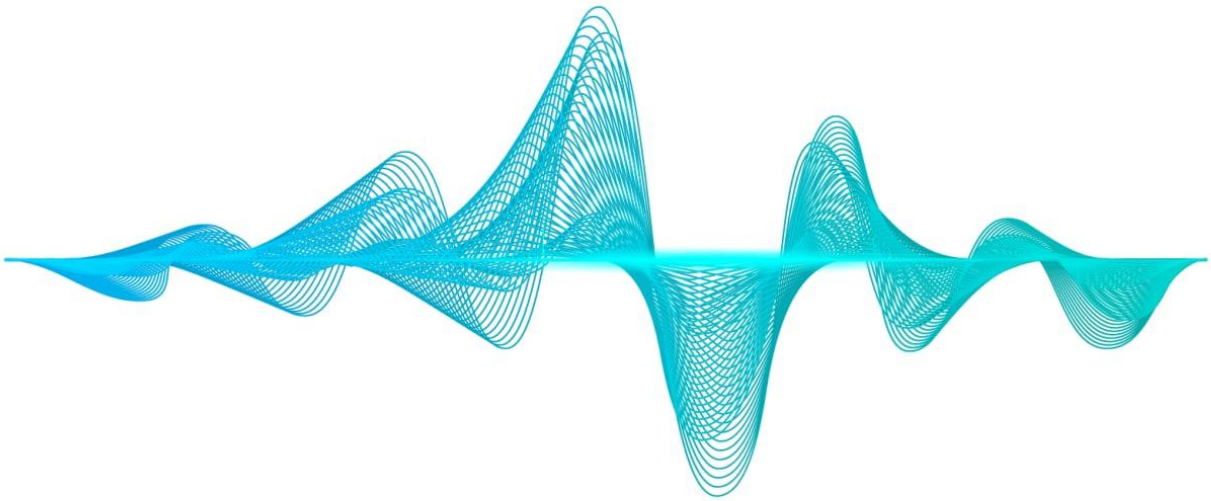
УНИВЕРСИТЕТ ИТМО

В.А. Волохов, О.В. Махныткина,
И.Д. Мещеряков, Е.В. Шуранов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО КУРСУ «ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ»



Санкт-Петербург
2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

В.А. Волохов, О.В. Махныткина,
И.Д. Мещеряков, Е.В. Шуранов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО КУРСУ «ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ»

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.04.02 Информационные системы и технологии
в качестве Учебно-методическое пособие для реализации основных профессиональных образовательных программ высшего образования магистратуры

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2022

Волохов В.А., Махныткина О.В., Мещеряков И.Д., Шуранов Е.В., Методические указания к выполнению лабораторных работ по курсу «Цифровая обработка сигналов» – СПб: Университет ИТМО, 2021. – 60 с.

Рецензент(ы):
Рыбин Сергей Витальевич, кандидат физико-математических наук, доцент (квалификационная категория "доцент практики") факультета информационных технологий и программирования, Университета ИТМО.

Лабораторные работы по курсу «Цифровая обработка сигналов». Предназначены для специальностей, ориентированных на обработку звука. Помимо практики в классических задачах ЦОС в лабораторных работах рассматриваются способы извлечения аудиопризнаков из сигнала для решения наиболее распространённых задач аудиообработки (обработка речи, анализ звука).

Университет ИТМО – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

Содержание

ВВЕДЕНИЕ

Цифровая обработка сигналов является актуальным направлением для многих задач обработки речевых сигналов, например, для распознавания речи, голосовой биометрии, синтеза речи [1] и т.д. В данном методическом пособии приводятся дополнительные пояснения к выполнению лабораторных работ по курсу «Цифровая обработка сигналов» для студентов высших учебных заведений, обучающихся по направлениям подготовки, связанным с обработкой звуковых (речевых) сигналов. Шаблоны лабораторных работ находятся в github-репозитории [2].

Учебно-методическое пособие включает в себя 4 лабораторные работы, охватывающие основные темы цифровой обработки звуковых сигналов – «Сигналы и аудиофайлы», «Анализ сигналов», «Простейшие фильтры» и «Акустические признаки». Для каждой лабораторной работы поставлены цели и задачи, предоставлены краткие теоретические сведения, позволяющие лучше разобраться в функциях, применяемых в лабораторной работе, а также список заданий и описания к ним. Помимо этого, в пособии указан список литературы, позволяющий более подробно рассмотреть некоторые вопросы цифровой обработки сигналов, находящиеся на стыке с областью машинного обучения.

ЛАБОРАТОРНАЯ РАБОТА № 1 «Сигналы и аудиофайлы»

Цель работы: познакомиться с основными методами работы с аудиофайлами в Python; разобраться в том, как работает свертка, и научиться применять различные фильтры.

Краткие теоретические сведения

Сигнал (в теории информации и связи) — носитель информации, используемый для передачи сообщений в системе связи.
Обработка сигнала подразумевает под собой его анализ или синтез:

- анализ заключается в понимании информации, переносимой сигналом;
- синтез заключается в создании сигнала, содержащего заданную информацию.

Основными концепциями цифровой обработки сигналов являются:

- дискретизация по времени — преобразование сигнала, непрерывного во времени, в сигнал, заданный отдельными значениями в дискретные моменты времени;
- дискретизация по амплитуде (квантование) — преобразование некоторой величины с непрерывной шкалой значений в величину, имеющую дискретную шкалу значений.

Звук – это аналоговый сигнал, то есть он является непрерывным по времени и по значениям. Для того, чтобы работать со звуком на цифровом устройстве, надо преобразовать его в цифровое представление. Для этого надо разделить непрерывный сигнал на промежутки времени (дискретизация сигнала) и разбить непрерывные значения на интервалы (квантование сигнала). Оптимальный шаг дискретизации определяется теоремой Котельникова (теорема отсчётов).

Теорема Котельникова. Непрерывный сигнал $x(t)$ можно представить в виде интерполяционного ряда:

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT_Д) \frac{sin\left(\frac{\pi(t - nT_Д)}{T_Д}\right)}{\frac{\pi(t - nT_Д)}{T_Д}},$$

где $T_Д$ – период дискретизации (интервал Найквиста). Следствие из этой теоремы звучит следующим образом: любую функцию (сигнал) $S(t)$, состоящую из частот от 0 до F_B периодов в секунду, можно непрерывно передавать с любой точностью с помощью чисел, следующих друг за другом через интервалы времени $T_Д = 1/(2F_B)$ секунд. Выбранные параметры дискретизации и квантования сигнала напрямую влияют на качество цифрового сигнала. В соответствии с теоремой требуется, чтобы частота дискретизации аналогового сигнала была, по крайней мере, вдвое больше полосы полезного сигнала, иначе информация об исходном виде

аналогового сигнала будет потеряна. Если выбрать частоту дискретизации меньше (а в большинстве практических устройства и равной) удвоенной полосы частот преобразуемого аналогового сигнала, то возникает эффект, известный как наложение спектра.

Цифровой сигнал формируется с использованием аналого-цифрового преобразователя. Таким образом, цифровой сигнал является дискретным, квантованным и закодированным (прямой, обратный и дополнительный коды).

При этом восприятие звука человеком построено так, что он воспринимает частоты гармонических составляющих аудиосигнала, а потому под звуком всегда понимается набор гармонических сигналов, на которые этот звук можно разложить.

Гармонические колебания – колебания, при которых физическая величина изменяется с течением времени по гармоническому (синусоидальному или косинусоидальному) закону. В общем случае гармонические колебания задаются формулой:

$$y = A\cos(\omega t + \varphi),$$

где A – это амплитуда, ω – циклическая частота (рад/с), φ – фаза (сдвиг), t – время.

Одна из наиболее частых операций, которая выполняется при обработке сигналов, – это свёртка. Свёртка имеет много различных применений, например, с ее помощью можно убрать из сигнала шумы или применить к сигналу эффект эха. Для того, чтобы разобраться, что такое свертка, нужно сначала понять, что такое взаимная корреляция.

Взаимная корреляция – это функция оценки корреляции двух последовательностей $h(n)$ и $x(n)$, вычисляемая по формуле:

$$y(n) = \sum_{u=-\infty}^{\infty} h(u)x(n+u).$$

Свёртка — это взаимная корреляция, в которой один из двух сигналов инвертируется по временной оси перед началом выполнения операции. Формула свёртки:

$$y(n) = \sum_{u=-\infty}^{\infty} h(u)x(n-u).$$

Выше были рассмотрены базовые принципы анализа сигналов. Далее рассмотрим синтез.

Алгоритм Карплуса-Стронга — алгоритм синтеза звука, заключающийся в пропускании короткого сигнала через линию задержки с фильтром. В зависимости от параметров полученный звук может быть похож на звук струны,

извлекаемый медиатором или тэппингом, либо на звуки некоторых ударных инструментов. В основе алгоритма лежит рекурсивный фильтр порядка *M*. (см. рисунок 1).

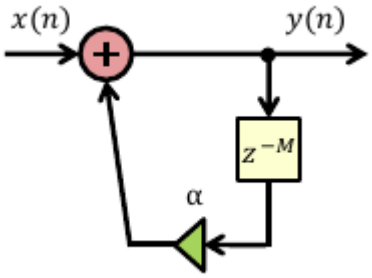


Рисунок 1 — Алгоритм Карплуса-Стронга, где z – задержка порядка M, α – множитель.

Основными этапами алгоритма являются:

1. Генерируется короткий сигнал белого шума (длина M отсчетов).
2. Сигнал подается на вход линии задержки длиной M отсчетов.
3. Выход линии задержки пропускается через звуковой фильтр.
4. Для поддержания стабильной положительной обратной связи коэффициент передачи фильтра должен быть строго меньше 1 для всех частот. В качестве фильтра может использоваться однополюсной фильтр низких частот. Характеристики фильтра являются определяющими для гармонической структуры затухающего сигнала. *В лабораторной работе предложено реализовать низкочастотный фильтр, выполняющий операцию арифметического усреднения двух соседних отсчётов.*
5. Выход фильтра микшируется с генерируемым коротким сигналом и подается на вход линии задержки.

Задание 1

1. Какая метаинформация хранится в .wav файле?
2. Запишите аудиофайл со своим голосом с помощью программы *Audacity* или *Adobe Audition*. Загрузите его в лабораторную и нарисуйте форму волны считанного файла.
3. Поменяйте частоту дискретизации (увеличьте и уменьшите в 2 раза) и прослушайте получившуюся запись. Что изменилось? При какой частоте дискретизации становится невозможно разобрать человеческую речь? Чем .wav отличается от других кодеков, например, .mp3 или .ogg?

