дискретном множестве точек

Рассмотрим следующую задачу. Имеется функция $x(t)$, заданная на всей вещественной оси. Подсчитываем значения функции с некоторым шагом T , в результате чего получаем последовательность $y[n] = x(nT)$. Возникает вопрос, можно ли восстановить функцию во всех точках, зная только последовательность $y[n]$. Из примера с подменой частот следует, что это можно сделать не всегда. Ниже будет доказан фундаментальный результат, носящий название 'Теорема Котельникова-Шеннона', который дает ответ на вопрос в одном случае.

Пусть дано

$$X(w) = \int_{-\infty}^{\infty} x(t)e^{-2\pi j w t} dt$$

преобразование Фурье от исходной функции, причем функция $X(w)$ имеет компактный носитель – $|w| \geq M \Rightarrow X(w) = 0$ для некоторого M . Тогда при выполнении неравенства (1.22)

$$T < \frac{1}{2M} \quad (1.22)$$

могут быть восстановлены значения функции f во всех точках.

Другими словами, частота выборки должна быть в два раза больше максимальной частоты, присутствующей в сигнале. Это условие еще называют частотой Найквиста. Обратим внимание, что неравенство (1.22) нельзя ослабить. Действительно, пусть $x(t) = \sin(2\pi t) = \frac{1}{2j}(e^{2\pi j t} - e^{-2\pi j t})$. Это означает, что спектр функции сосредоточен в двух точках $w = \pm 1$, то есть $M = 1$ (вспомним, как подсчитывается преобразование Фурье от функции $e^{2\pi j t}$). Если заменить в (1.22) знак неравенства на равенство, то получим $T = 1/2$. Очевидно, что в этом случае $y[n]$ – нулевая последовательность, по которой восстановить функцию нельзя.

1.7.1 Преобразование Фурье от единичной последовательности *

Преобразование Фурье от единичной последовательности задается формулой

$$U(w) = \sum_{n=-\infty}^{\infty} \exp(-2\pi j w n)$$

Очевидно, что этот ряд не может сходиться в обычном смысле. Условимся в дальнейшем не указывать в явном виде пределы суммирования

по всем целым числам. Введем функционал

$$d(t) = \sum_n \delta(t - n).$$

Согласно определению, для любой функции $\phi \in K$

$$\int \phi(t) d(t) dt = \sum_n \phi(n).$$

В этих обозначениях

$$U(w) = \int d(t) \exp(-2\pi j w t) dt.$$

Покажем, что преобразованием Фурье $d(t)$ является сама эта функция, то есть

$$d(t) \Leftrightarrow d(w). \quad (1.23)$$

Действительно, как обычно, положим

$$U_N(w) = \int \sum_{n=-N}^N \delta(t - n) \exp(-2\pi j w t) dt = \sum_{n=-N}^N \exp(-2\pi j w n).$$

Обратим внимание на то, что это периодическая функция с периодом 1, представленная суммой геометрической прогрессии. Имеем:

$$U_N(w) = \frac{\exp(-2\pi j N w) - \exp(2\pi j (N + 1) w)}{1 - \exp(2\pi j w)}.$$

Умножим числитель и знаменатель на $e^{-\pi j w}$. Получим

$$U_N(w) = \frac{\sin(2\pi(N - 1/2)w)}{\sin(\pi w)}.$$

В окрестности 0 $\sin(\pi w)$ можем заменить на πw , поэтому имеем

$$\int_{-c}^c \phi(w) U_N(w) dw = \int_{-c}^c \frac{\phi(w) \sin(2\pi(N - 1/2)w)}{\pi w} dw \rightarrow \phi(0), N \rightarrow \infty,$$

где c любое малое положительное число. Таким образом, в окрестности 0 $U_N(w) \rightarrow \delta(w)$. Очевидно, что

$$\int_c^{1-c} U_N(w) \phi(w) dw \rightarrow 0, N \rightarrow \infty.$$

Это означает, что на интервале $[-c, 1-c]$ $U_N(w) \rightarrow \delta(w)$, $N \rightarrow \infty$. По определению в нашем случае преобразование Фурье есть периодическая

функция с периодом 1. Зная ее на интервале $[0,1]$, можем продолжить эту функцию на всю ось.

$$d(t) = \sum \delta(t - n) \Leftrightarrow U(w) = \sum_k \delta(w - k). \quad (1.24)$$

Для отыскания преобразования для произвольного T вспомним равенство $\delta(Tx) = \delta(x)/T$ и воспользуемся соотношением

$$\sum \delta(t - nT) = \sum \delta(T(\frac{t}{T} - n)) = \frac{1}{T} d\left(\frac{t}{T}\right) \Leftrightarrow$$

продолжим

$$\frac{1}{T} T \sum_k \delta(wT - k) = \sum_k \delta(T(w - \frac{k}{T})) = \frac{1}{T} \sum_k \delta(w - \frac{k}{T}).$$

Итак, доказана формула

$$\sum \delta(t - nT) \Leftrightarrow \frac{1}{T} \sum_k \delta(w - \frac{k}{T}). \quad (1.25)$$

1.7.2 Восстановление функции по дискретным значениям

Пусть $u_T(t) = \sum_n \delta(t - nT)$. Преобразование Фурье от последовательности дискретных значений запишем в виде

$$Y(w) = \sum_n x(nT) e^{-2\pi j n T w} = \int x(t) u_T(t) e^{-2\pi j w t} dt,$$

поэтому обратное преобразование от $Y(w)$ можно представить в форме

$$\int Y(w) e^{2\pi j w t} dw = x(t) u_T(t). \quad (1.26)$$

Используя формулу (1.23) и преобразовании Фурье от произведения функций, получаем:

$$Y(w) = U_T(w) * X(w) = \frac{1}{T} \int_{-\infty}^{\infty} \sum_k \delta(v - \frac{k}{T}) X(w - v) dv$$

откуда вытекает

$$Y(w) = \sum_k X\left(w - \frac{k}{T}\right) / T \quad (1.27)$$

Эта формула устанавливает связь между преобразованиями Фурье от функции и последовательности. Как и следовало ожидать, $Y(w)$ имеет период $1/T$.

Предположим, что спектр $X(w)$ исходного сигнала ограничен числом M , то есть носитель находится внутри интервала $[-M, M]$. Выберем число T таким образом, чтобы носители (точки, где функция может отличаться от 0,) разных слагаемых в правой части (1.27) не пересекались. При $k = 0$ носитель находится в интервале $[-M, M]$. При $k = 1$ носитель находится в интервале $[1/T - M, 1/T + M]$. Чтобы носители не пересекались, необходимо выполнение неравенства: $M < 1/T - M$, или (1.22). В этом случае функция $X(w)$ однозначно определяется функцией $Y(w)$ в равенстве (1.27). Для любого w значение функции $Y(w)$ зависит лишь от одного слагаемого в правой части (1.27). Величина $2M$ называется частотой выборки Найквиста. Если частота выборки больше указанной величины, то спектр непрерывного сигнала может быть восстановлен по спектру последовательности. С другой стороны, функция восстанавливается по своему преобразованию Фурье. Это означает, что при выполнении условия (1.22), когда спектр функции имеет носитель в интервале $(-M, M)$, сама функция может быть восстановлена по своим значениям в точках $x(nT)$. Это и есть теорема Котельникова – Шеннона, играющая исключительно важную роль в цифровой обработке сигналов.

Найдем формулу, восстанавливающую функцию по ее дискретным значениям. При выполнении (1.22) носитель $X(w)$ лежит в интервале $(-1/2T, 1/2T)$ и этот носитель не пересекается с носителями других слагаемых. Положим $H(w) = 1, |w| < 1/2T$ и 0 в остальных точках. Обратное преобразование Фурье от этой функции имеет вид

$$h(t) = \int_{-1/2T}^{1/2T} e^{2\pi i w t} dw = \frac{\sin(t\pi/T)}{\pi t}$$

Согласно (1.27), $X(w) = TY(w)H(w)$. Переходя к обратным преобразованиям и свертке используя (1.26), получим

$$\begin{aligned} x(t) &= T \int_{-1/2T}^{1/2T} Y(w)H(w)e^{2\pi i w t} dw = \\ Th(t) * \sum x(t)\delta(t - nT) &= T \sum x(nT)h(t - nT) \end{aligned} \quad (1.28)$$

Формула (1.28) позволяет восстановить функцию по ее значениям в точках при сделанных предположениях о спектре и частоте выборки. Эта формула имеет, в основном, теоретическое значение, поскольку для получения значения в одной точке t надо знать величины $x(nT)$ для всех целых n . Для проверки правильности (1.28) можно ограничиться конечным числом значений исходной функции. Тогда вместо бесконечной суммы получим конечное выражение, которое можно сравнить с правильным результатом. Рассмотрим пример. Функция $\sin(2\pi t) = (e^{2\pi j t} - e^{-2\pi j t})/2j$, поэтому носитель спектра состоит из двух точек ± 1 .

`import numpy as np`

```

N = 300
T = 0.1
Arg = np.arange(-N*T, N*T, T)
Fun = np.sin(2*np.pi*Arg)
Pnt = 0.632
NewArg = Pnt - Arg
H = np.sin(NewArg*np.pi/T)/(NewArg *np.pi)
Val = T*sum(Fun*H)
print(Val, np.sin(2*np.pi*Pnt))

```

Увеличивая шаг T , мы уменьшаем число слагаемых в сумме, поэтому точность вычисления по формуле (1.28) уменьшается. Скачок происходит при $T \geq 0.5$ то есть при нарушении условия (1.22). Формула (1.22) имеет теоретическое значение. На практике придется ограничиться конечным числом слагаемых. Если нужно восстановить исходную функцию по дискретными значениям, обычно используют какую-либо интерполяцию. Соответствующий пример будет приведен позже.

1.7.3 Шум при переходе к целому формату в процессе оцифровки

Как отмечалось ранее, в процессе оцифровки аналогового сигнала $x(t)$ иногда переходят к представлению значений сигнала в виде целых чисел. Это нельзя сделать без искажений, и ниже приводится теоретический результат, позволяющий оценить возникающие искажения.

Пусть исходный дискретный сигнал $x[n] = x(nT)$ удовлетворяет условию $|x[n]| \leq M$, а для его оцифровки выбраны B двоичных разрядов. Каждый из 2^B интервалов имеет длину $d = 2M/2^B$. Обозначим через $y[n]$ ближайшую к $x[n]$ границу интервала. В результате получаем $x[n] = y[n] + e[n]$, где $e[n]$ - ошибка оцифровки. Для оценки влияния данной процедуры на качество сигнала рассмотрим простейшую модель, согласно которой исходный сигнал является случайным с дисперсией σ_x^2 , а $e[n]$ имеет равномерное распределение на интервале $[-d/2, d/2]$. Ошибка имеет нулевое среднее, а дисперсия этого равномерного сигнала вычисляется по формуле

$$\frac{1}{d} \int_{-d/2}^{d/2} t^2 dt,$$

то есть дисперсия $\sigma_e^2 = d^2/12$. Для оценки влияния шума на сигнал принято использовать отношение сигнал-шум, величину

$$sn = 10 \log_{10} \left| \frac{\sigma_x^2}{\sigma_e^2} \right|, \quad (1.29)$$

измеренную в децибелах (db). Полагая $M = 4\sigma_x$ (согласно неравенству Чебышева, вероятность превысить эту границу мала), получим

$$d = \frac{8\sigma_x}{2B}, sn = 20B \log_{10}(2) + 10 \log_{10}(3/16) \approx 6B - 7.2.$$

В результате экспериментов было показано, что величина $sn = 90db$ обеспечивает высокое качество звука, поэтому значение $B = 16$ используется в большинстве бытовых звуковых карт. Заметим, что приведенное выражение для отношения сигнал-шум не является единственно возможным. Иногда вместо средне квадратических отклонений используют максимальные значения сигнала и шума.

Проведем экспериментальную проверку найденной формулы. В качестве сигнала выберем синус. Для того, чтобы удобно было пользоваться встроенной функцией *round*, выберем $d = 1$, сдвинем график вверх, а функцию масштабируем так, чтобы она заполняла весь интервал возможных значений:

```
import numpy as np
B = 16
d = 1
Freq = 64
N = 1024
Span = d * (2**B - 1)
Arg = 2*np.pi*Freq/N
Sin = (np.sin(Arg*np.arange(N))+1)*Span/2
Sigma_x_2 = np.var(Sin)
Round = np.round(Sin)
Error = Sin - Round
Sigma_e_2 = np.var(Error)
S_N = 10 * np.log10(Sigma_x_2/Sigma_e_2)
print(S_N)
```

Проведите вычисления для разных значений B и сравните найденные значения с величинами, полученными теоретически.

В результате экспериментов была найдена минимальная мощность P_0 звукового сигнала, который может быть услышан человеком. На самом деле, эта величина зависит не только от человека, но и от частоты этого звука, однако такая условная величина принята в технике измерений. Когда говорится, что звук мощностью P более 100 децибел может нанести вред человеку, имеется в виду что

$$10 \log_{10} \frac{P}{P_0} > 100.$$

Глава 2

Фильтрация

2.1 Линейные фильтры

Фильтрация является одним из наиболее распространенных методов обработки сигналов. Ее смысл заключается в изменении спектра сигнала нужным образом. При этом отдельные области спектра усиливаются, а другие ослабляются. Самый простой пример. Пусть наблюдаемый сигнал $y(t) = x(t) + n(t)$ есть сумма полезного сигнала $x(t)$ и шума $n(t)$ и от шума надо избавиться. Предположим, что спектры двух последних различаются. В этом случае создается фильтр, который усиливает спектр $x(t)$ и ослабляет спектр $n(t)$. Мы ограничимся ситуацией, когда наблюдаемый сигнал задан в виде последовательности $y[0], y[1], \dots$, а фильтрация реализуется с помощью свертки наблюдаемого сигнала со специальной последовательностью $h[0], h[1], \dots$, которая, вообще говоря, является бесконечной. Назовем последовательность $z[n]$ результатом фильтрации последовательности $y[n]$ с помощью последовательности $h[n]$ (см. 2.1).

$$z = y * h \quad (2.1)$$

Назовем такой фильтр линейным. В дальнейшем под словом фильтр всегда понимается линейный фильтр. Переходя к преобразованиям Фурье, получим (1.13)

$$Z(w) = Y(w)H(w) \quad (2.2)$$

Формула (2.2) является основой теории линейных фильтров. Она интерпретируется следующим образом. Для каждого значения w_0 спектр исходного сигнала $Y(w_0)$ заменяется новым значением $Y(w_0)H(w_0)$. Таким образом появляется возможность усиливать или ослаблять отдельные частоты в исходном сигнале. Теперь задача сводится к построению такой последовательности $h[n]$, преобразование Фурье которой имеет нужный вид. Из свойств свертки вытекают важные свойства фильтров:

1. Линейность. Пусть y_1, y_2 есть результат фильтрации последовательностей x_1, x_2 соответственно одним и тем же фильтром, тогда

результат фильтрации $A_1 \cdot x_1 + A_2 \cdot x_2$ есть последовательность $A_1 \cdot y_1 + A_2 \cdot y_2$.

2. Инвариантность. Если y есть результат фильтрации последовательности x , то результат фильтрации сдвинутой последовательности x^p имеет вид y^p .

Последовательность $h[n]$ называется функцией отклика фильтра. Она возникает на выходе фильтра, когда на его вход подается сигнал вида $1, 0, 0, \dots$. Преобразование Фурье от этой последовательности называется передаточной функцией фильтра. Если последовательность $h[n]$ имеет лишь конечное число элементов отличных от нуля, то в этом случае вычисление свертки не вызывает затруднений. В противном случае возникает проблема устойчивости.

2.1.1 Устойчивость фильтра

Чтобы свертка бесконечных последовательностей имела смысл, надо ввести дополнительные ограничения.

Определение 1 *Фильтр называется устойчивым, если он переводит любую ограниченную последовательность в ограниченную.*

Покажем, что фильтр устойчив тогда и только тогда, когда

$$\sum_n |h[n]| < \infty \quad (2.3)$$

Сначала докажем достаточность. Пусть входная последовательность ограничена по модулю числом M . Результат свертки

$$|z[n]| = \left| \sum_k x[n-k]h[k] \right| \leq M \sum_k |h[k]|,$$

то есть все выходные значения ограничены одним числом.

Обратно, пусть фильтр устойчив. Последовательность $h[n]$ есть результат фильтрации ограниченной входной последовательности, состоящую из одной единицы и нулей. Из устойчивости фильтра следует ограниченность последовательности $h[n]$. Возьмем в качестве входной последовательности $x[n] = \bar{h}[-n]/|h[-n]|$, если $h[n] \neq 0$. Реакция в нуле на эту последовательность имеет вид

$$y[0] = \sum_k x[-k]h[k] = \sum_k |h[k]| < \infty,$$

то есть (2.3) имеет место.

Из этого доказательства не видно, как вычислять результат фильтрации в случае бесконечной функции отклика. Об этом будет сказано ниже.

2.1.2 Фильтры с конечным временем отклика

Предположим, что в последовательности $h[n]$ лишь конечное число элементов отличны от нуля. В этом случае фильтр называется фильтром с конечным временем отклика (finite impulse response, FIR). Другими словами, результат фильтрации есть

$$z[n] = \sum_{k=L}^N h[k]y[n-k].$$

Переходя к преобразованиям Фурье и учитывая (2.2), получим, что

$$Z(w) = \sum_{k=L}^N h[k]e^{-2\pi jkw}Y(w).$$

Другими словами, передаточная функция фильтра имеет вид

$$H(w) = \sum_{k=L}^N h[k]e^{-2\pi jkw}.$$

Заметим, что такой фильтр всегда устойчив. Конечное время отклика в названии фильтра указывает на тот факт, что изменение в одном члене входной последовательности будет замечено лишь в конечном числе членов выходной последовательности.

2.1.3 Фильтры с бесконечным временем отклика

Фильтром с бесконечным временем отклика (infinite impulse response, IIR) называется фильтр, в функции отклика которого бесконечное число членов отличны от нуля. В общем случае результат фильтрации таким фильтром подсчитать нельзя, однако, есть ситуация, когда это сделать можно. Это фильтры, определенные с помощью рекуррентного соотношения (2.4).

$$z[n] = - \sum_{k=1}^M a_k z[n-k] + \sum_{k=L}^N b_k y[n-k], \quad (2.4)$$

где M - натуральное (оно называется порядком фильтра), а L, N - любые целые числа. Формула (2.4) позволяет подсчитать значения $z[n]$, если известны начальные значения этой последовательности. В данном случае это удастся сделать, поскольку полагаются нулевыми значения функции $z[n]$ для отрицательных значений индекса. Обычно полагают, что и входная последовательность $y[n]$ обладает свойством: $y[m] = 0, m < n_0$ для некоторого n_0 . Переходя в (2.4) к преобразованиям Фурье и вспоминая вид преобразования Фурье от сдвинутой последовательности, получим

$$Z(w) = - \sum_{k=1}^M a_k e^{-2\pi jkw} Z(w) + \sum_{k=L}^N b_k e^{-2\pi jkw} Y(w),$$

или

$$\begin{aligned} Z(w) &= H(w)Y(w) \\ H(w) &= \frac{\sum_{k=L}^N b_k e^{-2\pi jkw}}{1 + \sum_{k=1}^M a_k e^{-2\pi jkw}} \end{aligned} \quad (2.5)$$

Это означает, что в этом случае последовательность $h[n]$ в (2.1) получается путем обратного преобразования Фурье, примененного к функции $H(w)$.

Функции отклика $h[n]$ имеет бесконечное число ненулевых элементов, хотя и не может быть произвольной сходящейся последовательностью. Функцию отклика можно найти формальным представлением $H(w)$ в виде ряда. Для этого надо вспомнить формулу для суммы бесконечной геометрической прогрессии

$$\frac{1}{1-c} = 1 + c + c^2 + c^3 + \dots$$

Нужная функция отклика имеет вид $U(1 - V + V^2 - V^3 + \dots)$ где

$$U = \sum_{k=L}^N b_k e^{-2\pi jkw}, \quad V = \sum_{k=1}^M a_k e^{-2\pi jkw}$$

с последующим суммированием коэффициентов при одинаковых степенях $e^{-2\pi jw}$. Численные значения функции отклика можно также найти интегрированием,

$$h[k] = \int_0^1 H(w) e^{2\pi jkw} dw.$$

Проблема устойчивости рассмотренного фильтра будет рассмотрена ниже.