Задание 2

1. Загрузите файл *data.pickle* и прочитайте сериализованные данные.
2. Постройте график трех сигналов *a*, *b*, *c* из раздела *«task2»* файла с помощью функции *draw_signal*, основанной на *matplotlib.pyplot.plot*.
3. Попробуйте подобрать коэффициенты для этих сигналов. Сгенерируйте сигналы (1000 отсчетов) с подобранными коэффициентами. Постройте

графики сгенерированных сигналов и пройдите тест (основанный на функции *numpy.allclose*) на схожесть с оригинальным. Выпишите коэффициенты каждого из сигналов в специальные поля шаблона.

Задание 3

1. Реализуйте операцию свертки с помощью библиотеки *numpy*.
2. Сравните её с существующей реализацией *scipy.signal.convolve*.
3. Постройте графики фильтра (сигнал *b*), исходного сигнала (сигнал *a*) и результата свертки сигналов (т.е. свертки *a* и *b*).

Задание 4*

1. Реализуйте алгоритм Карплуса-Стронга. В качестве фильтра используйте усреднитель двух смежных отсчетов. Проверьте результат работы функции тестами из шаблона.
2. Отрисуйте и воспроизведите полученный сигнал. На что влияют параметры генерации?
3. Попробуйте имитировать звучание разных струн гитары. Визуализируйте полученные графики и подпишите названия нот.

Примечания

В задании 1 для считывания файлов воспользуйтесь библиотекой *scipy* или *librosa*. Для воспроизведения аудиофайла удобно использовать класс *Audio* из модуля *IPython.display*, а для отрисовки – *matplotlib*.

В задании 2 фаза, период и амплитуда сигнала – целочисленные. Для генерации пользуйтесь библиотекой *numpy* и функциями *arange*, *sin*, *cos*. Для сигнала укажите все коэффициенты, которыми описывается сигнал.

В задании 3 функцию свертки можно реализовать без библиотеки *numpy* через цикл.

Задание 4 – дополнительное и не является обязательным, но рекомендуется тем студентам, которые планируют дипломные работы, связанные с данным направлением. Также задания со звездочкой могут быть учтены в рекомендациях освобождения от экзамена (требования к освобождению от экзамена уточняются в каждом семестре с преподавателем, ведущим лекции).

Выполнение работы и критерии приемки

В первой лабораторной работе «Сигналы и аудиофайлы» необходимо в заданном шаблоне исходных кодов (Приложение А) успешно реализовать следующие функции:

1) Сгенерировать сигналы a, b и c в соответствии с описанием;
2) Свертка сигналов;
3) Отобразить графики a, b, c и свёрток;
4) * Алгоритм Карплуса-Стронга
    a. Генерация гитарных нот;

b. Затухание амплитуд и ее визуализация;

и ответить на 3 вопроса:

1) Что хранится в .wav файле? Как узнать параметры дискретизации и квантования .wav файла?
2) Запишите аудиофайл со своим голосом. Загрузите его. Попробуйте поменять ему частоту дискретизации. Нарисуйте форму волны считанного файла. Воспроизведите полученные сигналы. При какой частоте дискретизации становится невозможно разобрать человеческую речь?
3) Чем .wav отличается от других кодеков, например, .mp3 или .ogg?

В шаблоне исходных кодов (Приложение А) предусмотрены автоматические проверки для всех реализуемых студентом функций. Они помогают понять, все ли сделано правильно. Прохождение автоматических проверок является необходимым условием выполнения работы.

ЛАБОРАТОРНАЯ РАБОТА № 2 «Анализ сигналов»

Цель работы: познакомиться с основными видами сигналов и изучить методы анализа цифровых сигналов

Краткие теоретические сведения

Примеры сигналов дискретного времени:

1. Единичный импульс. Изображение единичного импульса представлено на рисунке 2.

$$x(n) = \delta(n) = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases}$$

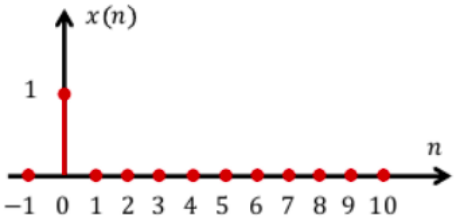


Рисунок 2 – Единичный импульс

2. Единичный скачок. Изображение единичного скачка представлено на рисунке 3.

$$x(n) = \delta(n) = \begin{cases} 1, n \geq 0 \\ 0, n < 0 \end{cases}$$

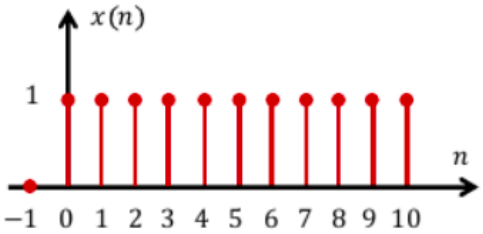


Рисунок 3 – Единичный скачок

3. Синусоида. Формула построения сигнала:

$$x(n) = A_0 sin(\hat{\omega}_0 n + \varphi_0),$$

где A_0 – амплитуда синусоиды, $\varpi_0 = \omega_0 T_Д = 2\pi f_0 T_Д$ – циклическая частота, f_0 – линейная частота синусоиды, $T_Д$ – период дискретизации. Изображение синусоиды представлено на рисунке 4.

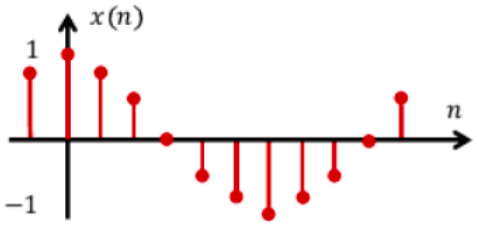


Рисунок 4 – Синусоида

Многие сигналы удобно анализировать, раскладывая их на синусоиды (гармоники). Тому есть несколько причин. Например, подобным образом работает человеческое ухо. Оно раскладывает звук на отдельные колебания различных частот. Кроме того, можно показать, что синусоиды являются «собственными функциями» линейных систем (т.к. они проходят через линейные системы, не изменяя формы, а могут изменять лишь фазу и амплитуду). Еще одна причина в том, что теорема Котельникова формулируется в терминах спектра сигнала.

Преобразование Фурье (Fourier transform) – операция, сопоставляющая одной функции вещественной переменной другую функцию вещественной переменной. Эта новая функция описывает коэффициенты («амплитуды») при разложении исходной функции на элементарные составляющие — гармонические колебания с разными частотами (подобно тому, как музыкальный аккорд может быть выражен в виде суммы музыкальных звуков, которые его составляют).

Существует несколько видов преобразования Фурье для дискретного времени:

1. Классическое преобразование Фурье дискретного времени.
2. Быстрое преобразование Фурье.
3. Оконное преобразование Фурье.

Компьютер способен работать только с ограниченным объемом данных, следовательно, реально он способен вычислять только последний вид преобразования Фурье. Рассмотрим его подробнее.

Преобразование Фурье для последовательностей конечной длины – это преобразование Фурье для дискретных сигналов. Формула анализа (переход от временной области к частотной):

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \qquad k = 0,1,\dots,N-1.$$

Формула синтеза (переход от частотной области к временной):

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn}, \qquad n = 0,1,\dots,N-1.$$

При этом переход от временной области к частотной области не изменяет энергию сигнала (равенство Парсеваля):

$$\sum_{n=0}^{N-1}|x(n)|^2=\frac{1}{N}\sum_{k=0}^{N-1}|X(k)|^2.$$

Дискретное преобразование Фурье обладает свойством скрытой периодичности, например, $X(N)=X(0),\ X(N+1)=X(1)$ и т. д. При явно заданной периодичности Фурье преобразования термин дискретное преобразование Фурье заменяется на дискретный ряд Фурье:

• Дискретный ряд Фурье отображает *N*-периодический сигнал в *N*-периодическую последовательность коэффициентов;

• Обратный дискретный ряд Фурье отображает *N*-периодическую последовательность коэффициентов в *N*-периодический сигнал;

• Дискретный ряд Фурье *N*-периодического сигнала является математическим эквивалентом к дискретному преобразованию Фурье одного периода.

Вычисление преобразований Фурье требует очень большого числа умножений (около N^2). Существует способ выполнить эти преобразования значительно быстрее: примерно за $N\cdot log_2 N$ операций умножения. Этот способ называется быстрым преобразованием Фурье (БПФ, FFT, *Fast Fourier Transform*).

Идея простейшего алгоритма БПФ с прореживанием по времени состоит в том, чтобы разбить исходный *N*-точечный сигнал $x(n)$ на два более коротких сигнала, ДПФ которых могут быть скомбинированы таким образом, чтобы получить ДПФ исходного *N*-точечного сигнала.

Так, если исходный *N*-точечный сигнал разбить на два $N/2$-точечных сигнала, то для вычисления ДПФ каждого из них потребуется около $(N/2)^2$ комплексных умножений. Тогда для вычисления искомого *N*-точечного ДПФ потребуется порядка $2(N/2)^2=N^2/2$ комплексных умножений, т.е. вдвое меньше по сравнению с прямым вычислением. Операцию разбиения можно повторить, вычисляя вместо $(N/2)$-точечного ДПФ два $(N/4)$-точечных ДПФ и сокращая тем самым объем вычислений еще в два раза. Выигрыш в два раза является приблизительным, поскольку не учитывается, каким образом из ДПФ меньшего размера образуется искомое *N*-отсчетное ДПФ.

Существует большое количество алгоритмов БПФ. Однако все они являются частными случаями единого алгоритма, базирующегося на задаче разбиения одного массива чисел на два. Тот факт, что это можно сделать более чем одним способом, определяет многообразие алгоритмов БПФ. Однако у большинства алгоритмов БПФ есть особенность: они способны работать лишь тогда, когда длина анализируемого сигнала *N* является степенью двойки. Обычно это не представляет большой проблемы, так как анализируемый сигнал всегда можно

дополнить нулями до необходимого размера. Число N называется размером или длиной БПФ (FFT size).

При этом следует подчеркнуть, что алгоритм БПФ является точным. Он даже точнее стандартного, так как, сокращая число операций, он приводит к меньшим ошибкам округления.

Поскольку преобразование Фурье определено в комплексной области, анализировать результат напрямую проблематично. Для анализа сигналов принято разбивать результат преобразования Фурье на два спектра: амплитудный и фазовый. Амплитудный показывает, как меняется модуль сигнала, а фазовый – как меняется угол, образуемый на комплексной плоскости. Графически частотные спектры изображают в виде отрезков длиной $A_n(\varphi_n)$, проведенных перпендикулярно к оси, на которую наносятся значения $0,\ \omega_1,\ \omega_2 = 2\omega_1,\ \omega_3 = 3\omega_1,\ \ldots$ (см. рисунок 5). Спектр периодического сигнала называется линейчатым или дискретным, так как состоит из отдельных линий, соответствующих дискретным частотам $\omega = n * \omega_1\ (n\ =\ 0, 1, 2, \ldots)$.