Для указанного вида фильтров изменение в одном члене входной последовательности формально будет наблюдаться во всех членах последующей выходной последовательности. На практике эти изменения можно будет наблюдать лишь в нескольких членах выходной последовательности, следующих за измененной входной последовательностью. Поэтому говорят о времени установления фильтра. В остальных членах изменения незаметны вследствие ограниченной точности представления чисел. Ниже это будет показано на примере.

2.1.4 Соединение фильтров

Если имеются два фильтра, то их можно соединить последовательно или параллельно. Пусть это фильтры с функциями отклика h_1 и h_2 . Их реакции на входную последовательность y обозначим через z_1 и z_2 соответственно — $z_1 = y * h_1$, $z_2 = y * h_2$. Из определения передаточной

функции вытекает, что при параллельном соединении фильтров выход $z = y * (h_1 + h_2)$, а при последовательном соединении $z = z_1 * h_2 = h_1 * h_2 * y$ в силу коммутативности и ассоциативности операции свертки это означает, что передаточные функции при параллельном соединении складываются, а при последовательном соединении фильтров перемножаются. Отсюда следует интересное свойство – не имеет значения порядок, в котором эти фильтры применяются к y при последовательном соединении.

2.1.5 Устойчивость фильтров с бесконечным временем отклика

Начнем с простейших IIR фильтров – фильтров первого порядка. Рассмотрим фильтр вида

$$z[n] = -cz[n-1] + y[n]$$

Его передаточная функция имеет вид

$$H(w) = \frac{1}{1 + ce^{-2\pi jw}} \quad (2.6)$$

Исследуем вопрос устойчивости фильтра. Найдем функцию отклика фильтра, подав на его вход последовательность $1, 0, 0, \dots$, либо разложим в ряд дробь (2.6). Это ряд

$$H(w) = 1 - cG + c^2G^2 - c^3G^3 + \dots, \quad G = e^{-2\pi jw}.$$

Легко видеть, что последовательность $|h[n]| = |c|^k$ будет удовлетворять условию (2.3) тогда и только тогда, когда $|c| < 1$. В общем случае обозначим через $H_1(w)$ знаменатель в формуле (2.5).

$$H_1(w) = 1 + \sum_{k=1}^M a_k G^k = G^M P(Z), \quad P(Z) = Z^M + \sum_{k=1}^M a_k Z^{M-k}, \quad Z = G^{-1}.$$

Разложим многочлен $P(Z) = (Z - c_1)(Z - c_2) \cdots (Z - c_M)$, используя корни этого многочлена. Формулу (2.5) представим в виде

$$H(w) = \sum b_k G^k \frac{1}{1 - c_1 G} \cdot \frac{1}{1 - c_2 G} \cdots \frac{1}{1 - c_M G}. \quad (2.7)$$

Это означает, что фильтр можно рассматривать как последовательное соединение FIR фильтра и M фильтров первого порядка. Таким образом, достаточным условием устойчивости IIR фильтра является неравенство $|c_k| < 1$ для всех корней многочлена $P(Z)$. Поясним содержательный смысл формулы (2.7). Все корни многочлена должны находиться внутри единичной окружности с центром в нуле. Когда $G = e^{-2\pi jw}$,

двигаясь по единичной окружности, приближается к какому-нибудь корню, значение передаточной функции увеличивается. Функция в числителе дроби $H(w)$ в (2.5) подбирается так, чтобы эта функция была близкой к нулю в нужных точках. Эти соображения лежат в основе теории проектирования фильтра с нужными свойствами.

Многочлен $P(Z)$ может иметь комплексные корни. Имея дело с вещественными сигналами, желательно избегать комплексных чисел. Изначально числа a_k, b_k в (2.4) являются вещественными, поэтому и коэффициенты $P(Z)$ будут таковыми. Отсюда следует, что вместе с комплексным корнем c_k число $\bar{c}_k$ также будет корнем этого многочлена. Это значит, что

$$\frac{1}{1 - c_k G} \frac{1}{1 - \bar{c}_k G}$$

есть передаточная функция вещественного ИИР фильтра второго порядка. Таким образом, любой вещественный ИИР фильтр можно реализовать в виде последовательного соединения вещественного ФИР фильтра и вещественных ИИР фильтров первого и второго порядка. Применяя те же соображения к числителю функции $H(w)$, можем заключить, что его можно представить в виде последовательного соединения ФИР фильтров первого и второго порядка. Это важное замечание оказывается полезным в случае, когда нужный фильтр реализуется в 'железе'.

Из курса алгебры известно, что любая рациональная дробь вида $R(z) = Q_1(z)/Q_2(z)$, где Q_1, Q_2 – многочлены, Q_2 не имеет кратных корней, может быть представлена в виде

$$R(z) = Q_1(z) \sum_k \frac{d_k}{z - c_k},$$

где суммирование производится по всем корням многочлена $Q_2(z)$. Это означает, что вместо последовательного соединения фильтров можно часть фильтров второго порядка соединить параллельно. Это важно иметь в виду, когда точность вычисления на ограниченном ресурсе является существенным фактором. Это предмет самостоятельной теории, и здесь мы ограничимся только этим замечанием.

Обратим внимание, что при рассмотрении свойств передаточной функции $H(w)$ мы перешли в область комплексных чисел. Это распространенный прием при создании фильтров с нужными свойствами. Мы не рассматриваем эту теорию, поскольку в настоящее время существуют доступные реализации большинства используемых фильтров в пакете `scipy`.

2.1.6 Случай конечной последовательности

Предыдущий материал предполагал, что исходная последовательность бесконечная. На практике мы имеем дело с конечными последовательно-

стями, и под преобразованием Фурье понимается дискретное преобразование. Как было показано ранее, в случае конечных последовательностей всегда предполагается их периодичность. В этой связи, остается справедливой формула (2.2), только теперь преобразование Фурье заменяется на ДПФ. На первый взгляд, для конечных последовательностей нет проблем с устойчивостью фильтра, однако, это не так. Используя формулу фильтрации с неустойчивым фильтром, мы, как правило, быстро получаем значения для отфильтрованной последовательности, которые превышают границу чисел, представимых в компьютере.

2.1.7 Фильтры низких и высоких частот. Идеальный фильтр

В результате фильтрации спектр сигнала умножается на передаточную функцию фильтра. Фильтры условно делятся на два класса: фильтры низкой и фильтры высокой частот. Есть и другие фильтры, но они реализуются в виде комбинаций этих двух. Отметим полосовой фильтр, пропускающий сигналы только из заданной полосы частот и стоп фильтр, подавляющий сигналы из заданной полосы частот. Основным параметром фильтра является частота отсечения (*cutoff*). В идеале фильтр низкой частоты пропускает без изменения все сигналы, частота которых ниже *cutoff* и подавляет все сигналы, частота которых больше *cutoff*. Фильтр высоких частот является зеркальным. Если $Low(w)$ передаточная функция идеального фильтра низких частот, то она должна обладать свойством :

$$Low(w) = \begin{cases} 1, & w \leq cutoff, \\ 0, & w > cutoff. \end{cases}$$

Предположим, что такой фильтр можно реализовать с помощью некоторой вещественной функции отклика $h[n]$

$$Low(w) = \sum_{k=-\infty}^{\infty} h[k] e^{-2\pi j k w}.$$

Легко видеть, что $Low(-w) = \bar{Low}(w)$, поэтому $Low(w) = 1, |w| \leq cutoff$ и 0 вне этой области. Теперь

$$h[k] = \int_{-cutoff}^{cutoff} e^{2\pi j k w} dw$$

и $h[n]$ имеет бесконечное число ненулевых значений как для положительных, так и для отрицательных значений аргумента. Это означает, что такой фильтр реализовать невозможно, поскольку чтобы найти результат фильтрации в момент времени t_0 нужно знать значения сигнала во все моменты $t > t_0$, то есть после этого момента. Однако имея дело с конечными сигналами, можно реализовать идеальный фильтр методами

ДПФ. Если $Low(w)$ передаточная функция идеального фильтра низких частот, то она должна обладать свойством :

$$Low(w) = \begin{cases} 1, & w \leq cutoff, \\ 0, & w > cutoff. \end{cases}$$

Предположим, что такой фильтр можно реализовать с помощью некоторой вещественной функции отклика $h[n]$

$$Low(w) = \sum_{k=-\infty}^{\infty} h[k] e^{-2\pi j k w}.$$

Легко видеть, что $Low(-w) = \bar{Low}(w)$, поэтому $Low(w) = 1, |w| \leq cutoff$ и 0 вне этой области. Теперь

$$h[k] = \int_{-cutoff}^{cutoff} e^{2\pi j k w} dw$$

и $h[n]$ имеет бесконечное число ненулевых значений как для положительных, так и для отрицательных значений аргумента. Это означает, что такой фильтр реализовать невозможно, поскольку чтобы найти результат фильтрации в момент времени t_0 нужно знать значения сигнала во все моменты $t > t_0$, то есть после этого момента. Однако имея дело с конечными сигналами, можно реализовать идеальный фильтр методами ДПФ.

2.1.8 Реализация фильтров в пакете `scipy`

Мы не останавливаемся на методах проектирования фильтров, поскольку в настоящее время имеются прекрасные реализации большинства используемых фильтров. Примером такой реализации служит модуль `scipy.signal`. Здесь мы поясним некоторые детали связанные с конечностью входных последовательностей. Предполагается, что фильтр имеет структуру

$$\begin{aligned} a[0]z[n] &= -a[1]z[n-1] - \dots - a[k]z[n-k] \\ &+ b[0]x[n] + b[1]x[n-1] + \dots b[n-m]x[n-m] \end{aligned}$$

(см. 2.4) и задается двумя наборами чисел A, B . Если это FIR фильтр, то $a[0] = 1$, а остальные коэффициенты из A нулевые. Пакет `scipy.signal` содержит функцию `lfilter`. Результат фильтрации сигнала `Sign` с помощью этой функции есть

$$Res = lfilter(B, A, Sign)$$

В случае FIR фильтра $A = 1$. Обратим внимание, что выбор $B=1$ приводит к ошибке, правильный выбор в этом случае $B=[1]$. Результат фильтрации имеет ту же длину, что и исходный сигнал. Для этого исходный сигнал заменяется новым сигналом $[0, \dots, 0, \text{Sign}]$, где число нулей на единицу меньше длины последовательности B . Пример

```
from scipy import signal as SGN
import numpy as np
Sign = np.float_([2,-3,4])
B = np.float_([1,-1])
Res = SGN.lfilter(B,1,Sign)
print(Res)
[ 2. -5.  7.]
```

Исходный сигнал имеет длину 3, фильтр имеет длину 2, поэтому к исходному сигналу слева добавляется 0. Коэффициенты B записываются в обратном порядке $[-1,1]$, получившийся массив движется вдоль сигнала, его элементы накладываются на значения сигнала, перемножаются, а потом складываются. При первом наложении -1 наложится на 0, а 1 на 2. При втором наложении 1 наложится на -2 , а -1 на 2. Предусмотрена ситуация, когда последовательно соединяются два IIR фильтра. В этом случае последние элементы первого фильтра становятся начальными значениями для второго фильтра. В функции `lfilter` имеется опционный параметр `zi`, с помощью которого задается начальное состояние вместо нулевого. Например $Zi = lfilter_{zi}(B, A)$ обеспечивает появление на выходе фильтра $lfilter(B, A, Sign, zi = Zi)$ в начальный момент времени сигнала $Sign[0]$. В общем случае пусть система состоит из двух фильтров, соединенных последовательно. При активной опции `zi` в первом фильтре он выдает не только результат фильтрации, но и новое состояние. Это состояние используется во втором фильтре, присоединенного к выходу данного, что позволяет сгладить скачок на выходе всей системы. Более содержательные примеры рассмотрим в следующем пункте.

2.2 Примеры применения различных фильтров

2.2.1 Пример FIR фильтра

При создании FIR фильтров чаще всего пользуются функцией $B = \text{firwin}(\text{numtaps}, \text{cutoff}, \text{pass_zero} = \text{True}, fs = \text{None})$ (приведены лишь основные параметры). Эта функция вычисляет набор коэффициентов B FIR фильтра. Параметр fs устанавливает желаемую частоту выборки в Гц. Параметр `numtaps` определяет длину фильтра. Пусть $\text{numtaps} = M$. Обычно M выбирается нечетным числом. Первой особенностью функции является то, что последовательность коэффициентов симметрична, $B[k] = B[M - k - 1]$. Ниже будет показано, для

чего накладывают это ограничение. Вместо привычной классификации «фильтр низких частот, фильтр высоких частот, полосовой фильтр» используют другое описание фильтра. Если *pass_zero = True*, (это означает, что нулевая частота проходит через фильтр), то параметры *cutoff* задают фильтр низких частот или stop-band фильтр. Если *cutoff* одно число, то это фильтр низких частот, и *cutoff* задает частоту отсеивания, если *cutoff* есть список из нескольких пар чисел, то это stop-band фильтр, подавляющий частоты из каждой пары списка. Выбрав *pass_zero = False*, получаем фильтр высоких частот или полосовой фильтр в зависимости от того, является ли *cutoff* числом или списком из нескольких пар элементов. С помощью функции *signal.freqz* легко построить передаточную функцию фильтра. Команда имеет следующее описание

```
w, h = freqz(B, A = 1, worN = 512, whole = False, plot = None, fs = 6.28)
```

Здесь *worN=512* – количество точек, по которым рисуется передаточная функция, а *fs* устанавливается равной частоте выборки. Выполнив команду *plot(w, abs(h))*, получаем график передаточной функции. Обратим внимание, что по оси X откладываются значения от 0 до половины частоты выборки.

Пример:

```
B = firwin(17, 1800, fs = 22050),
w, h = freqz(B, fs = 22050),
plot(w, abs(h)).
```

Соответствующий график представлен на Рис.2.1. Как и следовало ожидать, график отличается от графика идеального фильтра. Можно убедиться, что с увеличением длины фильтра график передаточной функции будет больше походить на график передаточной функции идеального фильтра. Все фильтры спроектированы таким образом, чтобы было выполнено условие – в точке *cutoff* значение модуля передаточной функции примерно равно 0.5.

Еще один пример.

Если *B = firwin(17, [600, 1200], pass_zero = False), fs = 22050*, то в результате получаем полосовой фильтр, передаточная функция которого представлена на Рис. 2.2.

В качестве примера IIR фильтра рассмотрим фильтр Баттеруорта из модуля *signal*.

```
[B, A] = signal.butter(N, cutoff, type, fs).
```

Здесь *N* – длина фильтра, *type* принимает значения 'lowpass', 'highpass', 'bandpass' и др., а параметр *cutoff* определен также, как и выше, но для полосового и стоп фильтров возможна только одна пара. Пример.

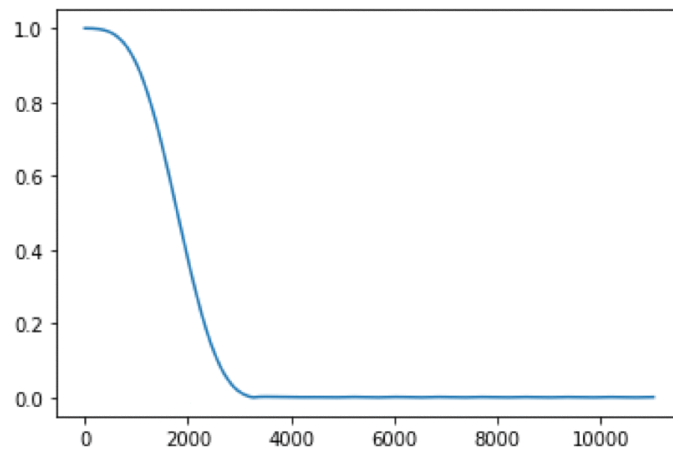


Рис. 2.1: Передаточная функция FIR фильтра низких частот

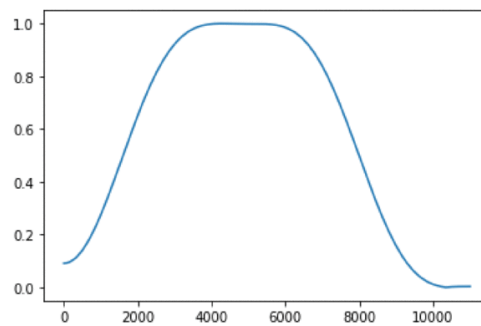


Рис. 2.2: Передаточная функция полосового FIR фильтра

$B, A = \text{butter}(7, 1800, 'lowpass', fs = 22050)$. Для рисования графика передаточной функции фильтра надо сначала выполнить команду $w, h = \text{freqz}(B, A)$. Ниже на Рис.2.3 приведены график передаточной функции этого фильтра. Обратим внимание, что при меньшей длине фильтра передаточная функция фильтра Баттеруорта больше похожа на передаточную функцию идеального фильтра. Это имеет практические последствия. Рассмотрим пример, на котором обнаруживаются дополнительные особенности FIR и IIR фильтров.

```
import numpy as np
from scipy import signal as SGN
from matplotlib import pyplot as plt
Fs = 22050
Fr = 2000
Len = 1024
```

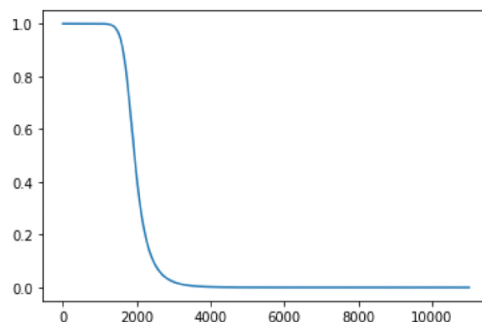


Рис. 2.3: Передаточная функция IIR фильтра низких частот Баттеруорта

```
Arg = np.arange(Len)
Sn = np.sin(2.*np.pi*Fr*Arg/Fs)
B = SGN.firwin(17,1800,fs=Fs)
Res1 = SGN.lfilter(B,1.,Sn)
B,A = SGN.butter(17,1800,'lowpass',fs = Fs)
Res2 = SGN.lfilter(B,A,Sn)
fig,ax = plt.subplots()
ax.plot(Res1[:200],label='fir')
ax.plot(Res2[:200],label='iir')
ax.legend()
```

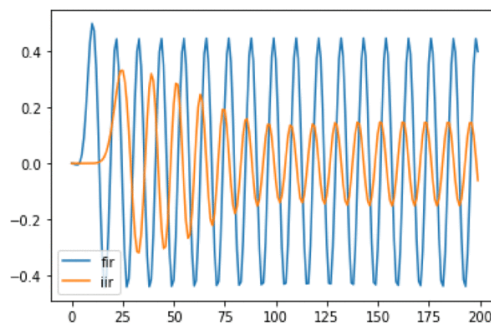


Рис. 2.4: Сравнение эффективности работы FIR и IIR фильтров одинаковой длины

Амплитуда исходного сигнала равна 1. Поскольку его частота превышает значение `cutoff`, в идеале мы должны получить сигнал с нулевой амплитудой. Фактически получили сигналы с положительной амплитудой. Результат представлен на Рис.2.4. IIR фильтр работает более эффективно, однако, присутствует время установления этого фильтра, в

результате чего фильтр начинает работать с некоторым лагом. У второго эффективность хуже, но он начинает работать сразу. Это не единственное преимущество FIR фильтра. О другой важной характеристике симметричного фильтра будет сказано ниже.

2.2.2 Представление IIR фильтра в виде последовательного соединения фильтров первого и второго порядка

Покажем, каким образом реализуется теоретический результат, о котором говорилось в предыдущем пункте. Построим многочлен с корнями $.5, -.8, .7j, -.7j$. Очевидно, что все они лежат внутри единичной окружности. Этот многочлен четвертой степени с коэффициентами

```
import numpy as np
P = np.poly1d([.5, -.8, .7j, -.7j], True)
print(P.c)
[ 1.      0.3      0.09      0.147 -0.196]
```

Используя эти числа, построим IIR фильтр $y[n] = -0.3y[n-1] - 0.09y[n-2] - 0.147y[n-3] + 0.196y[n-4]$. Передаточная функция этого фильтра имеет вид

$$H(w) = \frac{1}{1 + 0.3G + 0.09G^2 + 0.147G^3 - 0.196G^4} X(w), \quad G = e^{-2\pi jw}.$$

Построим график этой функции.

```
from scipy import signal as SGN
from matplotlib import pyplot as plt
B = [1]
A = P.c
w, h = SGN.freqz(B, A)
plt.plot(w, abs(h))
```

Результат представлен на Рис. 2.5.

Покажем, каким образом можно реализовать этот фильтр в виде последовательного соединения нескольких фильтров малого порядка. Многочлен $P(u) = (u - 0.5)(u + 0.8)(u^2 + 0.49)$ Множителям отвечают фильтры $y[n] = 0.5y[n-1]$, $y[n] = -0.8y[n-1]$, $y[n] = -0.49y[n-2]$. Выполнив команды,

```
A1 = [1, -0.5]
A2 = [1, 0.8]
A3 = [1, 0, 0.49]
B=[1]
```

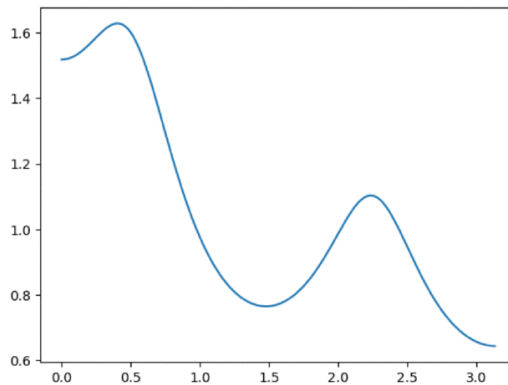


Рис. 2.5: Передаточная функция ИИР фильтра четвертого порядка

```
w,h1 = SGN.freqz(B,A1)
w,h2 = SGN.freqz(B,A2)
w,h3 = SGN.freqz(B,A3)
plt.plot(w,abs(h1*h2*h3))
```

убедимся, что передаточная функция имеет тот же график. В `scipy.signal` имеется функция `sosfilt`, которая реализует идею сведения произвольного фильтра к фильтрам второго порядка.

2.2.3 Восстановление сигнала по отфильтрованному результату

Напомним основную формулу фильтра

$$a_0 y[n] = -a_1 y[n-1] - a_2 y[n-2] - \dots - a_k y[n-k] + b_0 x[n] + b_1 x[n-1] + \dots + b_m x[n-m].$$

Обычно рассматривался случай $a_0 = 1$, но, на самом деле, здесь допустимо любое ненулевое число. Перепишем эту формулу

$$b_0 x[n] = -b_1 x[n-1] - \dots - b_m x[n-m] + a_0 y[n] + a_1 y[n-1] + \dots + a_k y[n-k] \quad (2.8)$$

Мы получили формулу той же конструкции, в которой массивы коэффициентов A, B поменялись местами. Кажется, что задача восстановления исходного сигнала по результатам фильтрации имеет простое решение. Однако здесь имеется проблема — устойчивость фильтра. Для того, чтобы получить решение задачи в указанной форме необходима устойчивость нового фильтра. Для этого достаточно, чтобы корни многочлена $f(z) = b_0 z^m + b_1 z^{m-1} + \dots + b_m$ лежали внутри единичного круга. Существует решение задачи и в том случае, когда это условие не выполнено,

но это решение выглядит более сложно. Достаточно показать, как восстановить последовательность $x[0], x[1], \dots$, по результатам фильтрации $y[0], y[1], \dots$ согласно формуле

$$y[n] = b_0 x[n] + b_1 x[n-1] + \dots + b_m x[n-m].$$

Процедуру фильтрации можно описать следующим образом. Имеются интервалы длины $m+1$ $[b_m, b_{m-1}, \dots, b_1, b_0]$ и $x[n-m], x[n-m+1], \dots, x[n-1], x[n]$, которые накладываются друг на друга. Наложившиеся элементы перемножаются, результаты складываются и получают значение $y[n]$. Построим матрицу $Matr$, первый столбец которой есть $[b_m, b_{m-1}, \dots, b_0]$. Остальные столбцы выберем произвольно с единственным условием – матрица должна быть невырожденной. Построим матрицу U , состоящую из $m+1$ столбцов, в которой строка с номером i имеет вид $x[i], x[i+1], \dots, x[i+m]$. Число строк определяется естественным образом, пока не исчерпается входной массив. Положим

$$D = U \cdot Matr, \quad U = D \cdot Matr^{-1}$$

По построению, первый столбец матрицы U и есть исходный массив. Как уже отмечалось, столбцы матрицы D являются результатом фильтрации входной последовательности различными фильтрами, а для восстановления $x[n]$ достаточно умножения матрицы D на первый столбец матрицы $Matr^{-1}$.

Рассмотрим пример. Пусть $y[n] = (x[n] + x[n-1] - x[n-2] - x[n-3])/2$. Единица является корнем многочлена $f(z) = (z^3 + z^2 - z - 1)/2$, поэтому стандартная процедура восстановления входной последовательности здесь неприменима. Построим матрицу

$$Matr = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

Мы дополнили первый столбец до невырожденной матрицы. Это экземпляр матрицы Адамара, для нее обратная матрица совпадает с транспонированной деленной на 4. Образует входную последовательность 1, 2, ..., 10.

```
Inpt = np.arange(1.,11)
B = [SGN.lfilter([-1.,-1.,1.,1.],1,Inpt),
SGN.lfilter([1.,1.,1.,1.],1,Inpt),
SGN.lfilter([1.,-1.,-1.,1.],1,Inpt),
SGN.lfilter([-1.,1.,-1.,1.],1,Inpt)]
BF = np.float_(B)
InvMatr = np.float_([[1,1,-1,-1],
[1,1,1,1],[1,-1,-1,1],[1,-1,1,-1]])
```

```
Res =1/4* np.dot(BF.T,InvMatr)
print(Res)
```

```
[[ 0.  0.  0.  1.]
 [ 0.  0.  1.  2.]
 [ 0.  1.  2.  3.]
 [ 1.  2.  3.  4.]
 [ 2.  3.  4.  5.]
 [ 3.  4.  5.  6.]
 [ 4.  5.  6.  7.]
 [ 5.  6.  7.  8.]
 [ 6.  7.  8.  9.]
 [ 7.  8.  9. 10.]]
```

Произведение $U \cdot Matr$ заменили фильтрацией. Для этого в функции `lfilter` взяли в качестве первого аргумента столбцы матрицы `Matr`, записанные в обратном порядке. Появление нулей в первых трех столбцах результата объясняется особенностью реализации операции фильтрации. Для получения входной последовательности вместо умножения на всю обратную матрицу достаточно умножить на последний столбец.

2.3 Фаза сигнала

До сих пор все внимание было сосредоточено на модуле передаточной функции фильтра. Как правило, функция $H(w) = e^{2\pi i\theta(w)}|H(w)|$ является комплекснозначной. В этом пункте выясним, как аргумент $\theta(w)$ влияет на свойства фильтра.

2.3.1 Фазовый сдвиг сигнала в результате фильтрации

Пусть имеются сигнал и его преобразование Фурье — $f(t) \Leftrightarrow F(w)$. Применим к сигналу фильтр с передаточной функцией $H(w)$. В результате получим новый сигнал — $g(t) \Leftrightarrow H(w)F(w) = e^{2\pi i\theta(w)}|H(w)|F(w)$. Это означает, что спектр сигнала $g(t)$ получается из спектра сигнала $f(t)$ умножением на вещественное число $|H(w)|$ и на комплексное число $e^{2\pi i\theta(w)}$. Ранее была доказана формула: если $f(t) \Leftrightarrow F(w)$, то $f(t - t_0) \Leftrightarrow \exp(-2\pi i w t_0)F(w)$, другими словами, появление комплексного множителя в передаточной функции ведет к сдвигу отфильтрованного сигнала относительно исходного. Проблема заключается в том, что в общем случае этот сдвиг для каждой частоты будет свой, зависит от $\theta(w)$. Представим, что исходный сигнал представлен в виде ряда Фурье, тогда после фильтрации разные слагаемые будут сдвинуты на разные величины, что ведет к искажению формы сигнала после фильтрации. При

обработке речевых файлов считается, что фазовое искажение не улавливается ухом, поэтому при обработке речевого сигнала этот эффект не учитывается. В то же время, когда важно сохранить форму сигнала после фильтрации, например, сохранить форму импульса, приходится использовать методы компенсации или фильтр с вещественной передаточной функцией. Проще всего это сделать, применив симметричный FIR фильтр. Пусть

$$y[n] = \sum_{k=0}^{2N} b_k x[n-k], b_k = b_{2N-k}.$$

Передаточная функция

$$\begin{aligned} H(w) &= \sum_{k=0}^{2N} b_k e^{-2\pi i k w} = e^{-2\pi i N w} \sum_k b_k e^{-2\pi i (k-N)w} = \\ &= e^{-2\pi i N w} (b_N + \sum_{k=0}^{N-1} b_k (e^{-2\pi i (k-N)w} + e^{2\pi i (k-N)w})) = e^{-2\pi i N w} H_1(w), \end{aligned}$$

где $H_1(w)$ – вещественная функция. В этой ситуации имеет место линейная зависимость фазового сдвига от длины и от частоты. Сделаем замену, положив $g(t) = f(t + N)$, тогда преобразование Фурье $G(w) = e^{2\pi i N w} F(w)$, а после фильтрации $H(w)G(w) = |H_1(w)|G(w)$, то есть в этом случае сдвига не будет. Из свойства инвариантности линейного фильтра следует, что нужный эффект достигается сдвигом на N позиций результат фильтрации сигнала f . Рассмотрим пример.

```
import numpy as np
from scipy.signal import firwin, lfilter
from matplotlib import pyplot as plt
Fs = 1024
N = 7
Fs = 1024
Arg = np.arange(Fs) * 2.* np.pi / Fs
Arg1 = Arg * 50
Arg2 = Arg * 100
Sn1 = np.sin(Arg1)
Sn2 = np.sin(Arg2)
B = firwin(2*N+1, 70, fs = Fs)
FSn1 = lfilter(B, 1., Sn1)
FSn2 = lfilter(B, 1., Sn2)
fig, ax = plt.subplots()
ax.plot(Sn1[:30], 'b', label = '50Hz')
ax.plot(Sn2[:30], 'g', label = '100Hz')
```

```

ax.plot(FSn1[:30], 'r', label='Filterd 50Hz')
ax.plot(FSn2[:30], 'k', label='Filtered 100 Hz')
ax.legend()

```

Результат выполнения скрипта представлен на Рис.2.6.

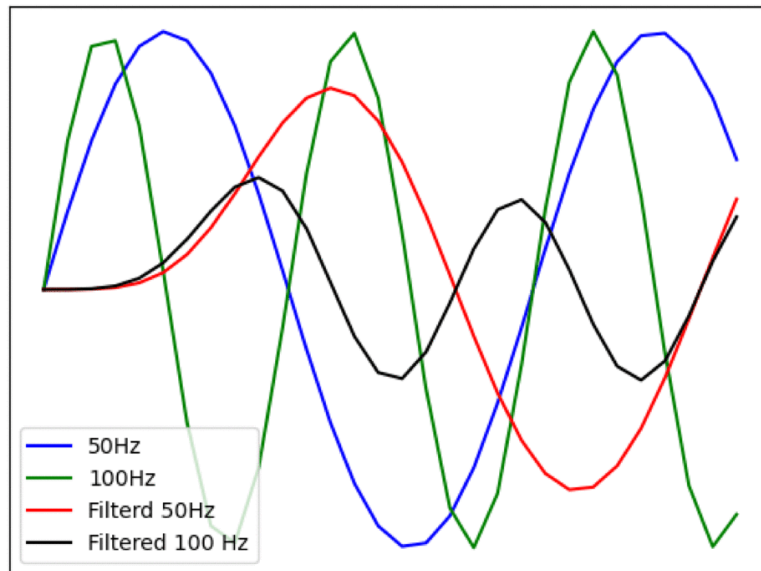


Рис. 2.6: Фазовый сдвиг после фильтрации сигналов с разными частотами

К сигналам с частотами 50 Hz и 100 Hz применили фильтр низких частот с частотой отсечения 70 Hz, поэтому в результате фильтрации амплитуда второго сигнала уменьшилась. Фазовые сдвиги (положения максимумов до и после фильтрации) в результате фильтрации разные. Применим технику, представленную выше.

```

N = 7
Fs = 1024
Arg = np.arange(Fs) * 2. * np.pi / Fs
Arg1 = Arg * 50
Arg2 = Arg * 100
Sn1 = np.sin(Arg1)
Sn2 = np.sin(Arg2)
B = firwin(2*N+1, 70, fs = Fs)
FSn1 = lfilter(B, 1., Sn1)
FSn2 = lfilter(B, 1., Sn2)
fig, ax = plt.subplots()
ax.plot(Sn1[:30], 'b', label='50Hz')

```

```

ax.plot(Sn2[:30], 'g', label = '100Hz')
ax.plot(FSn1[N:30 + N], 'r', label='Filterd 50Hz')
ax.plot(FSn2[N:30 + N], 'k', label = 'Filtered 100 Hz')
ax.legend()

```

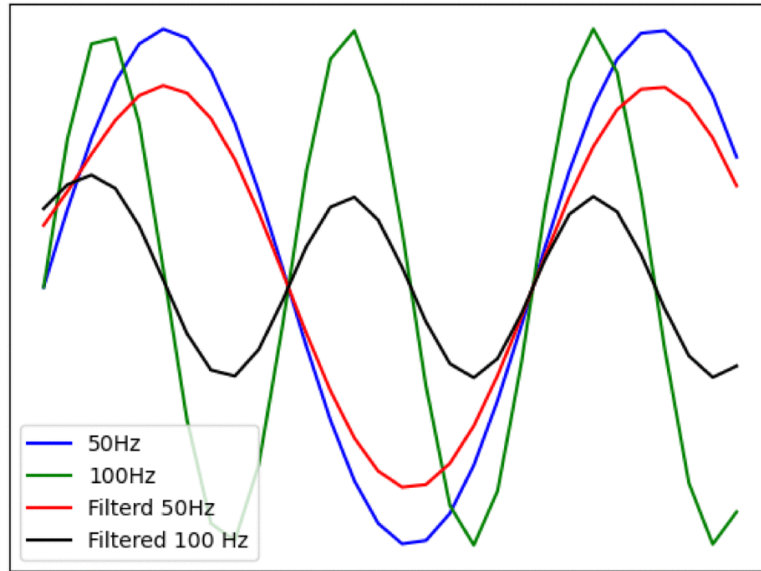


Рис. 2.7: Ликвидация фазового сдвига

Как следует из графиков на Рис. 2.7, фазовый сдвиг в результате фильтрации ликвидирован.

2.3.2 Двусторонний фильтр. Обратный проход фильтра

Как было замечено ранее, применение ИР фильтра приводит к фазовому сдвигу отфильтрованного сигнала по отношению к оригиналу, причем величина сдвига зависит достаточно сложно от частоты. В некоторой степени эту проблему можно решить, применив AllPass фильтр, теория которого будет изложена ниже. Другое решение основано на обратном проходе фильтра. Известны два решения этой проблемы.

1. Последовательное решение. Имеются последовательность x и фильтр с передаточной функцией $H(w)$. Последовательность y получена в результате фильтрации, поэтому $Y(w) = H(w)X(w)$. Положим $z[n] = y[-n]$, поэтому $Z(w) = Y(-w) = \bar{Y}(w)$. Пусть u получена в результате фильтрации z тем же фильтром. Имеем $U(w) = H(w)Z(w) = H(w)\bar{Y}(w)$. Теперь

$$U(w) = H(w)\bar{H}(w)\bar{X}(w) = |H(w)|^2 X(-w)$$

Это означает, что для четной последовательности, для которой $x[n] = x[-n]$ и, следовательно, $X(w) = X(-w)$ фаза отфильтрованного сигнала совпадает с фазой оригинала, для нечетной последовательности фаза меняется на 180 градусов. Для произвольного сигнала этот метод приводит к искажениям формы сигнала.

2. Параллельное решение. Применим фильтр к последовательности x и получим последовательность y . По определению $Y(w) = H(w)X(w)$. Запишем последовательность y в обратном порядке и получаем последовательность y_1 , для которой $Y_1(w) = \bar{H}(w)X(-w)$. Складываем $y[n]$ с y_1 . В результате получаем $Y(w) + Y_1(-w) = (H(w) + \bar{H}(w))X(w)$, если последовательность $x[n]$ четная, то есть преобразование с вещественной передаточной функцией.

Рассмотрим пример.

```
from scipy import signal as SGN
Fs = 512
N = Fs
Fr1 = 8
Fr2 = 25
Arg = np.arange(N)*2*np.pi/Fs
Cn = np.cos(Arg*Fr1)+np.cos(Arg*Fr2)
B,A = SGN.butter(3,20,btype='low',fs=Fs)
FCn = SGN.lfilter(B,A,Cn)
IFCn = FCn[::-1]
FIFSn = SGN.lfilter(B,A,IFCn)
fig,ax=plt.subplots()
ax.plot(Cn[:100], 'k', label='Orig')
ax.plot(FIFSn[:100], 'r', label="Filt")
ax.legend()
ax.set_xticks([])
ax.set_yticks([])
```

Исходная последовательность четная, но она есть сумма двух последовательностей, из которых первая пропускается фильтром, а вторая подавляется. График представлен на Рис.2.8. Условие четности или нечетности является существенным условием для отсутствия фазового сдвига. Например, заменив второе слагаемое на синус, получим фазовый сдвиг в отфильтрованном сигнале (см. Рис. 2.9

2.3.3 All-pass фильтр

Предыдущий подход работает только в случае четного или нечетного сигнала. Такой сигнал можно получить искусственно, продолжив исходный сигнал четным или нечетным образом. В случае длительного

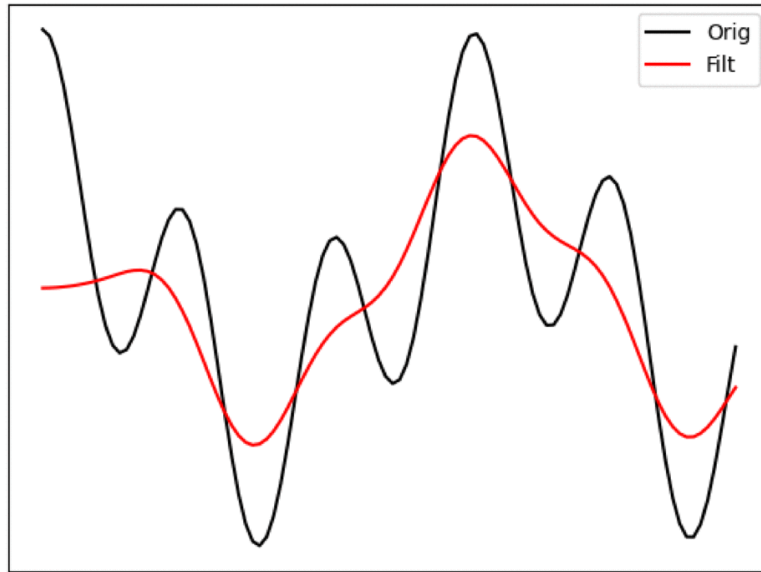


Рис. 2.8: Результат фильтрации четного сигнала

сигнала это приводит к увеличению времени обработки. Кроме того, требуется время на создание зеркального образа сигнала. С другой стороны, IIR фильтр имеет очевидные качественные преимущества перед FIR фильтром. Частично проблему фазового сдвига после применения IIR фильтра можно решить с помощью All-pass фильтра. Фильтр задается формулой

$$y[n] = ay[n-1] + ax[n] - x[n-1], \quad |a| < 1$$

Это устойчивый фильтр, а его передаточная функция имеет вид

$$H(w) = \frac{a - e^{-2\pi i w}}{1 - ae^{-2\pi i w}}$$

```
a = 0.8
B = [a,-1]
A = [1,-a]
w,h = sgn.freqz(B,A)
plt.plot(w,np.log(h/abs(h)).imag,'k')
```