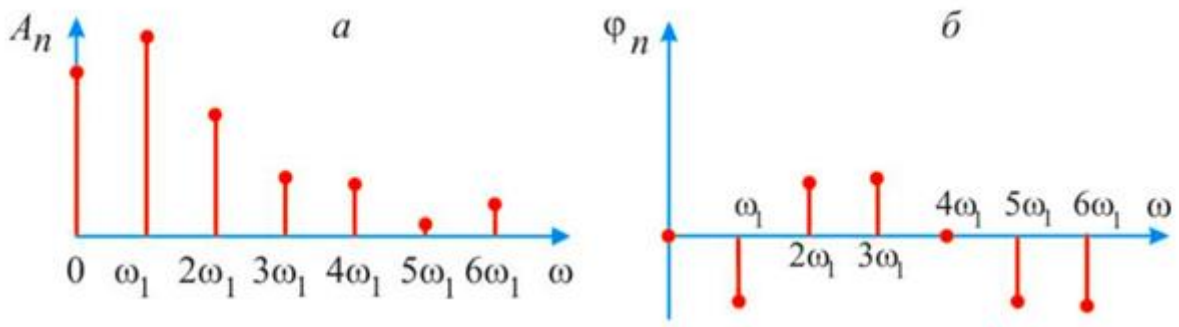


Рисунок 5 – Спектральные диаграммы периодического сигнала:
а – амплитудная; б – фазовая

Формула амплитудного спектра:

$$X(i) = Re(i) + j \cdot Im(i) = |X(i)| \cdot e^{j\frac{2\pi}{N}\Phi(i)},$$
$$A(i) = |X(i)|.$$

Формула фазового спектра:

$$\Phi(i) = arctan\left(\frac{Im(i)}{Re(i)}\right).$$

При анализе сигналов преобразование Фурье является хорошим инструментом для определения изменений частотного спектра сигнала, но ничем не может помочь при оценке времени их изменения. Помимо этого, его базисные функции – синусы и косинусы – имеют бесконечный носитель (т.е. отличны от нуля на всей числовой прямой). Это приводит к невозможности обнаружения

отдельных особенностей в сигнале, и только глобальные изменения улавливаются достаточно хорошо. Один из возможных путей устранения этого недостатка преобразования Фурье – разделение временной оси с использованием так называемой оконной функции. Преобразование Фурье оконной версии сигнала дает возможность связать частотный спектр с частью сигнала, локализованной в окне.

Идея этого преобразования очень проста: временной интервал существования сигнала разбивается на ряд промежутков — временных окон. В каждом промежутке вычисляется свое преобразования Фурье. Если в каком-то окне существовали частотные составляющие некоторого сигнала, то они будут присутствовать в спектре, а если нет — будут отсутствовать. Таким образом, можно перейти к частотно-временному представлению сигналов.

Кратковременное (оконное) преобразование выполняется с использованием выражения:

$$X(k,m) = \sum_{n=0}^{L-1} x(n+m)w(n)e^{-j\frac{2\pi}{L}kn},$$

где $w(n)$ – оконная функция, m – шаг окна (сдвиг на временной оси), L – длина сигнала. Таким образом, окно, перемещаясь, позволяет «просмотреть» весь сигнал. В каждом окне выполняется свое спектральное разложение, так что вместо обычно одной спектрограммы получается набор спектрограмм. Поскольку каждое окно охватывает небольшой участок по времени, точность описания локальных изменений сигнала может быть повышена. Часто используются окна Гаусса или иные окна, обеспечивающие малые искажения спектра из-за граничных явлений и уменьшающие проявление эффекта Гиббса.

Основные сложности в использовании оконного преобразования Фурье заключаются в выборе оптимальной длины, сдвига и формы окна в конкретной задаче.

Широкое окно позволяет получить узкополосную спектрограмму с более высоким частотным разрешением, однако при этом больше событий может произойти, и, как следствие, наблюдается малая точность по времени. С узким окном получается узкополосная спектрограмма, позволяющая получить точную позицию смены событий во временной области, одновременно с этим ДПФ будет содержать небольшое количество точек, и, как следствие, низкое частотное разрешение. Малый сдвиг окна приведёт к высокой вычислительной сложности, однако при этом будет достигаться малая вероятность пропуска события. С большим сдвигом окна ситуация противоположная — малая вычислительная сложность и большая вероятность пропуска событий. При этом форма окна влияет на искажение получившегося спектра, которые особенно заметны в областях резкой смены спектрального уровня, а также областях с константными уровнями. Наиболее обычным на практике для речевой обработки является окно Хэмминга.

Теперь рассмотрим более сложный сигнал – DTMF.

Dual-Tone Multi-Frequency (DTMF) – двухтональный многочастотный аналоговый сигнал, используемый для набора телефонного номера. Сфера применения тональных сигналов – автоматическая телефонная сигнализация между устройствами. Для кодирования символа в DTMF сигнал необходимо сложить два синусоидальных сигнала. Частоты синусоид берутся по приведённой ниже таблице из рисунка 6 из столбца и строки, соответствующих передаваемому символу. Каждый символ является ссылкой на звуковой файл, воспроизводящий соответствующий сигнал.

1	2	3	A	**697 Гц**
4	5	6	B	**770 Гц**
7	8	9	C	**852 Гц**
*	0	#	D	**941 Гц**
1209 Гц	**1336 Гц**	**1477 Гц**	**1633 Гц**	

Рисунок 6 – таблица передаваемых символов

Задание 1

1. Реализуйте дискретное преобразование Фурье.
2. Сравните результат с реализацией *scipy.fft.fft*.
3. Пройдите тесты, указанные в шаблонах.

Задание 1.1*

1. Реализуйте быстрое преобразование Фурье.
2. Сравните результат с *scipy.fft.fft*.
3. Пройдите тесты, указанные в шаблонах.

Задание 2

1. Реализуйте функции построения амплитудного и фазового спектра.
2. Пройдите тесты, указанные в шаблонах.
3. Постройте спектры для единичного импульса, единичного скачка и синусоиды и ответьте на вопросы:
    A. Почему амплитуда всех гармоник единичного импульса равна единице?
    B. Какие выводы можно сделать, смотря на амплитудный спектр? А на фазовый?
    C. Как перевести номер отсчета в частоту?
    D. Что в фазовом спектре является полезной информацией, а что – шумом? Почему?

Задание 3

1. Реализуйте оконное преобразование Фурье, пользуясь только библиотеками *numpy* и *scipy*.

2. Сравните результат с *librosa.stft*.
3. Пройдите тесты, указанные в шаблонах.

Задание 4

1. Проанализируйте сигнал паровозного гудка, состоящего из нескольких основных гармоник и шума.
2. Определите 3 основные гармоники сигнала. Ответ округлите до двух знаков после запятой.
3. Пройдите тесты, указанные в шаблонах.

Задание 5

1. Загрузите файл *«dtmf.wav»*.
2. Определите, используя таблицу, что за номер в нём закодирован.
3. Пройдите тесты, указанные в шаблонах.

Выполнение работы и критерии приемки

В первой лабораторной работе «Сигналы и аудиофайлы» необходимо в заданном шаблоне исходных кодов (Приложение Б) успешно реализовать следующие функции:

1) Дискретное преобразование Фурье;
2) Быстрое преобразование Фурье;
3) Построение амплитудного и фазового спектра;
4) Оконное преобразование Фурье;

и ответить на 6 вопросов:

1) Какие 3 гармоники составляют сигнал паровозного гудка?
2) Какой номер закодирован в аудиофайле *«dtmf.wav»*?
3) Почему амплитуда всех гармоник единичного импульса равна единице?
4) Какие выводы можно сделать, смотря на амплитудный спектр? А на фазовый?
5) Как перевести номер отсчета в частоту?
6) Что в фазовом спектре является полезной информацией, а что – шумом? Почему?

В шаблоне исходных кодов (Приложение Б) предусмотрены автоматические проверки для всех реализуемых студентом функций. Они помогают понять, все ли сделано правильно. Прохождение автоматических проверок является необходимым условием выполнения работы.

ЛАБОРАТОРНАЯ РАБОТА № 3 «Простейшие фильтры»

Цель работы: познакомиться с понятием фильтра, попробовать реализовать простейшие фильтры и научиться их применять в анализе сигналов.

Краткие теоретические сведения

Цифровой фильтр – это любая цифровая система, которая согласно заданному алгоритму осуществляет извлечение цифрового сигнала либо его параметров из действующей на входе системы смеси сигнала с помехой. Среди цифровых фильтров выделяют, например, частотно-избирательные фильтры (фильтр нижних частот – ФНЧ, фильтр верхних частот – ФВЧ, полосовой фильтр – ПФ и режекторный фильтр – РФ), фазовые корректоры (всепропускающий фильтр) и т. д.

Линейный инвариантный к сдвигу фильтр (ЛИС-фильтр) обладает следующими свойствами:

1. Линейность:

$$L\{a_1x_1(n) + a_2x_2(n)\} = a_1L\{x_1(n)\} + a_2L\{x_2(n)\}.$$

2. Инвариантность к сдвигу:

$$L\{x(n)\} = y(n), L\{x(n - n_0)\} = y(n - n_0).$$

У физически реализуемого фильтра реакция не может возникать раньше воздействия.

Основными характеристиками стационарных линейных дискретных фильтров являются:

1. Импульсная характеристика.
2. Переходная характеристика.
3. Амплитудно-частотная характеристика.
4. Фазо-частотная характеристика.

Импульсной характеристикой фильтра называется его реакция на единичный импульс при нулевых начальных условиях.

Переходной характеристикой фильтра называется его реакция на воздействие в форме единичного скачка.

Амплитудно-частотной характеристикой фильтра (АЧХ) называется зависимость амплитуды установившихся колебаний выходного сигнала от частоты её входного гармонического сигнала. Другими словами, АЧХ показывает, как влияет фильтр на амплитуду разных частот входного сигнала.

Фазо-частотной характеристикой фильтра (ФЧХ) называется зависимость фазы установившихся колебаний выходного сигнала от частоты её входного гармонического сигнала. Аналогична АЧХ, только показывает влияние на фазу сигнала.

В данной работе мы будем рассматривать два линейных инвариантных к сдвигу фильтра: фильтр с конечной импульсной характеристикой (КИХ-фильтр) и фильтр с бесконечной импульсной характеристикой (БИХ-фильтр).

Фильтр с конечной импульсной характеристикой (нерекурсивный фильтр, КИХ-фильтр) или FIR-фильтр (FIR сокр. от finite impulse response — конечная импульсная характеристика) — один из видов линейных цифровых фильтров, характерной особенностью которого является ограниченность по времени его импульсной характеристики (с какого-то момента времени она становится точно равной нулю). Такой фильтр называют ещё нерекурсивным из-за отсутствия обратной связи. Знаменатель передаточной функции такого фильтра — константа. КИХ-фильтр первого порядка представлен на рисунке 7.

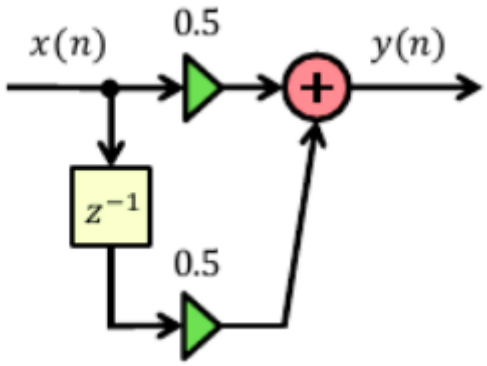


Рисунок 7 — Нерекурсивный КИХ-фильтр первого порядка

КИХ-фильтр первого порядка, изображенный на рисунке 7, представляет из себя арифметическое усреднение и описывается следующим линейным разностным уравнением:

$$y(n) = \frac{x(n) + x(n-1)}{2}.$$

Импульсная характеристика КИХ-фильтра:

$$h(n) = \frac{\delta(n) + \delta(n-1)}{2}.$$

Переходная характеристика КИХ-фильтра:

$$g(n) = \frac{u(n) + u(n-1)}{2}.$$

Амплитудно-частотная характеристика (АЧХ) КИХ-фильтра:

$$|H(e^{j\hat{\omega}})| = \sqrt{\frac{1+cos(\hat{\omega})}{2}}.$$

Фазочастотная характеристика (ФЧХ) КИХ-фильтра:

$$arg\left(H(e^{j\hat{\omega}})\right) = -arctan\left(\frac{\sin(\hat{\omega})}{1+\cos(\hat{\omega})}\right).$$

Фильтр с бесконечной импульсной характеристикой (рекурсивный фильтр, БИХ-фильтр) или IIR-фильтр (IIR сокр. от infinite impulse response — бесконечная импульсная характеристика) — линейный фильтр, использующий один или более своих выходов в качестве входа, то есть образующий обратную связь. Основным свойством таких фильтров является то, что их импульсная переходная характеристика имеет бесконечную длину во временной области, а передаточная функция имеет дробно-рациональный вид. Такие фильтры могут быть как аналоговыми, так и цифровыми. Рекурсивный БИХ-фильтр первого порядка представлен на рисунке 8.

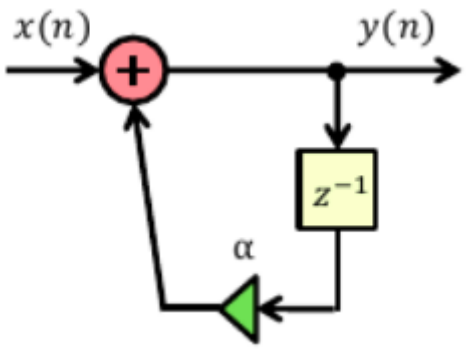


Рисунок 8 — Рекурсивный БИХ-фильтр первого порядка

БИХ-фильтр первого порядка представляет из себя систему «с памятью» (обратной связью) и описывается следующим линейным разностным уравнением:

$$y(n) = x(n) + ay(n-1).$$

Импульсная характеристика БИХ-фильтра:

$$h(n) = \delta(n) + ah(n-1) = a^n u(n).$$

Переходная характеристика БИХ-фильтра:

$$g(n) = u(n) + ag(n-1) = \left(\frac{1}{1-\alpha} - \frac{a^{n+1}}{1-\alpha}\right)u(n).$$

Амплитудно-частотная характеристика (АЧХ) БИХ-фильтра (при $\alpha > 0$ – фильтр низких частот, при $\alpha < 0$ – фильтр верхних частот):

$$|H(e^{j\widehat{\omega}})| = \frac{1}{\sqrt{1 - 2\alpha cos(\widehat{\omega}) + \alpha^2}}.$$

Фазочастотная характеристика (ФЧХ) БИХ-фильтра:

$$arg\left(H(e^{j\widehat{\omega}})\right) = -arctan\left(\frac{\alpha \cdot \sin(\widehat{\omega})}{1 + \alpha \cdot \cos(\widehat{\omega})}\right).$$

Задание 1

1. Реализуйте КИХ-фильтр и БИХ-фильтр.
2. Пройдите тесты, указанные в шаблонах.

Задание 2

1. Определите единичный импульс длиной 20 отсчетов (переменная *impulse*).
2. Получите импульсную характеристику фильтров (переменные *FIR_filter* и *IIR_filter*).
3. Постройте соответствующие графики.
4. Пройдите тесты, указанные в шаблонах.

Задание 3

1. Определите единичный скачок длиной 20 отсчетов (переменная *step*).
2. Получите импульсную характеристику фильтров.
3. Постройте соответствующие графики.
4. Пройдите тесты, указанные в шаблонах.

Задание 4

1. Получите амплитудно-частотную характеристику фильтров для единичного импульса (переменная *impulse*).
2. Постройте соответствующие графики.
3. Пройдите тесты, указанные в шаблонах.

Задание 5

1. Получите фазово-частотную характеристику фильтров для единичного импульса (переменная *impulse*).
2. Постройте соответствующие графики.
3. Пройдите тесты, указанные в шаблонах.

Выполнение работы и критерии приемки

В первой лабораторной работе «Сигналы и аудиофайлы» необходимо в заданном шаблоне исходных кодов (Приложение В) успешно реализовать следующие функции:

1) КИХ и БИХ фильтры;
2) Импульсная характеристика единичного импульса;
3) Импульсная характеристика единичного скачка;
4) АЧХ единичного импульса;
5) ФЧХ единичного импульса;

В шаблоне исходных кодов (Приложение В) предусмотрены автоматические проверки для всех реализуемых студентом функций. Они помогают понять, все ли сделано правильно. Прохождение автоматических проверок является необходимым условием выполнения работы.

ЛАБОРАТОРНАЯ РАБОТА № 4 «Акустические признаки»

Цель работы: познакомиться с основными видами акустических признаков и научиться их извлекать.

Краткие теоретические сведения

Спектрограмма — изображение, показывающее зависимость спектральной плотности мощности сигнала от времени. Спектрограммы применяются для идентификации речи, анализа звуков животных, в различных областях музыки, радио- и гидролокации, обработке речи, сейсмологии и в других областях.

Наиболее распространенным представлением спектрограммы является двумерная диаграмма: на горизонтальной оси представлено время, по вертикальной оси — частота; третье измерение с указанием амплитуды на определенной частоте в конкретный момент времени представлено интенсивностью или цветом каждой точки изображения (см. рисунок 9).

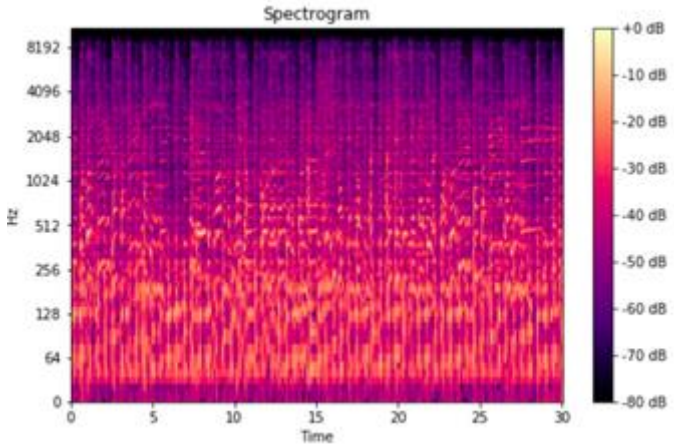


Рисунок 9 — Спектрограмма

Спектрограмма обычно создаётся одним из двух способов: аппроксимируется как набор фильтров, полученных из серии полосовых фильтров (это был единственный способ до появления современных методов цифровой обработки сигналов), или рассчитывается по сигналу времени, используя оконное преобразование Фурье. Эти два способа фактически образуют разные квадратичные частотно-временные распределения, но эквивалентны при некоторых условиях.

Метод полосовых фильтров обычно используется в аналоговой обработке для разделения входного сигнала на частотные диапазоны.

Создание спектрограммы с помощью оконного преобразования Фурье обычно выполняется методами цифровой обработки. Производится цифровая выборка данных во временной области. Сигнал разбивается на части, которые, как правило, перекрываются, и затем производится преобразование Фурье, чтобы рассчитать величину частотного спектра для каждой части. Каждая часть соответствует вертикальной линии на изображении — значение амплитуды в

зависимости от частоты в каждый момент времени. Спектры или временные графики располагаются рядом на изображении или трёхмерной диаграмме.

Спектрограмма сигнала *s(t)* может быть оценена путём вычисления квадрата амплитуды оконного преобразования Фурье сигнала *s(t)*. Эксперименты ученых показали, что человеческое ухо более чувствительно к изменениям звука на низких частотах, чем на высоких. То есть, если частота звука изменится со 100 Гц на 120 Гц, человек с очень высокой вероятностью заметит это изменение. А вот если частота изменится с 10000 Гц на 10020 Гц, это изменение человеческое ухо вряд ли сможет уловить.

В связи с этим была введена новая единица измерения высоты звука — мел. Она основана на психо-физиологическом восприятии звука человеком и логарифмически зависит от частоты (см. рисунок 10):

$$mel = 1127.01048 \cdot \ln\left(1 + \frac{freq}{700}\right).$$

Сравнение линейной и мел-шкалы представлено на рисунке 10.