Модуль этой передаточной функции равен 1, это означает, что фильтрация не вносит изменений в амплитуду спектра сигнала. Аргумент передаточной функции меняется вместе с частотой. Подбрав подходящим образом a , можем частично компенсировать фазовый сдвиг в окрестности некоторой точки, сделав его близким к линейному. Пример зави-

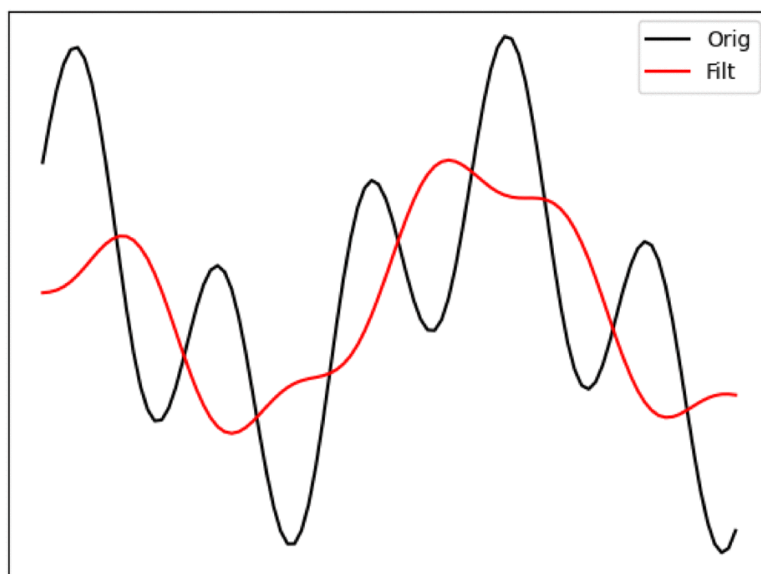


Рис. 2.9: Результат фильтрации сигнала при отсутствии четности

симости аргумента передаточной функции от частоты представлен на Рис.2.10

2.3.4 Фазовая и групповая задержка

Для характеристики ИР используют понятия фазовая и групповая задержка. Пусть имеется фильтр с передаточной функцией $H(w)$. Имеем представление в форме $H(w) = |H(w)|e^{i\theta(w)}$. С функцией $\theta(w)$ связаны две функции

$$t_p(w) = \frac{\theta(w)}{w}, \quad t_g(w) = \frac{d\theta(w)}{dw}$$

Первая из функций называется фазовой задержкой, а вторая — групповой задержкой. Отметим, что каждая из этих функций зависит от w . Если функция $\theta(w)$ является линейной, то обе функции будут постоянными. Это имеет место для симметричного фильтра. Отклонение от констант можно рассматривать как степень близости данного фильтра к FIR фильтру.

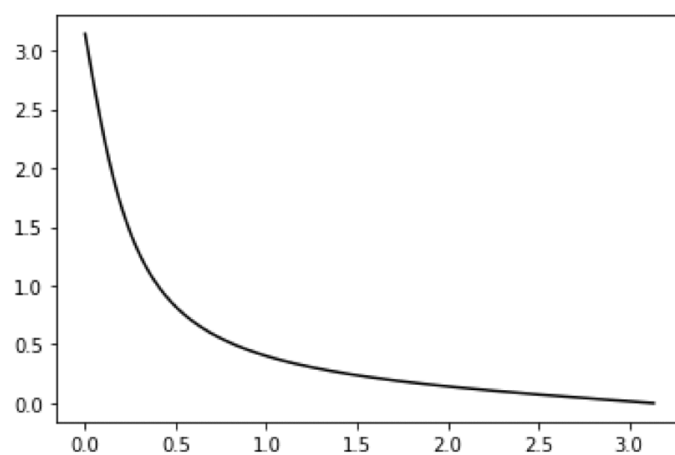


Рис. 2.10: Зависимость аргумента передаточной функции all-pass фильтра от частоты. Значение $a=0.8$

Глава 3

Примеры обработки цифровых сигналов

3.1 Работа со звуковым файлом

В этой главе рассматриваются различные эксперименты со звуковым файлом.

3.1.1 Загрузка файла

Загрузка файла в программу с диска осуществляется командой

```
from scipy.io.wavfile import read
[Fr,Data] = read('name.wav')
Если загружался стерео файл,
то надо разделить каналы
Chan1 = Data[:,0]
Chan2 = Data[:,1]
```

Здесь Fr – частота выборки, а Data имеет тип `numpy.int16`. Есть возможность просмотреть файл. В случае моно сигнала это делается командой

```
from matplotlib import pyplot as plt
plt.plot(Data)
```

Каждый эксперимент должен заканчиваться сравнением звучания оригинального файла и файла, полученного в результате эксперимента. Если `numpy`-массив содержит отсчеты звукового файла, то этот массив можно выгрузить на внешний носитель в виде wav-файла с помощью функции `scipy.io.wavfile.write`. После этого файл можно прослушать с помощью стандартного плеера. Эта процедура требует времени, поэтому лучше воспользоваться пакетом `playsound`. С его помощью можно прослушать файл, вызвав команду из программы. Пользователям Linux

можно предложить простой скрипт (небольшая модификация кода из [6]), который решает эту задачу.

```
def play_wave(Arr,Fr):
    import subprocess
    import tempfile
    from scipy.io.wavfile import write
    fl = tempfile.NamedTemporaryFile()
    filename = fl.name
    write(filename,Fr,Arr)
    cmd = 'aplay %s' % filename
    popen = subprocess.Popen(cmd, shell=True)
    popen.communicate()
    fl.close()
```

Здесь *Fr* — частота стробирования сигнала, а *Arr* — массив типа `numpy.int16`, содержащий отсчеты. Функция записывает содержимое *Arr* на диск во временный файл. Затем вызывается консольная утилита 'aplay', которая воспроизводит содержимое файла. После этого временный файл автоматически удаляется. В дальнейшем, если присутствует вызов программы *play_wave*, подразумевается вызов данной функции или любого эквивалента.

3.2 Сдвиг спектра сигнала

Спектр сигнала является важнейшей характеристикой сигнала. В случае звукового сигнала сдвиг спектра влево или вправо повышает или понижает высоту звука. Вообще говоря, подобный эффект можно реализовать сжимая или растягивая сигнал во времени, что можно сделать, изменив частоту выборки сигнала. subsectionПростейший способ изменения тона речевого сигнала Проще всего изменить тон можно растянув или сжав файл. Проигрывая этот файл с прежней частотой стробирования, получим эффекты понижения или повышения тона. Изменение длины файла осуществляет следующая функция.

```
import numpy as np
from numpy import interp
def changeFragm(Fragm,Coe):
    Ln = len(Fragm)
    Arg0 = np.arange(Ln)
    Ln1 = int(Ln*Coe)
    Arg1 = np.linspace(0,Ln-1,Ln1)
    return interp(Arg1,Arg0,Fragm)
```

Параметр $Coe > 1$ влечет понижение тона и наоборот. Нужные значения трансформированного фрагмента вычисляются с помощью линейной интерполяции. Это простое решение, но в результате применения функции *changeFragm* меняются время проигрывания и темп речи.

Гораздо интереснее выглядит задача повышения или понижения тона, сохраняя длину файла и частоту выборки, и, следовательно, время звучания.

3.2.1 Сдвиг с помощью модуляции и гребенки фильтров

Пусть задана последовательность x , преобразование Фурье которой равно $X(w)$. Напомним, что модуляцией сигнала $x[n]$ называется замена его на $x[n]f[n]$, где $f[n]$ некоторая заданная функция. Положим $y[n] = x[n] \exp(2\pi j w_0 n)$. Фактически это означает модуляцию исходного сигнала с помощью экспоненциальной последовательности специального вида. Найдем преобразование Фурье полученной последовательности.

$$Y(w) = \sum y[n] e^{-2\pi j w n} = \sum x[n] e^{-2\pi j (w - w_0) n} = X(w - w_0).$$

Это простейший способ сдвига спектра. Его недостаток заключается в том, что в результате получается комплексный сигнал. На первый взгляд, можно перейти к вещественному сигналу, взяв вещественную часть от $y[n]$, и избавиться от этой проблемы. Однако вещественная часть

$$z[n] = (y[n] + \bar{y}[n])/2 = x[n](\exp(-2\pi j w_0 n) + \exp(2\pi j w_0 n))/2,$$

поэтому $Z(w) = (X(w - w_0) + X(w + w_0))/2$, и вещественная часть будет содержать два сигнала со спектрами сдвинутыми в разные стороны. Тот же эффект достигается использованием преобразования сигнала $z'[n] = x[n] \cos(2\pi w_0 n)$. В этом случае для получения нужного результата надо подавить одну из составляющих сигнала. Если значение w_0 мало это трудно сделать, поскольку носители функций $X(w + w_0)$, $X(w - w_0)$ будут пересекаться. Выход заключается в представлении исходной последовательности $x[n]$ в виде

$$x[n] = \sum_k x^{(k)}[n],$$

где каждая составляющая $x^{(k)}[n]$ имеет очень узкий спектр. Для этого исходную последовательность надо пропустить через гребенку полосовых фильтров, к каждой полосе применить указанную выше процедуру модуляции и фильтрации, а потом сложить результаты. При создании гребенки фильтров часто приходится ограничиваться FIR фильтрами, поскольку не всегда удастся создать IIR фильтр с узкой полосой пропускания.

Для реализации этой идеи следует определиться с шириной полосы. Пусть это частоты из интервала $[F_0, F_1]$. Предполагается сдвинуть все частоты вверх на величину $Shift$. Для того, чтобы не пересекались интервалы частот сдвинутые вверх и вниз, должно быть выполнено неравенство $F_1 - Shift < F_0 + Shift$, или $F_1 - F_0 < 2 \cdot Shift$.

Рассмотрим пример. Пусть в исходный файл записан мужской голос. Если хотим сдвинуть спектр вверх на величину 150Hz, то каждая полоса должна иметь ширину не более 300Hz. Следует иметь ввиду, что данные значения весьма условны, поскольку мы имеем дело не с идеальными фильтрами, поэтому ширину полосы надо уменьшить, например, до 250 Hz. Наша цель достичь наблюдаемого эффекта при прослушивании модифицированного файла. Сначала надо ограничить спектр входного сигнала некоторой величиной $Bound$. Если мы имеем дело с речевым файлом, это значение можно выбрать из интервала 2000Hz - 4000Hz. При построении гребенки фильтров использована функция `firwin`, для которой важен параметр длина фильтра `FiltLen`.

```
import numpy as np
from scipy.io.wavfile import read,write
from scipy import signal as sgn

[Fr,Dat] = read('name_of_file')
Bound = 2500
Shift = 150
FiltLen = 13

def firstCut(Dat,Limit):
    #Restrict spectrum
    B = sgn.firwin(3,Limit,fs = Fr)
    Out = sgn.lfilter(B,1.,Dat)
    return Out
```

Создаем набор полос и выделяем части сигнала в этих полосах.

```
def createBandFrs(Beg,Step,End):
    Out = []
    B = Beg
    E = Beg + Step
    while E <=End:
        Band = (B,E)
        Out.append(Band)
        B += Step
        E = B + Step
    return Out
```

```

def getBand(Dat,Step,Band,FltLen):
    B = sgn.firwin(FltLen,Band,pass_zero=False,fs=Fr)
    Out = sgn.lfilter(B,1.,Dat)
    Ln = len(Dat)
    Arg = np.arange(Ln)*np.pi*2.*Step/Fr
    Cs = np.cos(Arg)
    ShiftArr = Out*Cs
    B = sgn.firwin(FltLen,(Band[0]+Band[1])*0.5,pass_zero = False,
                  fs = Fr)
    Signal = sgn.lfilter(B,1.,ShiftArr)
    return Signal

```

Главная программа

```

# main program
Rest = firstCut(Dat,Bound)
Step = 250
Beg = 10
End = Bound
Bands = createBandFrs(Beg,Step,End)
Out = np.zeros(len(Rest))
for BandCurr in Bands:
    Signal = getBand(Rest,Step,BandCurr,FltLen)
    Signal /= max(Signal)
    Out += Signal
Out /= max(Out)
play_wave(np.int16(2000*Out),Fr)

```

Возможны различные эксперименты с представленным скриптом. Для ограничения спектра обычно выбирают ИР фильтр, можно менять начальное значение частоты при построении гребенки, параметры сдвига, использовать ИР фильтр в качестве фильтра высоких частот. Наибольший интерес представляет набор весов, с которыми складывают результаты сдвига отдельных полос. Еще больший интерес представляет использование разных широт полос при построении гребенки, а также вариация частоты модуляции для разных полос. Та же схема применима для сдвига спектра вниз. Возможны и другие эксперименты. Можно сравнить получаемый результат с аналогичной опцией, присутствующей в программе audacity. Заметим, что описанный метод сдвига спектра не является единственно возможным.

3.2.2 Аналитический сигнал. Преобразование Гильберта

Существует метод, который теоретически может существенно подавить одну из составляющих, получившихся в результате сдвига. Предпола-

жим нам удалось построить фильтр с функцией отклика h_n , для которого передаточная функция $H(w)$ – преобразование Фурье от последовательности h_n имеет вид

$$H(w) = \begin{cases} -j & w > 0 \\ j & w < 0 \end{cases},$$

поэтому

$$h_n = j \int_{-1/2}^0 e^{2\pi j w n} dw - j \int_0^{1/2} e^{2\pi j w n} dw = \frac{1 - \cos(\pi n)}{\pi n} \quad (3.1)$$

Из (3.1) следует, что $h_{2k} = 0, \forall k$. Пусть имеется сигнал $x[n]$ и $X(w)$ – его спектр. Положим

$$y[n] = \sum_k x[n-k] h_k,$$

поэтому спектр отфильтрованного сигнала имеет вид

$$Y(w) = \begin{cases} -jX(w) & w > 0 \\ jX(w) & w < 0 \end{cases},$$

Если

$$c[n] = x[n] + jy[n], \quad (3.2)$$

то

$$C(w) = \begin{cases} 2X(w) & w > 0 \\ 0 & w < 0 \end{cases}$$

В силу указанной особенности спектра, сигнал $c[n]$ называют аналитическим. Пусть $z[n] = c[n] \exp(2\pi j w_0 n)$. Имеем преобразование Фурье от $z[n]$,

$$Z(w) = C(w - w_0).$$

Найдем $x_s[n] = \text{Re}(z[n])$. По определению $x_s[n] = (z[n] + \bar{z}[n])/2$. Но последовательности $\bar{z}[n]$ отвечает преобразование Фурье

$$\sum \bar{z}[n] \exp(-2\pi j n w) = \overline{\sum z[n] \exp(2\pi j n w)} = \bar{Z}(-w),$$

поэтому

$$X_s(w) = (C(w - w_0) + \bar{C}(-w - w_0))/2$$

Поскольку $C(w - w_0) = 0, w - w_0 < 0$, для положительных w спектр $X_s(w)$ последовательности $x_s[n]$ равен $2X(w - w_0)$, то есть получается из спектра $x[n]$ сдвигом на величину w_0 .

При практическом применении этого метода надо учитывать, что бесконечное число значений функции отклика h_n отлично от 0, поэтому приходится ограничиваться конечным числом слагаемых:

$$y[n] = \sum_{k=-N}^N h_k x[n-k]$$

Таким образом, дело сводится к обычному FIR фильтру. Этот фильтр получил название фильтра Гильберта. На Рис. 3.1 представлена передаточная функция фильтра для $N=12$. Покажем, каким образом она создана. Коэффициенты фильтра порождаются функцией hN

```
def hN(n):
    if (n>>1)<<1 == n:
        return 0
    else:
        Arg = np.pi*n
        return (1.-np.cos(Arg))/(Arg)
```

Как и для обычных FIR фильтров сглаживание коэффициентов специальным окном улучшает форму передаточной функции фильтра.

```
B = [hN(I) for I in range(-N,N+1)]
ArrB = np.float_(B)
ArrB *= sgn.hann(len(ArrB))
```

Для построения передаточной функции теперь надо найти ДПФ от построенного массива. Однако здесь мы сталкиваемся с небольшой проблемой. ДПФ определяется формулой

$$F(p) = \sum_{k=-N}^N h_k e^{-2\pi j(k+N)p/2N+1}.$$

Чтобы получить преобразование в виде

$$H(p) = \sum_{k=-N}^N h_k e^{-2\pi j(k)p/2N+1},$$

нужно воспользоваться формулой $H(p) = F(p)e^{2\pi jNp/2N+1}$.

```
Arg = np.arange(2*N+1)
Exp = np.exp(2*np.pi*1j*N*Arg/(2*N+1))
FB = np.fft.fft(ArrB)
plt.plot((1j*FB*Exp).real)
```

В идеале должен получиться график, составленный из прямых, образующих прямоугольник. На самом деле, это не так, поэтому следует иметь ввиду, что из-за замены фильтра $H(w)$ конечным фильтром в результате применения данной техники не получается идеальный сдвиг спектра. Появляется и составляющая со спектром $X(w + w_0)$, но она по мощности значительно уступает составляющей $X(w - w_0)$. Значение N в фильтре каждый раз приходится подбирать экспериментально.

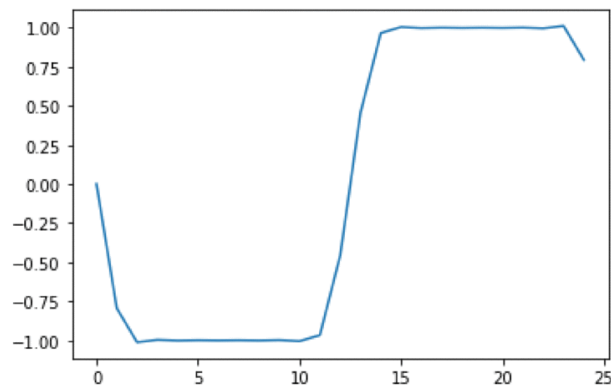


Рис. 3.1: Модуль передаточной функции фильтра Гильберта $N = 12$

3.2.3 Пример работы с аналитическим сигналом

Рассмотрим случай, когда исходный сигнал X имеет простейший спектр с носителем в двух точках.

```
L = 2048
Arg = np.arange(L)*2.*np.pi /L
Freq = 130
X = np.sin(Freq*Arg)
FX = np.fft.fft(X)
N = 7
B = [hN(I) for I in range(-N,N+1)]
ArrB = np.float_(B)
ArrB *= sgn.hann(len(ArrB))
Y = sgn.lfilter(ArrB,1.,X)
Y1 = Y[N:]
LnY_1 = len(Y1)
C = X[:LnY_1] + 1j*Y1
plt.figure()
plt.plot(abs(np.fft.fft(C)))
Shift = 60
Exp = np.exp (Arg* Shift *1j)
Z = C * Exp[:LnY_1]
ZR = Z.real
FZR = np.fft.fft(ZR)
plt.figure()
plt.plot(abs(FZR[:200]))
```

Результаты работы скрипта показаны на Рис.3.2. Здесь необходимо сделать два пояснения.

1. Значение N было уменьшено, поскольку в при $N = 12$ отрицательная часть спектра на графике не видна.
2. В результате сдвига подавленный отрицательный спектр также сдвинулся на ту же самую величину, что и основной (с учетом периодичности функции спектра с периодом L).
3. Замена Y на $Y1$. Это связано с особенностью реализации процедуры фильтрации в пакете Scipy, о которой было сказано в предыдущей главе, когда в начало сигнала помещаются нули. Пусть FIR фильтр имеет длину $2N + 1$. В процессе фильтрации интервал этой длины движется вдоль файла. Созданный файл должен иметь прежнюю длину, для этого в начало исходного файла добавляют $2N$ нулей. Согласно теории, элемент $x[n]$ складывается с элементом $y[n]$, вычисленным с помощью N элементов, стоящих как слева от $x[n]$, так и справа от него. Именно такие элементы $x[n]$ и $y[n]$ должны складываться.