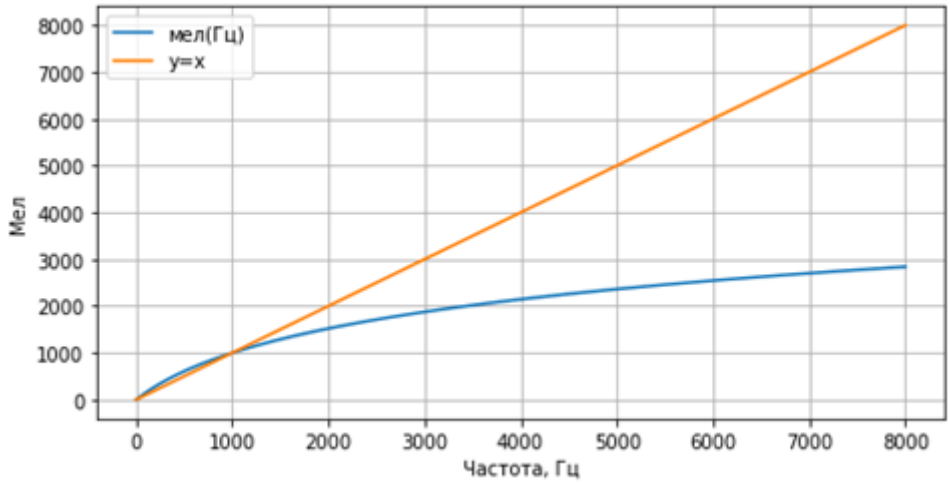


Рисунок 10 — Сравнение линейной и мел-шкалы

Таким образом, на практике для анализа речи обычно используется мел-спектрограмма. Мел-спектрограмма — это обычная спектрограмма, где частота выражена не в Гц, а в мелах. Переход к мелам осуществляется с помощью применения мел-фильтров к исходной спектрограмме. Мел-фильтры представляют из себя треугольные функции, равномерно распределенные на мел-шкале. В качестве примера на рисунке 11 изображены 10 мел-фильтров (на практике их берут больше, здесь их мало для наглядности).

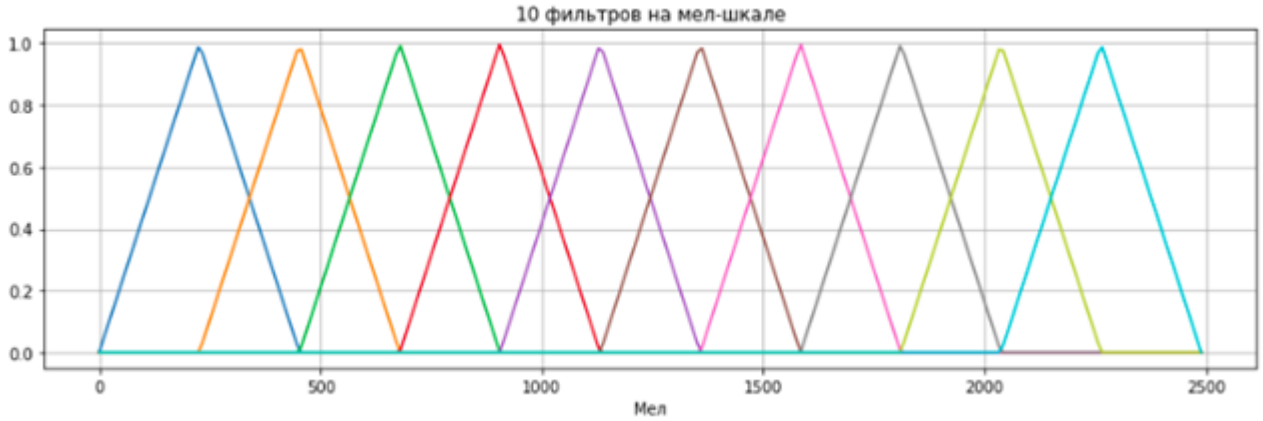


Рисунок 11 — Фильтры в мел-шкале

Перевод фильтров из мел-шкалы в линейную изображён на рисунке 12.

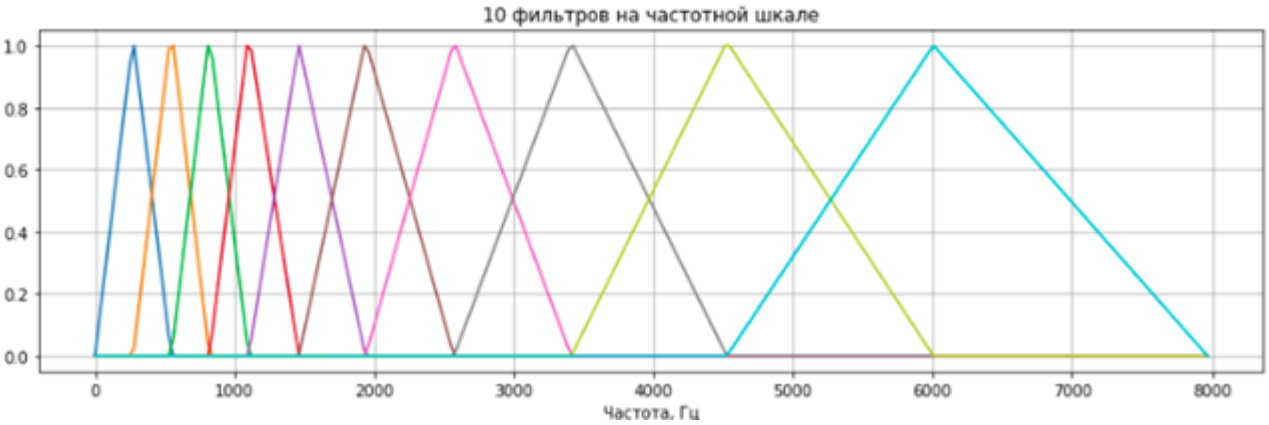


Рисунок 12 — Фильтры в частотной шкале

Каждый столбец исходной спектрограммы скалярно умножается на каждый мел-фильтр (размещенный на частотной шкале), после чего получается вектор чисел, по размеру равный количеству фильтров. В результате таких преобразований значения из низких частот спектрограммы остаются практически неизменными на мел-спектре, а в высоких частотах происходит усреднение значений из более широкого диапазона. В качестве примера на рисунке 13 изображена мел-спектрограмма, построенная по предыдущей спектрограмме с использованием 64 мел-фильтров.

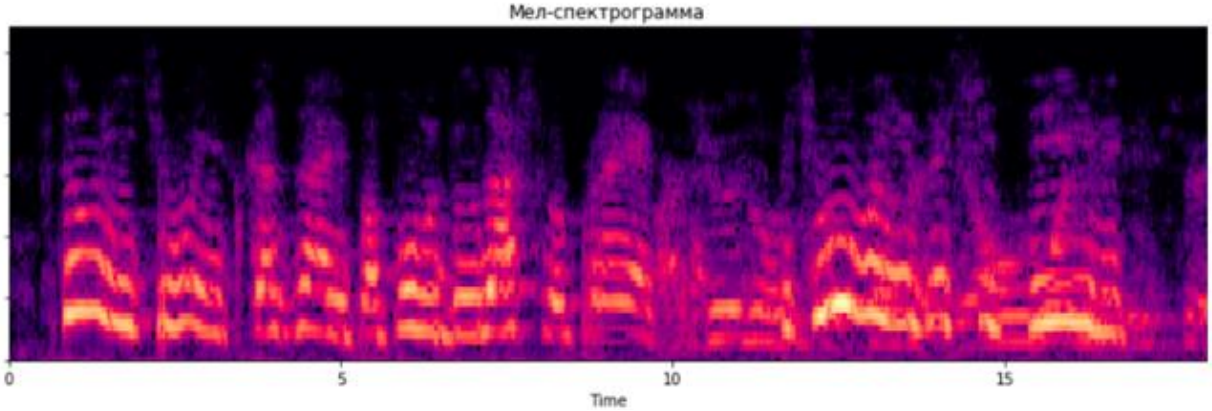


Рисунок 13 — Мел-спектрограмма

Мел-частотные кепстральные коэффициенты (Mel-frequency cepstral coefficients, MFCC) хорошо зарекомендовали себя как в задаче распознавания речи, так и в задаче распознавания дикторов. Успешное применение в обоих задачах одних и тех же акустических признаков указывает на то, что эти акустические признаки содержат в себе достаточно широкий спектр как дикторской, так и лингвистической информации.

Процедуру построения признаков MFCC коротко можно описать следующими шагами:

1. Разбиение входного сигнала на временные окна небольшой длины (20-30 мс), называемые фреймами, с фиксированным шагом смещения (10-15 мс).
2. Вычисление оценки периодограммы (спектральной плотности) спектра мощности.
3. Применение мел-фильтрбанка к спектрам мощности, суммируя энергию в каждом фильтре.
4. Взятие логарифма всех энергий на выходе банка фильтров.
5. Применение дискретного косинусного преобразования (Discrete Cosine Transform, DCT).
6. Сохранение первых 2-13 коэффициентов DCT.

Аудиосигнал постоянно меняется, поэтому для упрощения предполагается, что на коротких временных масштабах аудиосигнал не сильно изменяется (т. е. фрейм — статистически стационарный). Поэтому сигнал делится на фреймы в 20-40 мс. Если фрейм намного короче, у нас недостаточно выборок, чтобы получить надежную спектральную оценку, если он длиннее, сигнал слишком сильно меняется по всему кадру.

Следующим шагом является вычисление спектра мощности каждого фрейма. Это мотивировано человеческой улиткой (органом в ухе), которая вибрирует в разных местах в зависимости от частоты входящих звуков.

Спектральная оценка периодограммы по-прежнему содержит много лишней информации для автоматического распознавания речи (ASR). По этой причине берутся сгустки ячеек периодограммы и суммируются, чтобы получить представление о том, сколько энергии существует в различных частотных областях. Это выполняется нашим мел-фильтрбанком: первый фильтр очень узкий и дает представление о том, сколько энергии существует вблизи 0 Герц. По мере того, как частоты становятся выше, наши фильтры становятся шире, поскольку мы меньше беспокоимся о вариациях.

Как только у нас есть энергии банка фильтров, мы берем их логарифм. Это также обусловлено человеческим слухом: мы не слышим громкость в линейном масштабе. Как правило, чтобы удвоить воспринимаемую громкость звука, нам нужно вложить в него в 8 раз больше энергии. Это означает, что большие колебания энергии могут звучать не так уж по-разному, если звук с самого начала громкий. Почему логарифм, а не кубический корень? Логарифм позволяет нам использовать вычитание кепстрального среднего, которое является методом нормализации канала.

Заключительным шагом является вычисление DCT энергий фильтра. Есть 2 основные причины, по которым это делается. Поскольку все наши банки фильтров перекрываются, энергии банков фильтров вполне коррелируют друг с другом. DCT декоррелирует энергии, что означает, что диагональные ковариационные матрицы могут использоваться для моделирования объектов, например, в классификаторе HMM. Но обратите внимание, что сохраняются только 12 из 26 коэффициентов DCT. Это связано с тем, что более высокие коэффициенты DCT представляют собой быстрые изменения в энергиях банка фильтров, и оказывается, что эти быстрые изменения фактически ухудшают производительность ASR, поэтому мы получаем небольшое улучшение, отбрасывая их.

Формула для вычисления фильтрбанков:

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)}, & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)}, & f(m) \le k \le f(m+1) \\ 0, & k > f(m+1) \end{cases}.$$

Детектор активности речи (VAD)

Интервалы на записи, где голосовая информация присутствует, называют участками активной речи, а паузы между речевыми «всплесками» – участками пауз. Детектор активности речи (англ. Voice activity detector, VAD) – это алгоритм, позволяющий обнаружить голосовую активность во входном акустическом сигнале для отделения активной речи от фонового шума или тишины. Детектор активности речи обычно производит двоичное решение для обрабатываемого речевого фрейма (как правило, длиной 10-20 мс), указывая на наличие или отсутствие речи.

Детекторы речи можно разделить на следующие группы:

1. Простейшие (энергетические, ОТ, ZCR, и др.).
2. Телекоммуникационные (G.729B, Silk, AFE).
3. На основе СГР-моделей (Self Adaptive, GMM-VAD).
4. На основе нейросети (AuroraVAD, bDNN-VAD, RNN-VAD).
5. На основе распознавания фонем (PhnBrno).

В наиболее простой реализации наличие паузы определяется на основе сравнения суммарной энергии речевых данных с некоторым пороговым значением, которое отделяет паузу от речи. В таком случае необходимо подобрать порог так, чтобы не допустить слишком частое устранение ошибочных пауз, что может привести к потере полезных данных.

Задание 1

1. Посмотрите на спектрограмму и попробуйте найти признаки, по которым можно отличить произнесение «yes» от «no».
2. Определите, в каких частотах находится основная энергия этого речевого сигнала.
3. Впишите ответы в специальные поля шаблона.

Задание 2

Нарисуйте спектрограмму в мел-шкале. Для этого используйте формулу Дугласа О'Шонесси [3].

Задание 3

1. Реализуйте функцию вычисления *fbank*.
2. Пройдите тесты, указанные в шаблонах.
3. Визуализируйте полученные фильтрбанки.

Задание 4

1. Реализуйте функцию вычисления *mfcc*.
2. Пройдите тесты, указанные в шаблонах.
3. Визуализируйте *mfcc*.

Задание *5

1. Реализуйте простой VAD.
2. Настройте VAD, чтобы хорошо определялись границы слов.

Задание *6

1. Реализуйте классификатор слов «yes»/«no» с использованием реализованного VAD для разбиения входных файлов на отдельные слова. Добейтесь точности 0.95.
2. Визуализируйте кривые обучения и приведите логи обучения.

Примечание

В задании 3 описания формул для *fbank* можно найти в [4]. Обратите внимание, что реализации базовых функций, таких как вычисления *fbank,* могут отличаться в различных широко используемых библиотеках. Как правило, это не критично для большинства практических задач, но бывают сложности при поиске ошибок и сравнении реализаций. Реализуйте функцию вычисления *fbank* по формулам библиотеки *librosa* по аналогии с открытым исходным кодом. Формулы, используемые в библиотеке *librosa*, отличаются от используемых в лекциях и некоторых других источниках. Обратите внимание на использование *np.diff* и *np.substract*, а также на нормализацию *slaney* [5].

В задании 5 можно отличать паузу от речи по энергии *mfcc* спектра (или части спектра). Чтобы предотвратить скачки на краях слов, воспользуйтесь сверткой со сглаживающим окном, например, Ханна. Для знакомства с методами машинного обучения можете воспользоваться ресурсом [6].

В задании 6 классификацию можно сделать, например, с помощью SVM по усредненным признаки выделеных VAD'ом слов. Или любым другим удобным для вас способом.

Выполнение работы и критерии приемки

В первой лабораторной работе «Сигналы и аудиофайлы» необходимо в заданном шаблоне исходных кодов (Приложение Г) успешно реализовать следующие функции:

1) Спектрограммы сигнала;
2) Вычисления фильтрбанка;
3) Вычисления mfcc;
4) Детектор активности речи;
5) * Классификатор слов «yes»/«no»;

и ответить на 2 вопроса:

1) По каким признакам можно отличить произнесение «yes» от «no»?
2) В каких частотах находится основная энергия исследуемого речевого сигнала?

В шаблоне исходных кодов (Приложение Г) предусмотрены автоматические проверки для всех реализуемых студентом функций. Они помогают понять, все ли сделано правильно. Прохождение автоматических проверок является необходимым условием выполнения работы.

СПИСОК ЛИТЕРАТУРЫ

1. Матвеев Ю.Н., Рыбин С.В. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО КУРСУ «СИНТЕЗ РЕЧИ». Учебно-методическое пособие. СПб: Университет ИТМО, 2017. – 45с.
2. Репозиторий с лабораторными работами [Электронный ресурс]. URL: https://github.com/itmo-mbss-lab/dsp_labs_book
3. Mel scale [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Mel_scale
4. Описания формул для fbank [Электронный ресурс]. URL: https://studbooks.net/2037683/informatika/raschyot_filtrov.
5. Librosa documentation [Электронный ресурс]. URL: https://librosa.org/doc/main/generated/librosa.filters.mel.html
6. Курс «Машинное обучение» от К.В. Воронцова [Электронный ресурс]. URL:https://www.youtube.com/playlist?list=PLJOzdkh8T5krxc4HsHbB8g8f0hu7973fK

ПРИЛОЖЕНИЕ А – коды лабораторной работы 1

```
# Делаем базовые импорты
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# ЗАДАНИЕ 2
# Сначала определим функцию для отрисовки сигнала с хорошим масштабом и
сеткой
# Это поможет легче анализировать сигнал
def draw_signal(data, figsize=(14, 14)):
    plt.figure(figsize=figsize)
    plt.plot(data, linewidth=2)
    plt.minorticks_on()
    plt.xticks(np.arange(0, 1000, step=100))
    plt.yticks(np.arange((data.min().round())//10*10,
                         (data.max().round())//10*10+10, step=5))
    plt.grid(which='major',
        color = 'k',
        linewidth = 1)
    plt.grid(which='minor',
        color = 'k',
        linestyle = ':')
    plt.show()

# Читаем данные с подготовленными сигналами
import pickle
with open("resources/data.pickle", "rb") as f:
    test_data = pickle.load(f)
# Теперь можно приступать к практике!

# Визуализируем эталонный сигнал «a»
draw_signal(test_data['task2']['a'])
```

```
# Инициализация сигнала «a» для задания 2
a =
# Визуализируем инициализированный сигнал
draw_signal(a)
assert len(a) == 1000
assert np.allclose(a, test_data["task2"]["a"], atol=1)
print("Ok!")

# Визуализируем эталонный сигнал «b»
draw_signal(test_data['task2']['b'])
# Инициализация сигнала «b» для задания 2
b =
# Визуализируем инициализированный сигнал
draw_signal(b)
assert len(b)== 1000
assert np.allclose(b, test_data["task2"]["b"], atol=1)
print("Ok!")

# Визуализируем эталонный сигнал «c»
draw_signal(test_data['task2']['c'])
# сигнал состоит из двух гармоник
# Инициализация сигнала «c» для задания 2
c =
# Визуализируем инициализированный сигнал
draw_signal(c)
assert len(c)== 1000
assert np.allclose(c, test_data["task2"]["c"], atol=1)
print("Ok!")

# ЗАДАНИЕ 3

def convolve(in1, in2):
    #Реализуйте операцию свертки для двух сигналов in1 и in2
```

```
def test_convolve(a, b, print_debug=False):
    my_result = convolve(a, b)
    scipy_result = scipy.signal.convolve(a, b, method='direct')
    if print_debug:
        print(f"Your result {my_result}")
        print(f"Scipy result {scipy_result}")
    assert np.allclose(my_result, scipy_result), f"Test {a} conv {b} failed"
    print("Ok!")

a = np.repeat([0,1,0], 10)
b = np.array([0,1,2,3,2,1,0])

test_convolve(a, b, print_debug=False)

# ЗАДАНИЕ 4

def karplus_strong(noise, N):
    # Реализуйте алгоритм Карплуса-Стронга:
    # Noise - input
    # N - number of samples to generate
    # return y - generated signal based on Noise
    raise NotImplementedError()

plt.figure(figsize=(10,5))
plt.xlabel('n', fontsize=14)
plt.ylabel('Амплитуда', fontsize=14)
plt.xlim(0, gen_len)
plt.plot(np.linspace(0, gen_len+1, gen_len), gen_wav)
plt.tick_params(axis='both', which='major', labelsize=14)
plt.grid()
```