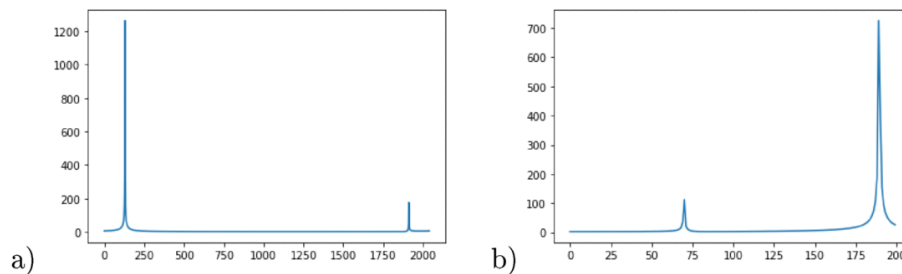


Рис. 3.2: Спектр преобразованного сигнала а) до и б) после сдвига

Взяв вместо функции `sin` любой файл с записью голоса, можно услышать сделанные изменения. Варьируя знак и величину `Shift`, получаем различные эффекты. Задачу можно усложнить, создать гребенку фильтров и экспериментировать с каждой полосой.

Функция `hilbert` в пакете `scipy`

Функция `hilbert` также решает задачу создания аналитического сигнала, но там реализован другой подход.

1. По сигналу x длины N находится ДПФ X .
2. В функции X обнуляется вторая половина коэффициентов и получается функция Y .
3. Подсчитывается обратное ДПФ от Y и получают аналитический сигнал s .

Если нужно сдвинуть спектр исходного сигнала, к полученному сигналу с применяют те же преобразования, что и выше. Преимущество данного подхода заключается в том, что получается 'чистый' аналитический сигнал без отрицательных компонентов в спектре. Недостаток метода проявляется при работе с большими входными файлами. Их приходится разбивать на участки приемлемые для нахождения ДПФ, после чего стыковать полученные файлы.

Вот пример работы скрипта, сдвигающего спектр в файле NameOfFile, на основе функции hilbert.

```
[Fr,Data] = read(NameOfFile)
Ln = len(Data)
Arg = np.arange(Ln)*2*np.pi/Fr
Exp = np.exp (-Arg* 160 *1j)
c = sgn.hilbert(Data)
Z = c* Exp
ZR = Z.real
ZR /= max(ZR)
play_wave(np.int16(20000*ZR),Fr)
```

3.3 Изменение частоты стробирования сигнала

Пусть имеется сигнал *Signal* с записью голоса, созданный с частотой выборки F_0 . Требуется заменить его сигналом *NewSignal* с частотой стробирования F_1 . Если проигрывать *NewSignal* с помощью функции *play_wave*, установив $Fr = F_1$, то можно услышать прежнее звучание (в разумных пределах для новой частоты.) Проще всего это сделать, используя линейную интерполяцию, как это было показано в предыдущем пункте.

```
import numpy as np
from matplotlib import pyplot as plt
Ln = 200
F_0 = 1
np.random.seed(23)
Signal = np.random.rand(Ln)
F_1 = 1.8
NewLen = int(F_0/F_1*Ln)
Arg = np.arange(Ln)
NewArg = np.linspace(0,Ln -1,NewLen)
NewSignal = np.interp(NewArg,Arg,Signal)
plt.plot(Arg,Signal)
plt.figure()
plt.plot(NewArg,NewSignal)
```

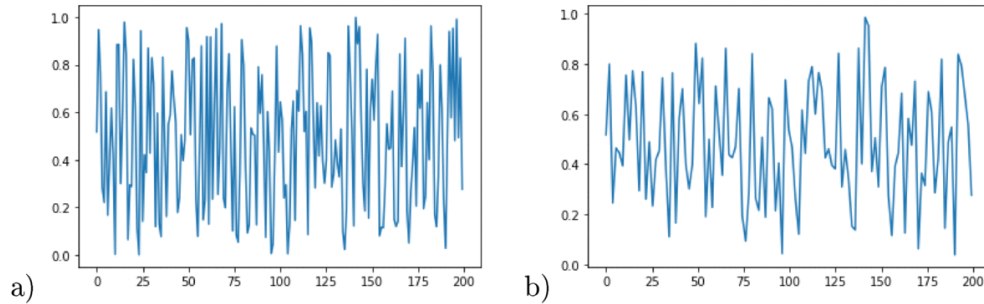


Рис. 3.3: Результат изменения частоты стробирования с помощью линейной интерполяции. а) Оригинальный сигнал, б) Сигнал после уменьшения частоты

Результат представлен на Рис.3.3. Это решение поставленной задачи, однако, при таком подходе трудно сказать, что произошло со спектром сигнала. В тех случаях, когда такая информация необходима, используют другой прием.

3.3.1 Децимация и обратная децимация сигнала

Пусть имеется дискретный сигнал $x[n]$. Речь идет о двух операциях: децимация, когда образуется сигнал $x_M[n] = x[Mn]$ (downsampling), и обратная децимация x^M , когда между отсчетами $x[n]$ и $x[n+1]$ вставляют $M - 1$ нулевой сигнал (upsampling). Чтобы понять, что происходит со спектром сигнала, вернемся к исходному непрерывному сигналу $x(t)$. При сохранении времени проигрывания в первом случае понижается частота выборки, а во втором случае частота выборки повышается, но промежуточные значения заменяются не истинными значениями сигнала, а нулями.

Децимация

Мы хотим установить связь спектра x_M со спектром исходного сигнала x . Имея значения x_M , мы можем восстановить x только при выполнении условий теоремы Котельникова-Шеннона. Ранее была установлена связь между преобразованием Фурье от непрерывного сигнала и от дискретного сигнала

$$X(w) = \sum x[nT]e^{-2\pi jnwT} = \frac{1}{T} \sum_k X_c(w - \frac{n}{T})$$

где $X(w)$ – преобразование Фурье от дискретного сигнала, полученного выборкой с частотой $1/T$ из непрерывного сигнала, а $X_c(w)$ – преобразование Фурье от непрерывной функции.

После децимации и замены $T \rightarrow MT$

$$Z(w) = \sum x[MnT]e^{-2\pi jwnMT} = \frac{1}{MT} \sum_n X_c(w - \frac{n}{MT}) \quad (3.3)$$

Если выбор MT согласован с условиями теоремы Котельникова-Шеннона, то

$$X(w) = \frac{1}{T} X_c(w), |w| < \frac{1}{2T}, \quad Z(w) = \frac{1}{MT} X_c(w), |w| < \frac{1}{2MT}$$

Если теперь перейдем к преобразованию Фурье от последовательности x_M , то получим

$$X_M(w) = \sum x[nMT]e^{-2\pi jwnT} = \sum x[Mn]e^{-2\pi jw/MTn} = Z(w/M).$$

Окончательно имеет

$$X_M(w) = \frac{1}{M} X_1(w/M) \quad (3.4)$$

Таким образом происходит растягивание спектра исходного сигнала. Еще раз отметим, что формула (3.4) справедлива, когда выполнены условия теоремы Котельникова-Шеннона. На практике в тех случаях, когда нет уверенности в том, что эти условия выполнены, перед операцией downsampling фильтруют исходную последовательность фильтром низких частот с тем, чтобы уменьшить ширину спектра исходного сигнала и исключить эффект подмены частот. При прослушивании полученного сигнала с прежней частотой стробирования время звучания уменьшится, и из низкого голоса получаем высокий голос. Прослушивая этот файл с частотой стробирования уменьшенной в M раз, получаем прежнее звучание.

Обратная децимация

Выясним, что происходит со спектром сигнала при операции upsampling. Пусть дана последовательность $x[n]$, а $y[n]$ получена в результате вставки $M - 1$ нулей между последовательными отсчетами исходной последовательности, $y[Mn] = x[n], y[k] = 0, k \neq Mn \quad (\forall n)$. Теперь

$$Y(w) = \sum_k y[k]e^{-2\pi jkw} = \sum_n x[n]e^{-2\pi jMnw} = X(Mw)$$

Это означает, что в результате операции upsampling спектр последовательности сужается в M раз, но остается периодической функцией. Наименьший период преобразования Фурье уменьшится в M раз, поэтому, чтобы оставить только первый компонент, следует пропустить сигнал через фильтр низких частот.

Комбинируя операции `downsampling` и `upsampling`, теоретически можно изменить частоту стробирования, умножив эту частоту на любой рациональный множитель. Достоинством указанного подхода является его простота. В то же время, надо иметь ввиду ограничения, накладываемые на операцию `downsampling`, для того, чтобы сохранить свойства исходного сигнала. Указанные операции приводят к сжатию или растяжению спектра сигнала. .

3.3.2 Пример реализации `upsampling` и `downsampling`

Рассмотрим пример применения описанных операций при работе с речевым файлом.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.io.wavfile import read
from scipy import signal as sgn
[Fr,Dat] = read ('Name_of_file')
N = 1024
```

Выберем фрагмент этого файла и нарисуем его спектр. Чтобы сделать результат более наглядным, пропускаем файл через полосовой фильтр. Формально теперь спектр сигнала лежит в известных пределах.(Рис.3.4)

```
B = sgn.firwin(17,(500,800),pass_zero = False, fs= Fr)
NewDat = sgn.lfilter(B,1.,Dat)
Chunk = NewDat[2000:2000 +N]
FChunk = np.fft.fft(Chunk)
Freq = np.arange(N)/N *Fr
plt.plot(Freq[:128],abs(FChunk[:128]))
```

Обратим внимание на массив *Freq*, содержащий частоты.

Операция `upsampling`. Рисуем спектр нужного фрагмента, а также спектр всего преобразованного файла (Рис. 3.5)

```
NewLen = 4*len(NewDat)
Up = np.zeros(NewLen)
Up[:4] = NewDat
Chunk = Up[8000:8000 +N]
FChunk = np.fft.fft(Chunk)
Freq = np.arange(N)/N*Fr
plt.plot(Freq[:128],abs(FChunk[:128]))
plt.figure()
#All spect
Freq = np.arange(NewLen)/NewLen*Fr
```

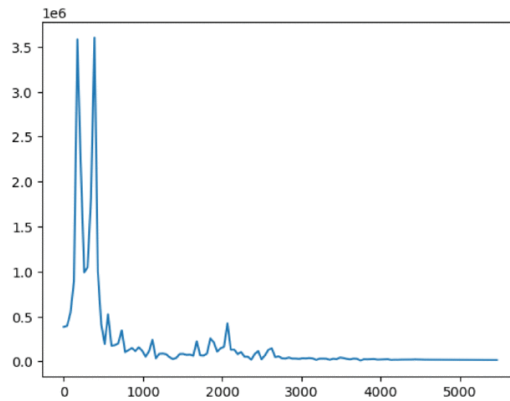



Рис. 3.4: Спектр исходного файла

```
plt.plot(Freq[:,4],abs(np.fft.fft(Up)[:,4]))
#Select the region
B = sgn.firwin(17,5000,pass_zero=True,fs = Fr)
NewDat1 = sgn.lfilter(B,1.,Up)
```

Как следует из Рис.3.5, спектр преобразованного файла получается периодическим, и нам нужно выделить только один участок, самый первый. Это делается с помощью фильтра низких частот.

Теперь попытаемся с помощью децимации вернуться к исходному файлу.

```
DwnDat = NewDat1[:,4]
DwnChunk = DwnDat[2000:2000 +N]
FDwnChunk = np.fft.fft(DwnChunk)
Freq = np.arange(N)*Fr/N
plt.plot(Freq[:,128],abs(FDwnChunk[:,128]))
```

Получившийся спектр на Рис.3.6 похож на спектр оригинала.

При построении спектра следует учитывать следующий важный момент. В результате операций *upsampling* и *downsampling* получаются новые последовательности, но предполагается, что они получены с одной и той же частотой выборки Fr .

3.3.3 B-Spline и сжатие звуковых файлов

Интерполяция в известных пределах позволяет сжимать звуковые файлы без потери качества. Предположим, что имеется запись звукового файла с высоким качеством, то есть с малым шагом T_0 . Такой файл будет занимать много места, поэтому запишем его с новым шагом $T_1 > T_0$, но так, чтобы сохранились условия теоремы Котельникова-Шеннона с

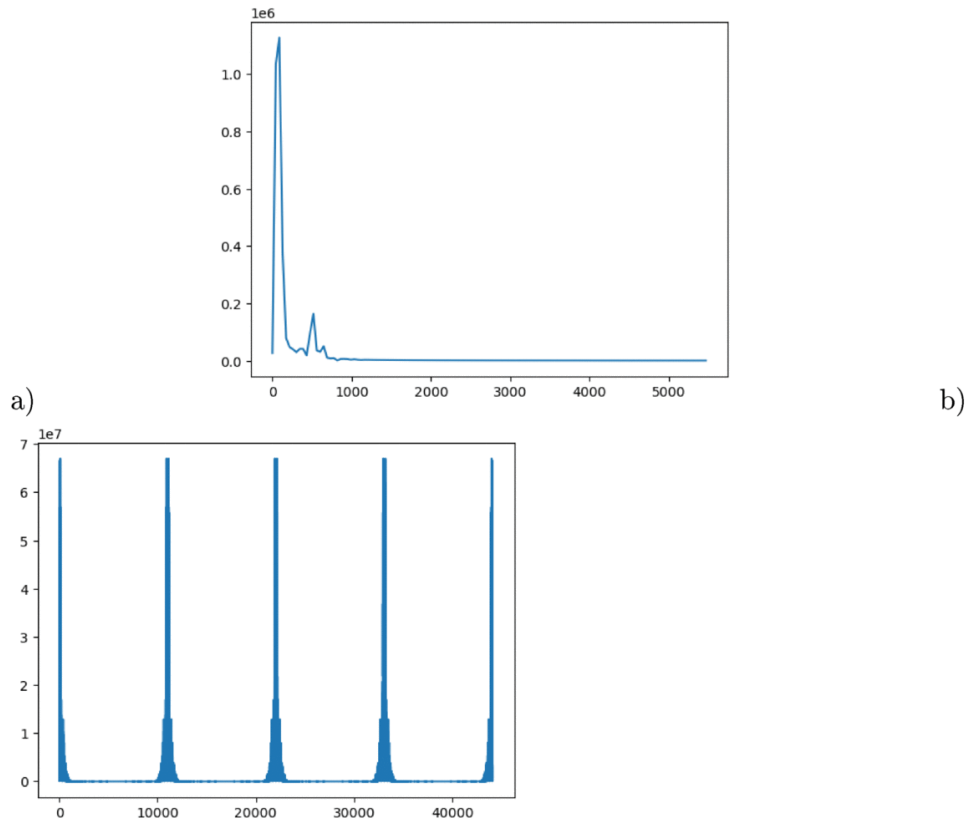


Рис. 3.5: Спектры: а) выделенная полоса спектра. б) спектр фрагмента после обратной децимации

новым шагом. В таком виде файл помещается на носитель. При воспроизведении надо вернуться к прежнему шагу. Это можно сделать с помощью интерполяции на основе B-Spline'ов (базисных сплайнов).

Остановимся только на основах теории этих сплайнов. Вводятся базисные функции, состоящие из фрагментов многочленов заданной степени. Эти фрагменты стыкуются между собой таким образом, чтобы базисная функция имела гладкую производную порядка $n - 1$, где n это степень многочленов из базиса. Другое фундаментальное понятие – набор точек (узлов) по которым строятся многочлены. Мы остановимся на простейшем случае, когда узлы отстоят друг от друга на расстоянии 1. Весь сплайн является линейной комбинацией базисных функций одной степени, причем все базисные функции получаются из одной функции путем сдвига аргумента. Теперь перейдем к определениям. Второй индекс в обозначении базисной функции означает степень многочлена.

$$1. \text{ Функция } B_{0,0}(t) = \begin{cases} 1, & -1/2 \leq t \leq 1/2 \\ 0, & \text{othercase.} \end{cases}$$

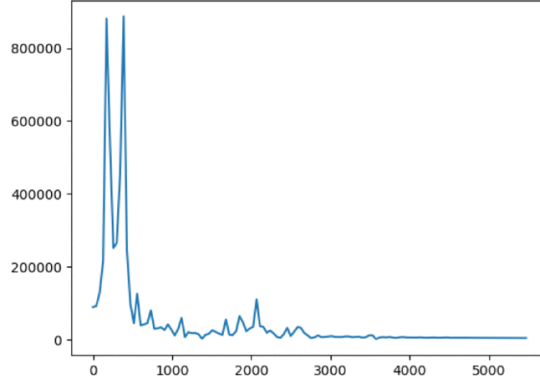


Рис. 3.6: Спектр восстановленного участка

2. $B_{k,n}(t) = B_{0,n}(t - k)$.

3. $B_{0,n} = B_{0,0} * B_{0,0} * \dots * B_{0,0}$ свертка n раз.

Найдем явный вид некоторых из базисных функций.

$$B_{0,1}(t) = \int_{-1/2}^{1/2} B_{0,0}(t-u)B_{0,0}(u)du = \int_{-1/2}^{1/2} B_{0,0}(t-u)du = \int_{t-1/2}^{t+1/2} B_{0,0}(v)dv$$

Очевидно, что $B_{0,1}(t) = 0$, $|t| > 1$. Если $t > 0$, то

$$B_{0,1}(t) = \int_{t-1/2}^{1/2} dv = (1-t)$$

Если $t < 0$, то

$$B_{0,1}(t) = \int_{-1/2}^{t+1/2} dv = t+1.$$

Итак, это непрерывная ломанная линия, имеющая разрывы производной в целых точках.

По определению,

$$B_{0,2}(t) = B_{0,1} * B_{0,0}(t) = \int_{-1}^1 B_{0,1}(u)B_{0,0}(t-u)du,$$

поэтому

$$B_{0,2}(t) = \int_{-1}^0 (1+u)B_{0,0}(t-u)du + \int_0^1 (1-u)B_{0,0}(t-u)du$$

Носитель этой функции находится в интервале $[-\frac{3}{2}, \frac{3}{2}]$. Он разбивается на 3 равные части, на каждой из которых определено свое аналитическое выражение. При $[-3/2 \leq t < -1/2]$ это первое слагаемое, при $-1/2 \leq t \leq 1/2]$ это оба слагаемых, а при $1/2 \leq t \leq 3/2$ – это второе слагаемое. Окончательно,

$$B_{0,2}(t) = \begin{cases} (t + 3/2)^2/2 & -3/2 \leq t \leq -1/2, \\ (-2t^2 + 3/2)/2 & -1/2 \leq t \leq 1/2, \\ (t - 3/2)^2/2 & 1/2 \leq t \leq 3/2. \end{cases}$$

Вид этой кривой представлен на Рис.3.7

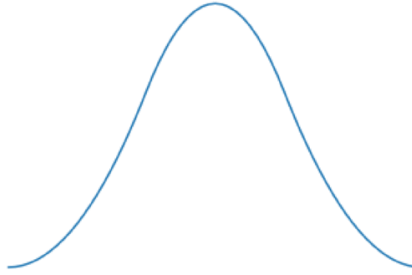


Рис. 3.7: Вид сплайна второго порядка

Интерполяция осуществляется следующим образом. Имеется множество точек $x_0, x_1, \dots, x_M$ и значения $y_0, y_1, \dots, y_M$. Выбирается степень сплайна n . Находятся коэффициенты $c_0, c_1, \dots, c_M$ такие, что функция

$$B(t) = \sum_{k=0}^M c_k B_{k,n}(t)$$

обладала свойством $B(x_i) = y_i, i = 0, \dots, M$. При этом должно быть выполнено условие, что все x_i лежат внутри носителя $B(t)$. Это достигается подходящим выбором узлов. В пакете `scipy` имеется реализация алгоритма, учитывающая все тонкости постановки.

Рассмотрим пример.

```
import numpy as np
import scipy

def getSNR(U,V):
    Err = U-V
    return 10*np.log10(np.var(U)/np.var(Err))
```

Таблица 3.1: Оценки SNR при интерполяции с помощью B-Spline в зависимости от степени сплайна

k	1	2	3	5	7	9
SNR	19.8	37.3	46.9	71.2	90.3	92.0

```

from scipy.interpolate import make_interp_spline
X = np.arange(0,10,.1)
F = lambda x:np.sin(2*np.pi*x) +np.cos(4*np.pi*x)
Y = F(X)
Bpl = make_interp_spline(X,Y,k=9)
Arg = np.arange(.5,9.5,.01)
U = F(Arg)
V = Bpl (Arg)
print(getSNR(U,V))

```

Спектр функции F находится в интервале $[-2,2]$ длины 4. Для выполнения условий теоремы Котельникова-Шеннона для шага T выборки должно быть выполнено неравенство $T < 1/8$. При вычислении SNR мы немного снизили область определения функции по сравнению с начальными установками. Результаты помещены в Таблицу 3.1. Значение SNR в районе 90 дБ считается приемлемой.

3.4 Сжатое описание цифрового файла. Линейное предсказание

Цифровой файл при стандартном хранении требует большого объема памяти. В некоторых задачах, например, при поиске нужного файла из заданного множества требуется использовать какие-то параметры ограниченного размера, с помощью вычисления которых удастся упростить поиск. Считается известной некоторая априорная информация о природе этих файлов. В частности, предполагается информационная избыточность присутствующая в этих файлах. Например по интервалу файла можно определить следующий за интервалом символ с какой-то заданной ошибкой. Это справедливо как для случайных стационарных файлов, так и для текстовых, речевых и музыкальных файлов. Простейший способ описания этой зависимости является линейное предсказание, когда по линейной комбинации с заданными коэффициентами элементов интервала можно вычислить следующий элемент файла с заданной ошибкой. При этом коэффициенты остаются постоянными для значительных фрагментов файла. Следует отметить, что в случае случайных файлов та же методика носит название авторегрессии. В дальнейшем используется терминология и техника случайных процессов, хотя обла-

стью применения являются файлы, не являющиеся случайными. При этом среднее значение случайной величины заменяется средним значением параметра на некотором интервале. Оправдание таким действиям заключается в том, что полученные значения для детерминированных файлов приводят к результатам, имеющим практическое значение.

3.4.1 Линейное предсказание

Пусть имеется вещественный случайный процесс $y[n]$ с дискретным временем, обладающий свойствами: $E(y[n]) = 0$, $E(y[n]y[n+p])$ зависит только от p . (стационарный в широком смысле). Задача заключается в предсказании следующего значения $y[n]$ на основе k предыдущих. Требуется выбрать коэффициенты $a_1, a_2, \dots, a_k$ таким образом, чтобы

$$E(y[n] - a_1y[n-1] - a_2y[n-2] - \dots - a_ky[n-k])^2 \rightarrow \min$$

Для отыскания коэффициентов найдем частные производные по параметрам и приравняем их нулю.

$$\frac{\partial}{\partial a_j} E(y[n] - a_1y[n-1] - \dots - a_ky[n-k])^2 =$$

$$2E((y[n] - a_1y[n-1] - \dots - a_ky[n-k])y[n-j]) = 0 \quad (3.5)$$

Положим $R(p) = E(y[n]y[n+p])$. Заметим, что $R(p) = R(-p)$. В этих обозначениях равенства (3.5) принимают вид системы из k уравнений:

$$R(1) = a_1R(0) + a_2R(1) + \dots + a_kR(k-1)$$

$$R(2) = a_1R(1) + a_2R(0) + \dots + a_kR(k-2)$$

последнее уравнение имеет вид

$$R(k) = a_1R(k-1) + a_2R(k-2) + \dots + a_kR(0)$$

Полученную систему запишем в матричной форме. Обозначим через

$$A_k = \begin{pmatrix} R(0) & R(1) & \dots & R(k-1) \\ R(1) & R(0) & \dots & R(k-2) \\ \dots & \dots & \dots & \dots \\ R(k-1) & R(k-2) & \dots & R(0) \end{pmatrix},$$

$$f_k = \begin{pmatrix} R(1) \\ R(2) \\ \dots \\ R(k) \end{pmatrix}, \quad \alpha_k = \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_k \end{pmatrix}.$$

Теперь система (3.5) имеет вид

$$A_k \alpha_k = f_k. \quad (3.6)$$

Решение можно записать в форме $\alpha_k = A_k^{-1} f_k$. На практике используется лишь значения сигнала на некоторых интервалах, где сигнал можно считать стационарным. Значения $R(p)$ автокорреляционной функции подсчитываются с помощью техники быстрого преобразования для ДПФ. Очевидная связь систем уравнений (3.6) для разных k подсказывает существование зависимости между решениями этих систем. Это действительно так. Существует быстрый способ решения систем, не требующий обращения матрицы. Этот алгоритм носит название алгоритма Durbin'a. Он находит применение в случае, когда реализуется на процессоре малой мощности.

3.5 Примеры использования линейного предсказания

3.5.1 Применение линейного предсказания для сжатия сигнала

Проблема уменьшения размеров звукового файла остается актуальной. Это всегда компромисс между качеством звучания и коэффициентом сжатия. Пусть задан сигнал $y[n]$, для которого найдены коэффициенты линейного предсказания. Рассмотрим сигнал

$$z[n] = y[n] - \sum_{p=1}^k a_p y[n-p] \quad (3.7)$$

Этот сигнал является результатом применения FIR фильтра к исходному сигналу. Далее, если известен сигнал $z[n]$, то с помощью коэффициентов линейного предсказания исходный сигнал может быть восстановлен

$$y[n] = \sum_{p=1}^K a_p y[n-p] + z[n] \quad (3.8)$$

то есть сигнал восстанавливается с помощью ИИР фильтра. По построению, сигнал $z[n]$ должен иметь меньшую среднюю амплитуду по сравнению с исходным. Идея сжатия заключается в хранении начальных значений $y[0], y[1], \dots, y[k-1]$, коэффициентов предсказания и значений $z[n]$. Для хранения последних понадобится меньшее число битов. На практике используется модифицированная процедура. Составляется специальная кодовая книга, в которой содержатся возможные наборы коэффициентов линейного предсказания. По найденным коэффициентам предсказания ищется набор в кодовой книге наиболее близкий к найденному, и этот набор кодируется одним числом. Невязка $z[n]$ после этого подсчитывается через коэффициенты выбранного кода из кодовой книги. Вся

область значений невязки разбивается на интервалы. В качестве погрешности хранится не само значение погрешности, а код интервала. Коэффициенты предсказания вычисляются лишь для некоторого промежутка времени. Они модифицируются при переходе от одного промежутка к другому. Эта идея (в модифицированной форме) была реализована в коммерческом продукте фирмы Microsoft. Она обеспечивала небольшой коэффициент сжатия без существенной потери качества. Существует открытый стандарт (Vorbis), в котором реализована эта идея. Недостатком является то, что и кодер и декодер должны содержать одну и ту же кодовую книгу, а сама книга в стандарт не входит. От этого недостатка свободен стандарт МРЗ, получивший настоящее время наибольшее распространение. Имеется и открытый вариант OGG формата для сжатия звуковых файлов, основанный на той же идее. В основе этих форматов лежит применение косинус преобразования. Подробнее об этом будет сказано ниже.

3.5.2 Применение линейного предсказания для получения сглаженного спектра сигнала

При вычислении спектра с помощью ДПФ получаем график модулей коэффициентов, который выглядит достаточно случайным образом. Существует простой способ усреднения. Используя линейное предсказание, можно получить более гладкую картину. Пусть $z[n]$ – вещественные случайные величины, обладающие свойством

$$E(z[n]) = 0, \quad E(z[n]^2) = s^2, n = 0, \dots, M-1, \quad E(z[n]z[m]) = 0, m \neq n. \quad (3.9)$$

Преобразование Фурье от этой последовательности

$$Z(w) = \sum z[n] \exp(-2\pi jnw).$$

Подсчитаем $E(|Z(w)|^2)$. Имеем

$$|Z(w)|^2 = \sum_n z[n] e^{-2\pi jnw} \overline{\sum_m z[m] e^{-2\pi jmw}} = \sum_n z[n]^2 + \sum_{m \neq n} z[m] z[n] e^{2\pi j(m-n)w}$$

Подсчитывая среднее значение и учитывая (3.9), получаем $E(|Z(w)|^2) = Ms^2$. Переходя в (3.7) к преобразованиям Фурье, получим

$$Y(w)(1 - \sum_p a_p \exp(-2\pi jpw)) = Z(w),$$

а посчитывая среднее значение, найдем

$$E(|Y(w)|^2) = \frac{Ms^2}{(|1 - \sum_p a_p \exp(-2\pi jpw)|)^2} \quad (3.10)$$

Таким образом получается сглаженная за счет усреднения форма спектра.

3.5.3 Обнаружение схожих музыкальных произведений

Формула (3.10) может служить основой для отыскания нужного файла из коллекции музыкальных файлов без прослушивания. В частности, можно обнаружить свои файлы, подвергшиеся небольшим изменениям злоумышленником с целью затруднить обнаружение нарушения авторских прав. Рассмотрим эту задачу более подробно. Пусть имеются два файла: оригинальный и другой файл. Требуется сравнить их без прослушивания. Предполагается, что оба файла воспринимаются на слух примерно одинаково, поэтому спектры сигналов должны различаться мало.

1. В каждом из файлов выбираем отрезки, отвечающие одному и тому же моменту времени, $Chunk_1, Chunk_2$.
2. Выбираем числа K_1, K_2 коэффициентов линейного предсказания соответствующих длин для каждого из отрезков. Если частоты стробирования в обоих файлах совпадают, то числа выбираем равными.
3. Пользуясь формулами (3.10), строим кривые сглаженного спектра и сравниваем их.

Ниже приведен скрипт, реализующий этот метод. Первая проблема, с которой мы сталкиваемся при реализации данной идеи, это способ нахождения коэффициентов линейного предсказания. Непосредственное применение формулы (3.6) для отрезков небольшой длины приводит к неустойчивым значениям. Анализируя доказательство формулы, заметим, что матрица системы уравнений имеет указанный вид только в случае бесконечной последовательности. Результаты сильно зависят от выбора K , а матрица системы может быть близкой к вырожденной. С целью избежать этой трудности коэффициенты линейного предсказания вычисляют с помощью линейной регрессии.

Линейная регрессия

Линейная регрессия решает следующую задачу. Имеется L векторов $v_0, v_1, \dots, v_{L-1}$ длины K и числа $b_0, b_1, \dots, b_{L-1}$. Требуется найти вектор α длины K и число C такие, что выполнено условие

$$\sum_{p=0}^{L-1} (b_p - (v_p, \alpha) - C)^2 \rightarrow \min.$$

Здесь (α, v) обозначает скалярное произведение. Задача отыскания коэффициентов линейного предсказания для конечных входных последовательностей сводится к задаче линейной регрессии следующим образом. Пусть отрезок файла, для которого решается задача, имеет вид:

$u_0, u_1, \dots, u_{M-1}$. Положим $v_0 = (u_0, u_1, \dots, u_{K-1})$, $v_1 = (u_1, u_2, \dots, u_K)$ и т.д., $b_0 = u_K, b_1 = u_{K+1}, \dots$ и т.д. Потребуем, чтобы $C = 0$. Искомые коэффициенты линейного предсказания это элементы вектора α записанные в обратном порядке. Модуль, реализующий линейную регрессию имеется в пакете `scikit-learn`.

Эксперимент

```
import numpy as np
from scipy.io.wavfile import read
from scipy import signal as sgn
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
[Fr, Dat] = read(name_of_file) # исходный файл для экспериментов
```

Исходный файл подвергся небольшой модификации – применили к нему фильтр низких частот.

```
B = sgn.firwin(11, 3000, pass_zero = True, fs= Fr)
DatNew = sgn.lfilter(B, 1, Dat)
```

Это почти не повлияло на восприятие содержимого, однако, отсчеты, стоящие на одинаковых местах, различаются. Выберем число коэффициентов K , начало и конец фрагментов, для которых будет производиться сравнение спектров.

```
K = K0
Sec = SEC
BegTime = int(Fr*Sec)
Duration = .5
EndTime = BegTime + int(Duration*Fr)
Chunk1 = np.float_(Dat[BegTime:EndTime])
Chunk2 = DatNew[BegTime:EndTime]
```

Теперь понадобится программа, вычисляющая коэффициенты предсказания.

```
def getPred(Chunk, K):
    Chunk -= np.mean(Chunk)
    Mod = LinearRegression(fit_intercept = False)
    X = []
    Y = []
    Beg = 0
    End = K
    while End <= len(Chunk) - 1:
        X.append(Chunk[Beg:End])
```

```

        Y.append(Chunk[End])
        Beg += 1
        End += 1
    AX = np.float_(X)
    AY = np.float_(Y)
    Mod.fit(AX,AY)
    return Mod

```

Мы хотим положить $C = 0$ в формуле регрессии. Для этого при создании объекта Mod использовали значение False для параметра *fit_intercept*. Для корректной работы программы необходимо, чтобы среднее значение элементов отрезка было нулевым. Мы сравниваем спектры, пользуясь

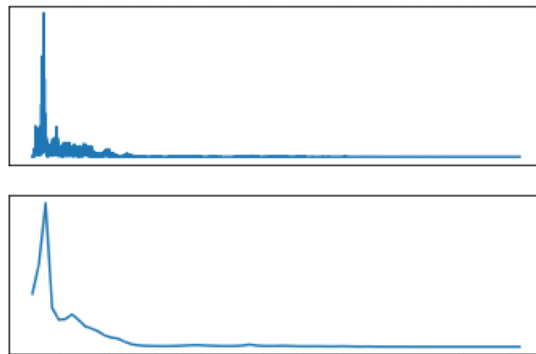


Рис. 3.8: Спектр сигнала и его сглаженная версия

формулой (3.10). С этой целью создадим набор точек, в которых будет вычисляться спектр.

```

NumPoints = PNTS
Points = np.linspace(0,.5,NumPoints)
def getGraphPoint(CoefPred,Point):
    K = len(CoefPred)
    Arg = -2.*np.pi*1j
    Numb = np.arange(1,K+1)
    Exs = np.exp(-Arg*Point*Numb)
    Prod = sum(CoefPred*Exs)
    return 1/(abs(1-Prod))

def getAllPnts(CoefPred,Points):
    Arr = np.zeros(len(Points))
    for I,Pt in enumerate(Points):
        Arr[I] = getGraphPoint(CoefPred,Pt)
    return Arr

```

Обычный спектр и его сглаженная версия представлены на Рис. 3.8.

Мы хотим сравнивать два спектра. Нас интересует форма кривой, а не абсолютные значения в точках. Это делается с помощью функции

```
def compArr(U,V):  
    S = sum(U*V)  
    return S/(np.linalg.norm(U)*np.linalg.norm(V)),
```

вычисляющей косинус угла между двумя векторами. Чем более схожи векторы, тем ближе абсолютное значение косинуса к 1.

Выбранный для исследования интервал файла может оказаться достаточно большим. В этом случае имеет смысл разбить его на более мелкие перекрывающиеся интервалы, найти спектр каждого из них, а потом усреднить результат.

```
def collectPredArrays(BigChunk,K,Points,Size,Step):  
    Ln = len(Points)  
    Final = np.zeros(Ln)  
    Count = 0  
    Beg = 0  
    End = Size  
  
    while End <= len(BigChunk):  
        Small = BigChunk[Beg:End]  
        Small -= np.mean(Small)  
        Mod = getPred(Small,K)  
        Pred = Mod.coef_[::-1]  
        Final += Arr  
        Count += 1  
        Beg += Step  
        End = Beg +Size  
    return Final/Count
```

- Обратите внимание, каким образом получают доступ к значениям коэффициентов регрессии. Обычно полагают $Step = Size/2$, но это не обязательно.

Сравнение спектров производится следующим образом.

```
Size =SIZE  
Step = STEP  
Arr1 = collectPredArrays(Chunk1,K,Points,Size,Step)  
Arr2 = collectPredArrays(Chunk2,K,Points,Size,Step)  
print(compArr(Arr1,Arr2))
```

Проводился эксперимент с $K0 = 15$, $NumPoints = 30$, $Size = 512$, $Step = Size/2$. Получено значение $compArr(Arr1,Arr2)$ равное 0.995 Для сравнения выберем отрезок той же длины из произвольного файла. Пусть это

Chunk3. Подсчитаем $Arr3 = collectPredArrays(Chunk3, K, Points, Size, Step)$ и сравним его с оригиналом $print(compArr(Arr1, Arr3))$. Здесь значение $compArr(Arr1, Arr3)$ равно 0.652. Это лишь пример применимости метода. Для других параметров результат может быть другим.

Замечание

На практике часто для оценки спектра фрагмента файла используют функцию `scipy.signal.welch`. Она также осуществляет вычисление спектра путем осреднения спектров малых отрезков, однако, вместо линейного предсказания использует ДПФ.

3.6 Другие преобразования на основе ДПФ, используемые в задачах обработки цифровых файлов

ДПФ является самым популярным преобразованием для исследования свойств цифровых файлов. Наиболее важное свойство этого преобразования – в случае стационарного сигнала независимость модулей коэффициентов преобразования от начала отсчета. В то же время, имея вещественный сигнал на входе, мы получаем комплексные коэффициенты. От этого недостатка свободны некоторые другие преобразования, нашедшие применение в цифровой обработке сигналов.

3.6.1 Преобразование Хартли (Hartley)

Преобразование Хартли непрерывного сигнала задается формулой

$$H(w) = \int_{-\infty}^{\infty} f(t)cas(2\pi wt)dt, \quad cas(t) = \cos(t) + \sin(t).$$

Обратное преобразование вычисляется по той же формуле:

$$f(t) = \int_{-\infty}^{\infty} H(w)cas(2\pi wt)dw.$$

Рассмотрим дискретный аналог этого преобразования:

$$H_k = \sum_{n=0}^{N-1} f(n)cas(2\pi kn/N), \quad k = 0, \dots, N-1. \quad (3.11)$$

Напомним, что выражение $N|n$ означает, что n делится на N , а $N \nmid n$ означает, что n не делится на N . При изучении свойств ДПФ была доказана формула

$$\sum_{k=0}^{N-1} e^{2\pi jkn/N} = \begin{cases} 0 & N \nmid n \\ N & \text{otherwise} \end{cases} \quad (3.12)$$

По определению,

$$\begin{aligned} cas(a) &= (e^{ja} + e^{-ja})/2 + (e^{ja} - e^{-ja})/(2j) = \\ &= 1/2e^{ja}(1 - j) + 1/2e^{-ja}(1 + j). \end{aligned} \quad (3.13)$$

Подсчитаем

$$\sum_{k=0}^{N-1} cas(an)cas(am), \quad a = 2\pi k/N \quad (3.14)$$

Имеем

$$\begin{aligned} cas(an)cas(am) &= (1/2e^{jan}(1 - j) + 1/2e^{-jan}(1 + j)) \\ & \quad (1/2e^{jam}(1 - j) + 1/2e^{-jam}(1 + j)). \end{aligned}$$

Условие $N|n-m$ может быть выполнено только при $m = n$. Здесь следует рассмотреть несколько частных случаев.

1. Пусть $N \nmid m - n$ и $N \nmid m + n$. В этом случае согласно (3.12) выражение (3.14) будет нулем.
2. Пусть $N|m + n$ и $N \nmid m - n$. Каждое произведение $cas(an)cas(am)$ будет представлено четырьмя слагаемыми. При суммировании два обратятся в 0, а два оставшихся в сумме дадут $N((1 + j)^2 + (1 - j)^2)/4 = 0$
3. Пусть $m = n$. В этом случае возможно выполнение каждого из условий: $N|2n$ или $N \nmid 2n$, но результат равен $2N(1 - j)(1 + j)/4 = N$.

Рассмотрим матрицу Q размера $N \times N$, в которой элемент

$$Q[m, n] = \frac{1}{\sqrt{N}} cas(2\pi mn/N).$$

Свойство (3.14) обеспечивает ортогональность матрицы Q . С другой стороны, $Q = Q^T$. Это означает, что преобразование с матрицей Q и обратное к нему задается одной и той же матрицей. Обычно множитель с квадратным корнем удаляют из прямого преобразования, поэтому обратное преобразование к (3.11) задают в форме

$$f(n) = \frac{1}{N} \sum_{k=0}^{N-1} H_k cas(2\pi kn/N). \quad (3.15)$$

На практике для быстрых вычислений преобразования Хартли используют ДПФ. Согласно (3.13), $cas(a) = \text{real}((1 + j)e^{-ja})$, поэтому для вещественного сигнала f формула (3.11) принимает вид

$$\langle H_0, H_1, \dots, H_{N-1} \rangle = \text{real}((1 + j)FF),$$

где FF есть ДПФ от вектора $\langle f(0), f(1), \dots, f(N - 1) \rangle$. Вот простой пример, иллюстрирующий сказанное.