ПРИЛОЖЕНИЕ Б – коды лабораторной работы 2

```
# Делаем базовые импорты
import librosa
import numpy as np
import scipy
import scipy.fft
import IPython.display as ipd
import matplotlib.pyplot as plt
import librosa.display
import librosa.filters
import hashlib

# Запретим numpy выполнять деление на 0
np.seterr(divide='raise', invalid='raise')
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

# Определим две функции для отрисовки сигнала.
def draw_waveform(wav, sr, figsize=(14, 5)):
    # Отрисовка звукового сигнала во временной области
    plt.figure(figsize=figsize)
    librosa.display.waveplot(wav, sr=sr)

def draw_sig(frames, name=None, figsize=(14, 3)):
    # Отрисовка сигнала в виде "леденцов на палочке"
    plt.figure(figsize=figsize)
    plt.stem(frames)
    if name is not None:
        plt.legend([name])
    plt.xlabel("n")
```

```
    plt.ylabel("Energy")
    plt.show()

# Создадим несколько простых сигналов. Они нам понадобятся для дальшейшего тестирования
signals = {}

name = "Единичный импульс"
signals[name] = np.array([1] + [0]*39, dtype=float)
draw_sig(signals[name], name)

name = "Единичный скачок"
signals[name] = np.array([1]*40 ,dtype=float)
draw_sig(signals[name], name)

name = "Синус"
signals[name] = np.sin(np.pi * np.arange(40)/10)
draw_sig(signals[name], name)
name = "Шум"
signals[name] = np.random.random(40)-0.5
draw_sig(signals[name], name)

# ЗАДАНИЕ 1

def DFT(x):
    # Реализуйте алгоритм дискретного преобразования Фурье
    raise NotImplementedError()

def test_DTF(x, transform=DFT):
    scipy_result = scipy.fft.fft(x)
    curr_result = transform(x)
    if scipy_result.shape != curr_result.shape:
```

```
        print("TEST_FAILED")
        print(f"Your spectrogram shape is {curr_result.shape}. "
              f"Scipy spectrogram shape is {scipy_result.shape}")
        return -1
    if not np.allclose(curr_result, scipy_result):
        print("TEST FAILED")
        print(f"scipy spectrogram: {scipy_result}")
        print(f"Your DTF spectrogram: {curr_result}")
        print(f"Average diff is {np.mean(
                np.abs(scipy_result - curr_result)
           )}")
        return -2
    print("TEST PASSED")
    return 0

for name, sig in signals.items():
    print(f"Checking '{name}'")
    assert test_DTF(sig) == 0, "Check you implementation"
print("All ok!")

def FFT(x):
    # Реализуйте алгоритм быстрого преобразования Фурье
    raise NotImplementedError()

for name, sig in signals.items():
    print(f"Checking '{name}'")
    assert test_DTF(sig, transform=FFT) == 0, "Check you implementation"
print("All ok!")

# ЗАДАНИЕ 2

#функция построения амплитудного спектра
```

```
def get_amplitude_from_spec(spec):
    # Реализуйте функцию построения амплитудного спектра
    raise NotImplementedError()

#функция построения фазового спектра
def get_phi_from_spec(spec):
    # Реализуйте функцию построения фазового спектра
    raise NotImplementedError()



def check_mean_var(sig, mean=None, var=None):
    if mean is not None and np.mean(sig) != mean:
        print(f"Bad mean. Expected {mean}, got {np.mean(sig)}")
        return False
    if var is not None and np.var(sig) != var:
        print(f"Bad var. Expected {var}, got {np.var(sig)}")
        return False
    return True

_spec = scipy.fft.fft(np.array([1]+[0]*10))
assert check_mean_var(get_amplitude_from_spec(_spec), 1.0, 0.0), \
    "Wrong Amplitude"
assert check_mean_var(get_phi_from_spec(_spec), 0.0, 0.0), \
    "Wrong Phase"

_spec = scipy.fft.fft(np.array([1]*10))
assert check_mean_var(get_amplitude_from_spec(_spec), 1.0, 9.0), \
    "Wrong  Amplitude"
assert get_amplitude_from_spec(_spec)[0] == 10, \
    "Wrong Amplitude"
assert get_phi_from_spec(_spec)[0] == 0, \
    "Wrong phase"
```

```
_spec = scipy.fft.fft(scipy.fft.ifft(np.array([0] + [10+5j]+[0]*6 + [10-5j])))
assert get_amplitude_from_spec(_spec).round(1)[1] == 11.2 , \
    "Wrong Amplitude. Make sure it is a complex number module."
assert get_phi_from_spec(_spec).round(1)[1] == 0.5 , \
    "Wrong Amplitude. Make sure it is an angle."

print("All OK!")

def draw_spec(spec, name=None, draw_A=True, draw_p=True, figsize=(14, 3)):
    if len(spec)<100:
        # Рисуем точки как "леденцы на палочках"
        draw_func = plt.stem
    else:
        # При большом N "леденцы" выглядят плохо,
        # Поэтому будем рисовать огибающую функцию
        draw_func = plt.plot

    if draw_A:
        plt.figure(figsize=figsize)
        plt.title("Amplitude spectrum")
        spec_A = get_amplitude_from_spec(spec)
        draw_func(spec_A)
        plt.ylabel("Magnitude")
        plt.xlabel("n")
        if name is not None:
            plt.legend([name])
        plt.show()

    if draw_p:
        plt.figure(figsize=figsize)
        plt.title("Phase spectrum")
```

```
        phi = get_phi_from_spec(spec)
        draw_func(phi)
        plt.ylabel("Radian")
        plt.xlabel("n")
        if name is not None:
            plt.legend([name])
        plt.show()
    return


# ЗАДАНИЕ 3

def STFT(x, n_fft=2048, hop_length=512, window='hann'):
    # Реализуйте алгоритм оконного преобразования фурье
    # x - signal
    # n_fft - fft window size
    # hop_length - step size between ffts
    # window - window type. See scipy.signal.get_window
    # return spectrogram
    raise NotImplementedError()

def test_stft(x, n_fft=2048, hop_length=512, window='hann'):
    librosa_result = librosa.stft(x, n_fft=n_fft, hop_length=hop_length, \
window='hann', center=True)
    result = STFT(x, n_fft=n_fft, hop_length=hop_length, window='hann')
    if librosa_result.shape != result.shape:
        print(f"Your shape {result.shape} != librosa stft shape { \
librosa_result.shape} ")
        return -1
    if not np.allclose(librosa_result, result):
        print(f"Wrong results. Diff {np.abs(librosa_result-result)}")
        return -2
```

```
    return 0

for name, sig in signals.items():
    print(f"Checking '{name}'")
    for n_fft in (8, 4):
        for hop_length in (n_fft//4, n_fft//2):
            assert test_stft(sig, n_fft=n_fft, hop_length=hop_length)==0, \
                f"Test failed. Params n_fft {n_fft}. Hop_length {hop_length}"
    print("ok")
print("All ok!")
# Определим функцию для отрисовки
def draw_stft(X, sr, figsize=(14, 5)):
    plt.figure(figsize=figsize)
    # X - комплекстная спектрограмма
    # Для получения энергии достаточно взять модуль
    Xdb = librosa.amplitude_to_db(abs(X))
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
    plt.show()

# Загрузим аудио файл с паровозным гудком
train_whistle, tw_sr = librosa.core.load("resources/train_whistle.wav")
print(f"Len is {len(train_whistle)} ( ). Sample rate is {tw_sr}")
player = ipd.Audio(train_whistle, rate=tw_sr)
ipd.display(player)
draw_waveform(train_whistle, sr=tw_sr)

train_whistle_spec = scipy.fft.fft(train_whistle)
print("Гудок паровоза в частотной области")
draw_spec(train_whistle_spec, draw_p=False, figsize=(15,5))

# ЗАДАНИЕ 4
```

```
# task_answer = [ freq1, freq2, freq3 ]
task_answer =

assert type(task_answer) == list or type(task_answer) == np.ndarray, \
        f"Wrong answer type. Expected list or numpy array. Got { \
type(task_answer)}"
assert len(task_answer) == 3, \
        f"Wrong len {len(task_answer)}."
assert (round(sum(task_answer)) == 1677 and \
            round(np.var(task_answer)) == 1407) or \
        (abs(sum(task_answer)  - 1836) <= 3 and \
            abs(np.var(task_answer) - 10153) <= 181), f"Wrong freqs"
print("Correct!")

# ЗАДАНИЕ 5

#dtmf.wav
dtmf, dtmf_sr = librosa.core.load("resources/dtmf.wav")
print(f"Len is {len(dtmf)}. Sample rate is {dtmf_sr}")
player = ipd.Audio(dtmf, rate=dtmf_sr)
ipd.display(player)
draw_waveform(dtmf, sr=dtmf_sr)

# phone_number = [first: int, second: int, third: int]

assert type(phone_number) == list or type(phone_number) == np.ndarray, \
        f"Wrong answer type. Expected list or numpy array. Got { \
type(phone_number)}"
assert len(phone_number) == 3, \
        f"Wrong len {len(phone_number)}."
assert round(np.mean(phone_number)) == 5 and \
            round(np.var(phone_number)) == 11 , f"Wrong number"
```

```
assert hashlib.md5(''.join(map(str, phone_number)).encode()).hexdigest() == \
"140f6969d5213fd0ece03148e62e461e", \
        f"Wrong hash. How did you pass mean/var test?"
print("Correct!")
```