```

import numpy as np
A = np.arange(10)
B = np.fft.fft(A)
C = ((1+1j)*B).real
E = np.fft.fft(C)
G = ((1+1j)*E).real
print(G/10)

```

На печать будет выдан исходный массив A . Преобразование Хартли используют в тех случаях, когда требуется оценить частоты, превышающие половину длины области определения сигнала.

3.6.2 Дискретное косинус преобразование (DCT)

Значительно более широкое распространение получило дискретное косинус преобразование (DCT), которое реализует разложение сигнала по специальному базису. Чисто экспериментально было показано, что разложение сигнала, встречающегося в реальных задачах, по этому базису оказывается более эффективным, о чем будет сказано ниже. Следует заметить, что существует целое семейство DCT преобразований. Здесь представлено одно из них. Это преобразование лежит основе форматов сжатия звука MP3 и изображения JPEG.

Пусть $x(0), \dots, x(N-1)$ - вещественный вектор. Рассмотрим симметричный вектор $x(N-1), \dots, x(0), x(0), \dots, x(N-1)$. Это вектор размерности $2N$. Перепишем его в виде $z(0), \dots, z(2N-1)$ и применим к нему преобразование Фурье в нормированной форме. Введем обозначения

$$q = \frac{1}{\sqrt{2N}}, \quad W = e^{-2\pi i/2N}$$

тогда

$$\begin{aligned}
 y(n) &= q \sum_{k=0}^{2N-1} z(k) W^{kn} = q \sum_{k=0}^{N-1} x(N-1-k) W^{kn} + q \sum_{k=N}^{2N-1} x(k-N) W^{kn} = \\
 &= q \sum_{k=0}^{N-1} x(k) W^{-kn+n(N-1)} + q \sum_{k=0}^{N-1} x(k) W^{kn+nN} = q W^{nN} \sum_{k=0}^{N-1} x(k) (W^{-kn} e^{2\pi i n/2N} + W^{kn})
 \end{aligned}$$

Домножим обе части получившегося равенства на $W^{nN} e^{-\pi i n/2N}$. Учитывая, что $W^{2nN} = 1$, получим

$$\begin{aligned}
 d(n) &= y(n) W^{nN} e^{-\pi i n/2N} = q \sum_{k=0}^{N-1} x(k) (W^{-kn+n/2} + W^{kn-n/2}) = \\
 &= 2q \sum_{k=0}^{N-1} x(k) \cos(\pi(2k+1)n/2N)
 \end{aligned}$$

Обратим внимание на то, что коэффициенты $d(n)$ являются вещественными числами. Таким образом, получили вещественные коэффициенты преобразования. Поскольку они найдены с помощью ДПФ, существует обратное преобразование. Прямое и обратное преобразования заложены в пакет `scipy`.

```
import numpy
from numpy.linalg import norm
from scipy.fft import dct,idct
A = np.arange(3)
DA = dct(A,norm='ortho')
print(norm(A),norm(DA))
InA = idct(DA, norm='ortho')
print(InA)
```

Параметр `norm='ortho'` обеспечивает ортогональность преобразования. Это означает, что оператор `print(norm(A),norm(DA))` напечатает два одинаковых числа (с учетом точности вычисления). Оператор `print(InA)` выдаст исходный массив A .

3.6.3 Сравнение эффективности трех преобразований

Каждое из рассмотренных выше преобразований имеет свои особенности. Чтобы сравнивать их, нужно выбрать какой-либо критерий. Одна из целей разложения сигнала по базису состоит в аппроксимации этого сигнала по частичному разложению. Пусть имеется сигнал $X = \langle x_0, x_1, \dots, x_{N-1} \rangle$ и некоторый базис $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$. Разложим этот сигнал по базису

$$X = \sum_{k=0}^{N-1} y_k \alpha_k.$$

Оставим только первые P из коэффициентов разложения, а остальные коэффициенты заменим нулями. В результате получим аппроксимацию исходного сигнала в виде

$$\tilde{X} = \sum_{k=0}^{P-1} y_k \alpha_k.$$

Найдем разность (невязку) $X - \tilde{X}$. Поскольку мы имеем дело с ортогональными преобразованиями, для сравнения длин векторов выберем Евклидову норму. Для эксперимента использован фрагмент речевого файла.

```
import numpy
from numpy.linalg import norm
```



```

from scipy.fft import dct,fft,ifft,idct
from scipy.io.wavfile import read
[Fr,Dat] = read(name_of_file)
Ln = LN
Fragm = Dat[3*Fr:3*Fr+Ln]

```

Сначала убедимся, что каждое из преобразований сохраняет длину. При этом надо учесть нормирующие множители.

```

FFragm = fft(Fragm)
DFragm = dct(Fragm, norm='ortho')
HFrags = ((1+1j)*FFragm).real
print(norm(Fragm))
print(norm(FFragm/np.sqrt(Ln)))
print(norm(DFragm))
print(norm(HFrags/np.sqrt(Ln)))

```

Выберем число P . Следует учесть, что каждый коэффициент ДПФ является парой вещественных чисел, причем коэффициент с индексом 0 всегда вещественное число. В двух других преобразованиях все коэффициенты вещественные. Для корректного сравнения методов надо выбирать число P нечетным. Заметим, что при восстановлении сигнала с помощью `ifft` должно быть выполнено условие симметричности коэффициентов Фурье для вещественного сигнала.

```

P = PNUM
PF = P//2
#DFT
Coef = np.zeros(Ln).astype(np.complex_)
Coef[0] = FFrags[0]
Coef[1:PF] = FFrags[1:PF]
Coef[-PF:] = FFrags[-PF:]
Rest = ifft(Coef).real
print('DFT', np.linalg.norm(Fragm-Rest))

```

Для двух других преобразований выбираем первые P коэффициентов.

```

#Hartley
Coef = np.zeros(Ln)
Coef[:P] = HFrags[:P]
FCoef = fft(Coef)

Rest = ((1+1j)*FCoef).real
print('Hartley', np.linalg.norm(Fragm-Rest/Ln))
#DCT
Coef = np.zeros(Ln)

```

```

Coef[:,P]= DFrags[:,P]
Rest = idct(Coef,norm ='ortho')
print('DCT',np.linalg.norm(Frags-Rest))

```

Приведем результаты эксперимента с фрагментом речевого файла. Выбраны параметры $L_n = 512$, $PNUM = 27$. Длина фрагмента равна 223344, длина невязки: DFT 73476, Hartley 148396, DCT 67109. Даже на этом примере видна эффективность DCT в процедурах сжатия. Это является причиной использования DCT в процедурах сжатия звука и изображений.

Замечание В рассмотренном примере мы оценивали эффективность процедуры сжатия с помощью вычисления длины невязки между исходным файлом и сжатым. Возможен другой критерий, когда качество сжатия оценивается с помощью корреляции между двумя сигналами. В этом случае оценивается сохранение формы сигнала после исключения части коэффициентов. Соответствующий пример будет приведен ниже.

Следует отметить, что реализация процедур сжатия на основе DCT является нетривиальной задачей. В формате JPEG все изображение разбивается квадраты одного размера, каждый из них аппроксимируется с помощью DCT, а для найденных коэффициентов разложения использованы процедуры архивирования найденных коэффициентов. Кроме того, приходится сглаживать границы соседних квадратов. Более сложный алгоритм применен в формате MP3. При создании этого метода были использованы факты из физиологии восприятия звуков человеком. Известно, что основная информация, заложенная в речевой файл, находится в полосе в окрестности 2kHz. Исходный сигнал пропускается через гребенку полосовых фильтров, сигнал на выходе каждого фильтра разбивается на фрагменты, и каждый фрагмент сжимается с помощью DCT. Эти фрагменты могут иметь разные длины, но все они кратны некоторой величине, поэтому они согласованы по времени. Процесс декодирования очень прост: надо применить обратное DCT преобразование к каждому фрагменту и сложить получившиеся результаты. Вся тонкость заключается в том, что для каждой полосы для восстановления сигнала с помощью DCT используют не все коэффициенты, а только часть, таким образом, в каждой полосе используется аппроксимация оригинального сигнала, при этом для каждой полосы точность восстановления своя. Это и есть know how создателей алгоритма кодирования. Достоинство данного формата заключается в том, что процесс декодирования доступен, он не зависит от способа кодирования и легко реализуется аппаратно. Этим объясняется широкое распространение формата. На самом деле, формат MP3 является параметрическим, есть возможность настройки качества восстановленного сигнала. Однако применение этого формата для записи классической музыки считается неприемлемым.

3.7 Преобразования не связанные с ДПФ

Несмотря на то, что ДПФ и связанные с ним преобразования являются основными инструментами для исследования сигнала, существуют и другие преобразования, обладающие преимуществом в некоторых ситуациях.

Непрерывное wavelet преобразование

Wavelet-преобразование возникло в связи с исследованием сигналов, возникающих в результате землетрясений. Обычно эти сигналы состоят из короткой вспышки и последующих затухающих колебаний. Требовалось предложить компактный способ, описывающий форму сигнала. Преобразования на основе преобразования Фурье плохо подходят для этой цели, поскольку они предназначены для исследования процессов близких к периодическим. Первоначальное решение поставленной задачи выглядело следующим образом. Пусть функция $f(t)$ описывает исследуемый процесс. Допускается, что эта функция принимает комплексные значения. Выбирается некоторая функция $u(t; param)$, зависящая от параметров $param$ и имеющая единственный всплеск. Параметры подбираются таким образом, чтобы эта функция как можно лучше совпадала с функцией $f(t)$. В качестве меры совпадения выбирается корреляция (скалярное произведение) этих функций. Эта идея реализуется с помощью некоторой функции $s(x)$ – материнской функции простой формы, имеющей всплески в нуле. Эта функция модифицируется и превращается в функцию

$$u(t, a, b) = s\left(\frac{t - b}{a}\right).$$

Скалярное произведение вычисляется по формуле

$$F(a, b) = \int_{-\infty}^{\infty} f(t) \bar{u}(t, a, b) dt.$$

Поскольку

$$\int_{-\infty}^{\infty} \left| s\left(\frac{t - b}{a}\right) \right|^2 dt = a \int_{-\infty}^{\infty} |s(t)|^2 dt,$$

величина

$$F(a, b) / \sqrt{a} \tag{3.16}$$

может служить мерой близости функций $f(t)$ и функции $u(t, a, b)$. Найдя значения параметров a, b , для которых (3.16) достигает максимума, получаем аппроксимацию исходного сигнала. При этом параметр b определяет момент всплеска, а параметр a отвечает за форму сигнала. Пусть

найденны значения a_0, b_0 , для которых (3.16) достигла максимума. Можно составить разность $g(t) = f(t) - F(a_0, b_0)/\sqrt{a_0} \cdot u(t, a_0, b_0)$ и применить к ней ту же самую операцию отыскания наилучшей аппроксимации с помощью материнской функции. Повторяя это действие несколько раз, в результате получим приближение функции $f(t)$ с помощью линейной комбинации функций, полученных сдвигом и масштабированием из функции $s(t)$. Как правило, такое разложение не обладает свойством ортогональности. Мы не будем останавливаться более подробно на этом случае

3.7.1 Дискретное wavelet преобразование

С практической точки зрения, наибольший интерес представляет дискретный аналог wavelet преобразования. Если непрерывный вариант wavelet был известен давно, то дискретный случай был разработан сравнительно недавно, и он слабо связан с теорией непрерывных wavelet.

На Рис.3.9 изображена схема зеркального фильтра, состоящего из 2 блоков. На вход схемы подается сигнал с временем T между импульсами. Этот сигнал обрабатывается каждым из 2 фильтров, в результате чего появляются два сигнала, но теперь время между импульсами равно $2T$. По этим двум потокам можно восстановить исходный сигнал. Матема-

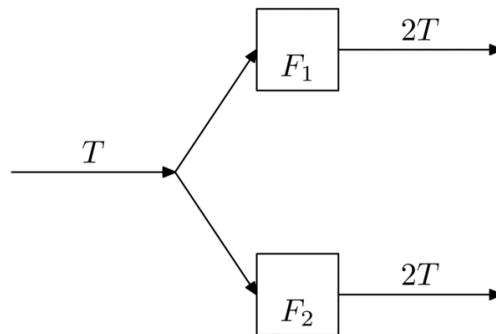


Рис. 3.9: Зеркальный фильтр

тическая модель этой схемы описывается следующим образом. Имеется

матрица

$$A = \begin{pmatrix} b_0 & b_1 & \cdots & b_{2K-1} & 0 & 0 \cdots & 0 & 0 & 0 \\ 0 & 0 & b_0 & b_1 & \cdots & b_{2K-1} & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & b_0 & b_1 & \cdots & b_{2K-1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_0 & p_1 & \cdots & p_{2K-1} & 0 & 0 \cdots & 0 & 0 & 0 \\ 0 & 0 & p_0 & p_1 & \cdots & p_{2K-1} & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & p_0 & p_1 & \cdots & p_{2K-1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

бесконечного порядка. Потребуем, чтобы эта матрица была ортогональной. Это условие сводится к выполнению следующих равенств

$$\sum b_i^2 = 1, \sum p_j^2 = 1, \sum b_i b_{i+2M} = 0, \sum p_i p_{i+2M} = 0,$$

для натуральных M и

$$\sum b_{i+2L} p_{i+2N} = 0 \quad (3.17)$$

для целых L, N , для которых (3.17) имеет смысл. Частный случай этого преобразования, когда $B = (1, 1)/\sqrt{2}$, $P = (1, -1)/\sqrt{2}$, под названием преобразования Хаара известен с начала прошлого века. Впервые существование чисел b_i, p_j , обладающих нужными свойствами при $K > 1$ доказала И.Добеши в конце прошлого века. Например, для $K = 2$ искомый набор имеет вид :

$$B = \frac{1}{4\sqrt{2}}(1+q, 3+q, 3-q, 1-q), \quad P = (B[3], -B[2], B[1], -B[0]), \quad q = \sqrt{3} \quad (3.18)$$

Матрица A естественным образом разбивается на два блока: строки с элементами b_i и строки с элементами p_j , $A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$. При вычислении произведения $A \cdot \langle x[0], x[1], \dots \rangle^T$ получаем два потока – результат произведения на первый блок A_1 это то, что соответствует верхней линии на Рис. 3.9, а произведение на второй блок A_2 – нижней линии. Поскольку матрица A ортогональна, обратная матрица получается транспонированием и имеет вид $A^T = (A_1^T, A_2^T)$. Восстановление сигнала сводится к умножению первого потока на A_1^T (верхняя линия на рисунке) и на A_2^T (нижняя линия) с последующим сложением.

Рассмотрим содержательный смысл представленного преобразования. Как уже отмечалось, преобразование разбивается на два блока. С точки зрения wavelet преобразования мы имеем две дискретные материнские функции: это строки $A_1[0]$ и $A_2[0]$. Операцию умножения сигнала на матрицу A можно рассматривать как движение каждой из строк вдоль сигнала, и в каждой позиции происходит вычисление скалярного

произведения отрезка сигнала и значений функции. Это аналог изменения параметра b в определении непрерывного преобразования. Легко проверить, что в (3.18) вектор B представляет значения медленно меняющейся функции, а вектор P представляет значения быстро меняющейся функции. Это означает, что в верхнем блоке на Рис.3.9 наибольшие коэффициенты получаются в моменты, когда на текущем интервале сигнал меняется медленно, а на нижнем – наоборот. Другими словами, из сигнала выделяются таким образом низкочастотная и высокочастотная части. Номер коэффициента в преобразовании указывает на позицию интервала. В реальной ситуации, когда сигнал является конечным, его приходится модифицировать. Есть несколько вариантов, как это можно сделать: в конец сигнала добавляются нули или последний элемент сигнала, делают сигнал симметричным, добавляя последние сигналы в обратном порядке, и другие методы. После этого все процедуры становятся обычными операциями линейной алгебры или фильтрации.

3.7.2 Реализация дискретного wavelet преобразования

Все общепринятые методы работы с wavelet преобразованием собраны в пакете PyWavelets. Он импортируется под именем pywt. Покажем, как сказанное выше реализовано в этом пакете. Имеется большое число различных wavelet преобразований. Пример, представленный формулой (3.18), включен в пакет как 'db2'. Выше преобразование сводилось к умножению на матрицу, однако, в пакете оно реализовано как пара фильтров. Коэффициенты этих фильтров это массивы B и P записанные в обратном порядке.

```
import numpy as np
import pywt
q = np.sqrt(3)
B = np.float_([1+q, 3+q, 3-q, 1-q])*.25/np.sqrt(2)
P = [B[3], -B[2], B[1], -B[0]]
W = pywt.Wavelet('db2')
print('Lo', W.dec_lo)
print('Hi', W.dec_hi)
print('B', B[::-1])
print('P', P[::-1])
```

Выполнив данный скрипт, можно убедиться в совпадении соответствующих массивов.

Как было указано ранее, после применения wavelet преобразования получаются два потока – результаты фильтрации фильтрами низких и высоких частот. По этим потокам можно восстановить исходный сигнал.

```
Data = np.arange(10)
```

```
[cL,cH] = pywt.dwt(Data,'db2')
IData = pywt.idwt(cL,cH,'db2')
print(IData)
```

3.7.3 Multiresolution

Наиболее популярной процедурой на основе вейвлет преобразования стала процедура под названием multiresolution. Процедура базируется на гипотезе, согласно которой после вейвлет преобразования речевого сигнала основная информация оказывается в низкочастотной части. Если исходный сигнал имел длину $2N$, то низкочастотная составляющая имеет длину N . Эту составляющую снова подвергают вейвлет преобразованию и т. д. При восстановлении сигнала высокочастотную составляющую заменяют нулями, а конечную низкочастотную часть считают краткой характеристикой речевого файла или его части. Представим реализацию этого подхода.

```
def getLowWavelet(Data,Wavelet,Level):
    Low = Data.copy()
    for I in range(Level):
        [Low,Hi] = pywt.dwt(Low,Wavelet)
    return Low

def restoreData(Low,Wavelet,Level):
    Data = Low
    for I in range(Level):
        Data = pywt.idwt(Data,np.zeros(len(Data)),Wavelet)
    return Data
```

Сравним эффективность указанной процедуры сжатия с аналогичной процедурой на основе DCT. Прямое сравнение сжатого сигнала с оригиналом не имеет смысла, поскольку сжатый сигнал имеет значительно меньшую мощность по сравнению с исходным. В этой связи в качестве критерия совпадения двух сигналов выберем совпадение их форм с помощью функции Corr. Чем ближе к 1 полученное значение, тем больше совпадение сигналов.

```
Norm = lambda F: np.linalg.norm(F)
Corr = lambda X,Y: sum(X*Y)/(Norm(X)*Norm(Y))
```

Для эксперимента выберем отрезок речевого файла на Рис.3.10. Эксперимент выглядит так.

```
Ln = 1024
Level = 4
Wavelet = 'db4'
```

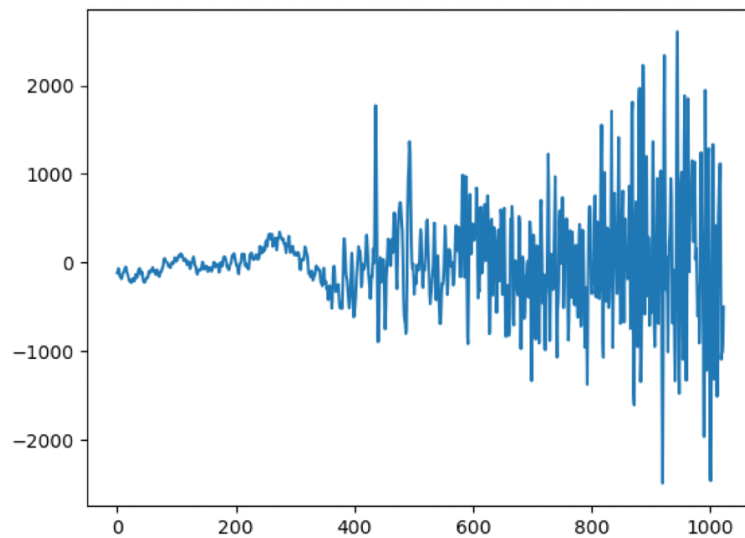


Рис. 3.10: Фрагмент речевого файла

```
[Fr,Data] = read(name_of_file)
Fragm = Data[Fr:Fr+Ln]
plt.plot(Fragm)
Compr = getLowWavelet(Fragm,Wavelet,Level)
PNUM =len(Compr)
Rest = restoreData(Compr,Wavelet,Level)
DFragm = dct(Fragm,norm='ortho')
DCoef = np.zeros(Ln)
DCoef[:PNUM] = DFrags[:PNUM]
DRest = idct(DCoef,norm='ortho')
print(Corr(Fragm,DRest))
print(Corr(Fragm,Rest[:Ln]))
```

В данном случае выданы значения 0.327 и 0.336, то есть wavelet сжатие показывает некоторое преимущество. В общем случае результат зависит как от выбора параметров Wavelet, Level, так и самого фрагмента файла, но каждый раз различие оказывается небольшим.

3.7.4 Сжатие файла и метод главных компонент

Особое место занимает подход, когда вместо фиксированного базиса применяется базис, построенный по самому сигналу. Этот метод основан на вероятностном подходе, и лучшие результаты достигаются для случайных сигналов, однако, он часто применяется в ситуации с детерминированным сигналом. Этот метод имеет различные модификации и назва-

ния, но в большинстве случаев он известен как РСА (principal component analysis).

Пусть имеется $m \times n$ вещественная матрица A . Ее можно рассматривать как набор строк $\alpha_1, \dots, \alpha_m$. каждая из которых есть вектор длины n . Требуется найти r независимых векторов $\beta_1, \dots, \beta_r$ длины n таким образом, чтобы

$$\sum_{k=0}^m |\alpha_k - \gamma_k|^2 \rightarrow \min. \quad (3.19)$$

Здесь γ_k есть проекция вектора α_k на подпространство с базисом $\beta_1, \dots, \beta_r$. Без ограничения общности можем предполагать, что векторы $\beta_1, \dots, \beta_r$ составляют ортонормированную базу. Задача (3.19) переписывается следующим образом

$$\sum_{k=0}^m \left| \alpha_k - \sum_{i=1}^r (\alpha_k, \beta_i) \beta_i \right|^2 \rightarrow \min.$$

Расписывая длину вектора через скалярное произведение и учитывая, что скалярное произведение двух векторов $\alpha = (a_1, \dots, a_n)$ и $\beta = (b_1, \dots, b_n)$, заданных строками, записывается в виде $\alpha\beta^T$ или $\beta\alpha^T$, получим, что дело сводится к отысканию ортонормированных векторов β_k , для которых

$$\sum_{i,k} (\alpha_k, \beta_i)^2 \rightarrow \max,$$

или

$$\sum_{i=1}^r \beta_i Q \beta_i^T \rightarrow \max, \quad (\beta_i, \beta_j) = \delta_{ij} \quad (3.20)$$

где

$$Q = \sum_{k=1}^m \alpha_k^T \alpha_k = A^T \cdot A.$$

По условию, матрица Q - неотрицательно определенная симметрическая матрица. Без доказательства сформулируем решение задачи (3.20) в общем случае: пусть $c_1, \dots, c_n$ собственные значения матрицы Q , причем $c_1 \geq c_2 \geq \dots \geq c_n$; подпространство, натянутое на векторы $\beta_1, \dots, \beta_r$ совпадает с подпространством, натянутым на r собственных векторов матрицы Q , отвечающих собственным значениям $c_1, \dots, c_r$. В частности, в качестве векторов $\beta_1, \dots, \beta_r$ можно взять ортонормированную систему из r собственных векторов, отвечающих r наибольшим собственным значениям.

Разобьём сигнал на отрезки длины n и составим из них матрицу A . Спроектируем строки A на подпространство, порожденное собственными векторами. В результате получим приближение сигнала с помощью

матрицы ранга r . Это приближение может рассматриваться как разложение исходного сигнала по базису, связанному с этим сигналом. . Он дает наилучшее приближение в смысле L_2 заданной матрицы в базисе, состоящем из r векторов.

Описанную процедуру иногда называют декорреляцией. Поясним смысл этого термина в данном контексте. Если исходные векторы $\alpha_1, \dots, \alpha_m$ являются значениями одного и того же случайного вектора с нулевыми средними значениями компонентов, то матрица Q/m становится выборочной матрицей ковариаций координат этих векторов. Найдем все собственные векторы $\beta_1, \dots, \beta_n$, этой матрицы, которые считаем ортонормированными строками. Из этих строк составим матрицу

$$B = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_n \end{pmatrix}$$

строками которой являются данные векторы. По построению $BQB^T = \text{diag}(c_1, \dots, c_n)$. Каждый из векторов α_k раскладывается по векторам β_i ,

$$\alpha_k = \sum_i d_{ki} \beta_i, \quad (3.21)$$

и коэффициенты разложения становятся случайными величинами с нулевыми средними. Пусть матрица D составлена из коэффициентов разложения. Равенство (3.21) равносильно равенству $A = DB$, откуда в силу ортогональности B следует $D = AB^T$. Теперь

$$D^T D = BA^T AB^T = BAB^T = \text{diag}(c_1, \dots, c_n). \quad (3.22)$$

Это означает отсутствие корреляции между коэффициентами разложения исходных векторов по выбранному базису.

3.7.5 Реализация РСА

Реализация метода основана на функции `numpy.linalg.svd`. Пусть вещественная матрица A имеет размер $m \times n$. Тогда $U, D, V = \text{svd}(A)$ создает полярное разложение матрицы $A = U \cdot D \cdot V$. Здесь U, V — ортогональные матрицы размеров $m \times m, n \times n$ соответственно, а D матрица размера $m \times n$, в которой ненулевые числа $d_0, d_1, \dots$ стоят по главной диагонали, а остальные числа равны нулю. Имеем

$$A^T \cdot A = V^T \cdot D^T \cdot U^T \cdot U \cdot D \cdot V = V^T \text{diag}(d_0^2, d_1^2, \dots) V,$$

или

$$A^T \cdot A \cdot V^T = V^T \cdot \text{diag}(d_0^2, d_1^2, \dots).$$

Это означает, что строки матрицы V являются собственными векторами матрицы $Q = A^T \cdot A$ и матрица $A \cdot V^T$ состоит из коэффициентов разложения строк матрицы A по собственным векторам. Произведем эксперимент с тем же фрагментом аудио файла длины 1024. Разобьём его на части длины $Lrow$ и выберем число базисных векторов (число коэффициентов в разложении) $NumCoe$.

```
def createSignAppr(Sign,BaseVect): #Аппроксимация одной строки
    Out = np.zeros(len(Sign))
    for Vect in BaseVect:
        Scl = np.dot(Sign,Vect)
        Out += Scl*Vect
    return Out

def createOrigAppr(Matr,BaseVect): #Аппроксимация всего сигнала
    Out = []
    for Str in Matr:
        Sign = createSignAppr(Str,BaseVect)
        Out.append(Sign)
    ArrOut = np.float_(Out)
    return ArrOut.flatten()

Lrow =LROW
NumCoe = NUMCOE
Beg = 0
End = Lrow
A = []
while End<= len(Fragm):
    A.append(Fragm[Beg:End])
    Beg += Lrow
    End += Lrow
U,D,V = np.linalg.svd(np.float_(A))
BaseVect = V[:NumCoe]
SignAppr =createOrigAppr(A,BaseVect)
print(Corr(Fragm,SignAppr))
```

Для значений $LROW = 256$, $NUMCOE = 2$ на печать выдано 0.94 (почти полное совпадение, понадобилось 8 коэффициентов). Для значений $Lrow = 64$, $NUMCOE = 4$ на печать выдано 0.88 (почти полное совпадение, понадобилось 64 коэффициента). Результат значительно превосходит использование dct или wavelet, если не учитывать необходимость хранения базисных векторов. В тех случаях, когда исходный файл очень большой, этот метод может оказаться эффективным.

3.8 Внедрение водяных знаков в аудио файлы

В этом пункте покажем, как представленные методы обработки цифровых сигналов могут быть использованы для решения конкретной задачи — внедрение водяных знаков (ВЗ), или watermark (WM) в аудио файлы (носитель). Не нужно путать проблему внедрения ВЗ с задачами стеганографии. Хотя для решения обеих проблем используют схожие технологии, цели у них разные. Особенность цифровых объектов состоит в том, что их можно копировать без потери качества. В этой связи возникает проблема подтверждения авторского права, в частности, доказательство авторства аудио файлов. Одним из средств защиты таких файлов являются водяные знаки. В оригинал – носитель – внедряется специальным образом фрагмент – ВЗ, который не мешает прослушиванию модифицированного файла, однако, этот фрагмент может быть обнаружен специальной программой, которой владеет автор произведения. При этом подразумевается, что качество носителя остается высоким.

Цель стеганографии — скрыть передачу информации с помощью носителя, но к качеству носителя не предъявляют высоких требований. В качестве носителя, как правило, используют аудио файл в сжатом формате. Структура файла в сжатом формате использует большое количество коэффициентов. Внося известные изменения в эти коэффициенты, удастся передать большой объем информации. Эта задача в книге не рассматривается. Всегда предполагается, что носитель имеет wav формат.

Важной проблемой на пути создания ВЗ является его устойчивость к различным преобразованиям исходного файла: фильтрация, изменение частоты стробирования, вырезание небольших фрагментов. Увеличение мощности сигнала, содержащего ВЗ, повышает устойчивость, но вредит качеству аудио файла. Таким образом, внедрение ВЗ всегда является компромиссом между устойчивостью к атакам и отсутствием помех в модифицированном файле. Иногда помехи вносят сознательно, чтобы заставить заинтересованных лиц покупать чистый продукт у автора.

3.8.1 Выбор водяного знака

Первый вопрос, который возникает при создании ВЗ – что выбрать в качестве внедряемого сигнала. Это может быть текст, изображение или даже музыкальное произведение. Проблема устойчивости к атакам решается только при перечислении этих атак.

Рассмотрим следующий пример. Предположим, что в качестве ВЗ выбран текст. Выбрав конкретную кодировку, превратим этот текст в битовую последовательность *Bits*. При извлечении ВЗ возможны помехи. Посмотрим, что происходит при небольших искажениях таких ВЗ. Функция `random01(Prob)` случайным образом генерирует 1 с вероятно-

стью Prob и 0 с вероятностью 1-Prob. Каждый бит в строке заменится на противоположный с вероятностью Prob и сохранит свое значение с вероятностью 1-Prob.

```
from numpy import random as RND
from bitarray import bitarray

def random01(Prob):
    ''' 1 with probability Prob'''
    Val = RND.random()
    return 1 if Val < Prob else 0

S = bytes('Это мой файл', 'cp1251')
a = bitarray()
a.frombytes(S)
Prob = 0.05
RND.seed(1234)
for I in range(len(a)):
    a[I] ^= random01(Prob)
V = a.tobytes().decode('cp1251')
print(V)
```

Мы выбрали кодировку cp1251. Установим Prob = 0.05. На выходе получим строку 'Эт® мМн фвйл'. Если бы использовалась кодировка utf8, выход был бы пустой, поскольку, не распознав байт, декодер прекращает работу. При Prob > 0.1 и WINDOWS-кодировка не спасает ситуацию.

Wave file кодировка

В качестве ВЗ может быть использовано музыкальное произведение. Идея такого решения заимствована из [5]. Достоинством такого подхода является устойчивость этого водяного знака к значительным помехам. Сначала выбирается хорошо узнаваемая мелодия, например, «В лесу родилась ёлочка». Музыкальный фрагмент превращается в битовую строку. Даже при искажении значительного количества битов, человек распознает эту мелодию. Превращение музыкального файла в битовую строку осуществляется следующим образом.

```
import numpy as np
from scipy.io.wavfile import read
[FrWm, DataWm] = read('elochka.wav')
S = np.std(DataWm)
WM = np.zeros(len(DataWm)).astype(np.int16)
Indx = np.where(DataWm > S)
WM[Indx[0]] = 1
play_wave(20000*WM, FrWm)
```

Выбирается порог (в данном случае стандартное отклонение). Все значения выше порога заменяются единицами, остальные равны нулю. При проигрывании нужно увеличить амплитуду сигнала. Конечно, качество по сравнению с оригиналом ухудшится, но мелодия остается узнаваемой.

3.8.2 Внедрение ВЗ с помощью Wavelet-преобразования

Режим multiresolution подсказывает простой способ для внедрения ВЗ. Как было показано ранее на примерах, основная информация содержится в низкочастотной части преобразования. Внедрив в высокочастотную часть сигнала ВЗ и восстановив исходный сигнал, получаем первый вариант внедрения ВЗ. В качестве Wavelet возьмем 'db3'.

```
def insWM2Wave(Data,WM):
    [cL,cH] = pywt.dwt(Data,'db3')
    [cL1,cH1] = pywt.dwt(cL,'db3')
    Ln =len(cH1)
    StdH = np.std(cH1)
    cHNew = np.zeros(Ln)
    Min = min(Ln,len(WM))
    cHNew[:Min] = StdH*WM[:Min]
    cL = pywt.idwt(cL1,cHNew,'db3')
    return pywt.idwt(cL,cH,'db3')

def getWMfromWave(Data):
    [cL,cH] = pywt.dwt(Data,'db3')
    [cL1,cH1] = pywt.dwt(cL,'db3')
    return cH1
```

```
import pywt
[Fr,Data] = read(name_of_file)
Carrier = insWM2Wave(Data,WM)
RestoreWM = getWMfromWave(Carrier)
RestoreWM /= max(abs(RestoreWM))
Music = np.int16(20000*RestoreWM)
play_wave(Music,Fr)
```

К сожалению данный вариант совершенно не противостоит атакам. Если мы изменим начало массива Carrier, положив его равным Carrier[1:], мы не сможем извлечь ВЗ.

3.8.3 Внедрение ВЗ с помощью dct-преобразования

Ранее было показано, что dct является весьма эффективным средством при разложении аудио файла по ортогональному базису. Это означает,

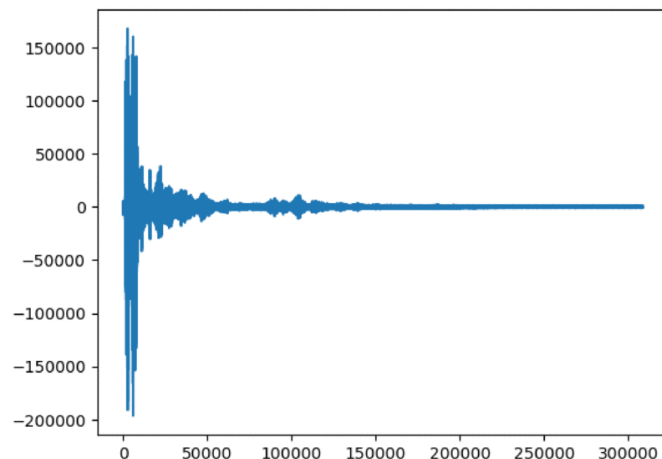


Рис. 3.11: Dct-спектр аудио файла

что в случае больших файлов коэффициенты содержат много значений близких к нулю. Это позволяет заменить их значениями ВЗ. Пример такого спектра представлен на Рис.3.11. Начиная с номера 150000 все коэффициенты можно модифицировать.

```
def putDctWM(Data,WM,Pos):
    Spect = dct(Data,norm='ortho')
    ForWM = Spect[Pos:]
    Std = np.std(ForWM)
    Ln = len(ForWM)
    Min = min(Ln,len(WM))
    ForWM[:Min] = Std*WM[:Min]/20
    return idct(Spect,norm='ortho')
```

```
def getDctWM (Carrier,Pos):
    Spec = dct(Carrier,norm='ortho')
    return Spec[Pos:]
```

Это ВЗ устойчив как к фильтрованию, так и удалению начального участка. Деление стандартного отклонения на 20 установлено экспериментально. Этот коэффициент может варьироваться в широких пределах.

3.8.4 Внедрение ВЗ с помощью интерполяции

Преимущество данного метода заключается в его простоте. Его реализация не требует каких-либо специальных пакетов. Метод устойчив к удалению произвольного фрагмента из носителя. Идея метода состоит в линейной интерполяции сигнала в точке $Sig[Pos]$, с помощью $(Sig[Pos - 1] + Sig[Pos + 1])/2$.

```

def getStdDiff(Data):
    Diff = []
    NData = np.float_(Data)
    for I in range(0,len(NData)-2,2):
        Mid = (NData[I] + NData[I+2])*0.5
        Diff.append(abs(NData[I+1]-Mid))
    return np.std(Diff)

def putInterp(In,WM):
    Carrier = np.float_(In)
    PosWM = 0
    PosIns = 1
    Std = getStdDiff(Carrier)
    Ln = len(Carrier)//2
    while PosIns < (len(Carrier) -1) and PosWM <len(WM):
        Midd = (Carrier[PosIns-1]+Carrier[PosIns+1])*0.5
        if WM[PosWM] == 1:
            Carrier[PosIns] = Midd +Std/40
        else:
            Carrier[PosIns] = Midd
        PosWM +=1
        PosIns += 2
    return Carrier

def getWMInter(In):
    Carrier = In.copy()
    Std = getStdDiff(Carrier)
    WM = []
    for I in range (1,len(Carrier)-1,2):
        Midd =(Carrier[I-1]+Carrier[I+1])*0.5
        if abs(Carrier [I] -Midd) >Std/40:
            WM.append(1)
        else:
            WM.append(0)
    return np.float_(WM)

```

Сначала оцениваем, насколько различаются само значение и интерполируемое с помощью двух соседних. После этого значение в средней точке заменяется интерполяцией, а к последней добавляется константа в зависимости от текущего значения ВЗ. Замена значения $Carrier[PosIns] \rightarrow Carrier[PosIns] = Midd + Std/40$ подбирается экспериментально. Фильтрация FIR фильтром длиной более 3 удаляет ВЗ.

Литература

- [1] *Г.Е.Шилов* "Математический анализ. Второй специальный курс." — Наука — 1965.
- [2] *А.Оппенгейм, Р.Шафер* "Цифровая обработка сигналов." — Техносфера — 2012
- [3] *С. Малла* "Вэйвлеты в обработке сигналов." — Мир — 2005.
- [4] *Р.Лайонс* "Цифровая обработка сигналов " —М. БИНОМ — 2006.
- [5] *Absalyamova, K.S., Latypov, R.K., Stolov, E.L* "Ternary Code of Melody and Reliable Audio Watermarking" —27th Telecommunications Forum, TELFOR 2019 —2019
- [6] *А.Б.Дауни* "Цифровая обработка сигналов на языке Python " — М.ДМК Пресс —2017
- [7] *Е.Л. Столов* "Цифровая обработка сигналов. Водяные знаки в аудиофайлах" —Лань —2018