ПРИЛОЖЕНИЕ В – коды лабораторной работы 3

```
# Делаем базовые импорты
import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.fft
import librosa
import IPython.display as ipd
import librosa.display

# Запретим numpy выполнять деление на 0
np.seterr(divide='raise', invalid='raise')
%matplotlib inline

# Из ЛР2
def get_magnitude_from_spec(spec):
    # Реализуйте функцию построения амплитудного спектра
    raise NotImplementedError()

def get_phi_from_spec(spec):
    # Реализуйте функцию построения фазового спектра
    raise NotImplementedError()

# Функция отрисовки аудио сигнала.
```

```
def draw_waveform(wav, sr, figsize=(14, 5)):
    # Отрисовка звукового сигнала во временной области
    plt.figure(figsize=figsize)
    librosa.display.waveplot(wav, sr=sr)
    plt.show()

# Функция отрисовки спектра
def draw_spec(spec, name=None, draw_A=True, draw_p=True, figsize=(14, 3)):
    if len(spec)<100:
        # Рисуем точки как "леденцы на палочках"
        draw_func = plt.stem
    else:
        # При большом N "леденцы" выглядят плохо,
        # Поэтому будем рисовать огибающую функцию
        draw_func = plt.plot

    if draw_A:
        plt.figure(figsize=figsize)
        plt.title("Magnitude spectrum")
        spec_A = get_magnitude_from_spec(spec)
        draw_func(spec_A)
        plt.\* ylabel("Magnitude")
        plt.xlabel("n")
        if name is not None:
            plt.legend([name])
        plt.show()

    if draw_p:
        plt.figure(figsize=figsize)
        plt.title("Phase spectrum")
        phi = get_phi_from_spec(spec)
        draw_func(phi)
```

```
        plt.ylabel("Radian")
        plt.xlabel("n")
        if name is not None:
            plt.legend([name])
        plt.show()
    return

# ЗАДАНИЕ 1

# Определяем фильтры
def FIR_filter(x, alpha_prev=0.5, alpha_curr=0.5):
    # Реализуйте КИХ-фильтр
    # alpha_prev - weight for previous frame
    # alpha_curr - weight for current frame
    # Реализуйте КИХ-фильтр
    raise NotImplementedError()

def IIR_filter(x, alpha=0.5):
    # Реализуйте БИХ-фильтр
    # alpha - weight for reccurent connection
    raise NotImplementedError()

def test_filters():
    x = np.ones(10)
    y = FIR_filter(x)
    assert y[0] == 0.5 and (y[1:] == 1).all(), \
        RuntimeError(f"bad FIR. x={x}. y={y}")
    y = FIR_filter(x, alpha_prev=0.1, alpha_curr=0.9)
    assert y[0] == 0.9 and (y[1:] == 1).all(), \
        RuntimeError(f"bad FIR(alphas 0.1 and 0.9). x={x}. y={y}")
    y = IIR_filter(x)
    assert y[0]==1 and (y[3:].round() == 2).all() and (y<2).all(), \
```

```
        RuntimeError(f"Bad IIR. x={x}. y={y}")
    y = IIR_filter(x, 0.2)
    assert y[0]==1 and (y[3:].round(2) ==1.25).all() and (y<1.25).all(), \
        RuntimeError(f"Bad IIR(alpha={0.2}. x={x}. y={y}")
    print("All Ok!")

test_filters()

# ЗАДАНИЕ 2

# Определите единичный импульс длиной 20 отсчетов (переменная impulse).
impulse =

print("Импульсная характеристика фильтров с aplha=0.5:")

# Получите импульсную характеристику фильтров (переменные FIR_filter и
IIR_filter)
fir_impulse_response =
iir_impulse_response =

# Построим графики
plt.figure(figsize=(8,2))
plt.title("КИХ-фильтр:")
plt.plot(impulse, 'o-')
plt.plot(fir_impulse_response, '.-')
plt.legend(['импульс', 'фильтр'])
plt.axis([-0.1, len(impulse), -0.1, 1.2])
plt.show()

plt.figure(figsize=(8,2))
plt.title("БИХ-фильтр:")
plt.plot(impulse, 'o-')
```

```
plt.plot(iir_impulse_response, '.-')
plt.legend(['импульс', 'фильтр'])
plt.axis([-0.1, len(impulse), -0.1, 1.2])
plt.show()

assert impulse.shape[0] == 20, "Bad impulse shape"
assert (fir_impulse_response[0:2] == 0.5).all() and \
        (fir_impulse_response[2:] == 0).all(), "Bad FIR."
assert iir_impulse_response.sum().round() == 2 and \
        iir_impulse_response.sum() < 2 and \
        (iir_impulse_response != 0).all(), "Bad IIR."
assert iir_impulse_response[1:].sum().round() == 1 and \
        iir_impulse_response[1:].sum()<1 and \
        iir_impulse_response[2:].sum() < 0.5, "Bad IIR."
print("All ok!")

# ЗАДАНИЕ 3

#Определите единичный скачок длиной 20 отсчетов (переменная step)
step =

print("Переходная характеристика фильтров с alpha=0.5:")

# Получите переходную характеристику фильтров
fir_step_response =
iir_step_response =

# Построим графики
plt.figure(figsize=(8,2))
plt.title("КИХ-фильтр:")
plt.plot(step, 'o-')
plt.plot(fir_step_response, '.-')
```

```
plt.axis([-0.1, len(step), 0, 1.2])
plt.legend(['скачок', 'фильтр'])

plt.show()

plt.figure(figsize=(8,2))
plt.title("БИХ-фильтр:")
plt.plot(step, 'o-')
plt.plot(iir_step_response, '.-')
plt.axis([-0.1, len(step), 0, 2.2])
plt.legend(['скачок', 'фильтр'])
plt.show()

assert step.shape[0] == 20, "Bad step shape"
assert fir_step_response[0] == 0.5 and \
        (fir_step_response[1:] == 1).all(), "Bad FIR."
assert iir_step_response[0] == 1 and iir_step_response[1] == 1.5 and iir_step_
response[2] == 1.75 and \
        iir_step_response.mean().round() == 2 and \
            (iir_step_response < 2).all(), "Bad IIR."
print("All ok!")

# ЗАДАНИЕ 4

print("Амплитудно-частотная характеристика фильтров с alpha=0.5")

# Получить амплитудно-частотную характеристику для единичного импульса
(переменная impulse)
fir_frequency_response =
iir_frequency_response =

# Построим графики
```

```
plt.figure(figsize=(6,2))
plt.title("КИХ-фильтр:")
plt.plot(fir_frequency_response, '.-')
plt.show()

plt.figure(figsize=(6,2))
plt.title("БИХ-фильтр:")
plt.plot(iir_frequency_response, '.-')
plt.show()

assert fir_frequency_response.shape[0] == \
iir_frequency_response.shape[0] == 10, f"Bad FR shape. Must be N//2."
_ideal_fir_fr = np.array([1., 0.98768834, 0.95105652, 0.89100652, 0.80901699,
       0.70710678, 0.58778525, 0.4539905 , 0.30901699, 0.15643447])
assert np.allclose(fir_frequency_response, _ideal_fir_fr), \
    f"Bad fir FR. diff is { \
           np.abs(fir_frequency_response - _ideal_fir_fr).sum() \
      }"
_ideal_iir_fr = np.aray([1.99999809, 1.82896351, 1.50587408, 1.22885364, \
       1.03088138, 0.89442634, 0.80089238, 0.73765316, 0.69689865, 0.67403739]
)
assert np.allclose(iir_frequency_response, _ideal_iir_fr), \
    f"Bad iir FR. diff is { \
           np.abs(iir_frequency_response - _ideal_iir_fr).sum()
      }"

print("All ok!")

# ЗАДАНИЕ 5

print("Фазово-частотная характеристика")
```

```
# Получите фазово-частотную характеристику фильтров для единичного импульса
(переменная impulse).

fir_phase_response =
iir_phase_response =

# Построим графики
plt.figure(figsize=(6,2))
plt.title("КИХ-фильтр:")
plt.plot(fir_phase_response, '.-')
plt.show()

plt.figure(figsize=(6,2))
plt.title("БИХ-фильтр:")
plt.plot(iir_phase_response, '.-')
plt.show()

assert fir_phase_response.shape[0] == iir_phase_response.shape[0] == 10, \
    f"Bad PR shape. Must be N//2."
_ideal_fir_pr = np.array([-0., -0.15707963, -0.31415927, -0.4712389, \
-0.62831853, -0.78539816, -0.9424778, -1.09955743, -1.25663706, -1.41371669])
assert np.allclose(fir_phase_response, _ideal_fir_pr), \
    f"Bad fir PR. diff is {np.abs(fir_phase_response - _ideal_fir_pr).sum()}"
_ideal_iir_pr = np.array([-0., -0.28649379, -0.45845783, -0.52023287, \
-0.51233491, -0.46364761, -0.39071251, -0.30300249, -0.20627323, \
-0.10433379])
assert np.allclose(iir_phase_response, _ideal_iir_pr), \
    f"Bad iir PR. diff is {np.abs(iir_phase_response - _ideal_iir_pr).sum()}"

print("All ok!")
```

ПРИЛОЖЕНИЕ Г – коды лабораторной работы 4

```
# Делаем базовые импорты

import librosa

import numpy as np

import scipy

import scipy.fft

import IPython.display as ipd

import matplotlib.pyplot as plt

import librosa.display

import librosa.filters

import hashlib

from glob import glob

import os

import sklearn


%matplotlib inline


import warnings

warnings.filterwarnings('ignore')


def draw_waveform(wav, sr, figsize=(14, 5)):

    # Отрисовка звукового сигнала во временной области

    plt.figure(figsize=figsize)

    librosa.display.waveplot(wav, sr=sr)

    plt.show()


# Скачаем датасет yes/no, необходимый для выполнения задания. Про датасет
можно почитать тут https://www.openslr.org/1/

![ ! -f  waves_yesno.tar.gz ] &&

wget https://www.openslr.org/resources/1/waves_yesno.tar.gz
```

```
# И распакуем
!tar -xvzf waves_yesno.tar.gz

# Загрузим и послушаем один из файлов
wav, sr = librosa.load("waves_yesno/0_1_0_1_1_1_0_0.wav")
draw_waveform(wav, sr)
ipd.Audio(wav, rate=sr)

# Построим спектрограмму загруженной вавки
stft = librosa.stft(wav)
stft_db = librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(15,10))
librosa.display.specshow(stft_db, sr=sr, x_axis='time', y_axis='hz')

# ЗАДАНИЕ 1

def mel(spec):
    # Реализуйте построение спектрограммы в мел-шкале
    #spec - stft spectrogram
    raise NotImplementedError()
def test_mel():
    x = np.random.randint(100, size=(1000, 100))
    x_mel = mel(x)
    x_hz = 700.0 * (10.0 ** (x_mel / 2595.0) - 1.0)
    assert np.allclose(x, x_hz), "TEST Hertz -> Mel -> Hertz  failed. "

test_mel()

# ЗАДАНИЕ 2

def mel_filters_pure(sr, n_fft, n_mels):
    # функция построения треугольных мел-фильтров в герц-шкале
```

```
    # sr - sample rate
    # n_fft - length of the FFT window
    # n_mels - number of filters
    # return mel filters matrix. [n_mel, n_fft]
    raise NotImplementedError
```

Обратите внимание что реализации базовых функций таких как вычисления fbank могут отличаться в различных широко используемых библиотеках. Как правило это не критично для большинства практических задач, но бывают сложности при поиске ошибок и сравнении реализаций.

Реализуйте функцию вычисления fbank по формулам библиотеки librosa по аналогии с открытым исходным кодом. Формулы, используемые в библиотеке librosa, отличаются от используемых в лекциях и некоторых других источниках. Обратите внимание на использование np.diff и np.substract, а также на нормализацию slaney. https://librosa.org/doc/latest/ modules/librosa/filters.html#mel

```
def mel_filters_librosa(sr, n_fft, n_mels):
    # функция построения треугольных мел-фильтров в герц-шкале. Реализуйте функцию вычисления fbank по формулам библиотеки librosa. Функции, используемые в библиотеке librosa, отличаются от используемых в ЦОС функций вычисления мел-фильтров.
    # sr - sample rate
    # n_fft - length of the FFT window
    # n_mels - number of filters
    # return mel filters matrix. [n_mel, n_fft]
    raise NotImplementedError

assert mel_filters_librosa(32, 46, 4).shape == (4, 24) and \
    mel_filters_librosa(65, 45, 5).shape == (5, 23), "Wrong shape"
assert np.allclose(
     mel_filters_librosa(16, 8, 4),
     librosa.filters.mel(16, 8, n_mels=4, htk=True)
     )
assert np.allclose(
     mel_filters_librosa(8600, 512, 40),
     librosa.filters.mel(8600, 512, n_mels=40, htk=True)
```

```
    )
print("All ok!")


def get_fbanks(
    wav: np.ndarray,
    sr: int,
    window_ms=25,
    step_mc=10,
    n_fbanks=40
):
    # Реализуйте функцию построения фильтрбанка
    # wav - input signal
    # sr - sample rate
    # window_ms - window length in milliseconds
    # step_ms - stft step in milliseconds
    # n_fbanks - number of filters
    # return fbank matrix [n_fbanks, time]
    raise NotImplementedError


def test_fbank(wav, sr, window_ms=25, step_mc=10, n_fbanks=40):
    n_fft = window_ms * sr//1000
    hop_length = step_mc * sr//1000
    fbanks_lib = librosa.feature.melspectrogram(wav, sr, n_fft=n_fft, hop_leng
th=hop_length, n_mels=n_fbanks, htk=True)
    fbanks = get_fbanks(wav, sr, window_ms=window_ms, step_mc=step_mc, n_fbank
s=n_fbanks)


    if fbanks_lib.shape != fbanks.shape:
        print("TEST FAILED")
        print(f"Shape {fbanks_lib.shape} != {fbanks.shape}")


    if not np.allclose(fbanks_lib, fbanks):
        print('TEST PASSED BUT WITH CALCULATION ERROR')
```

```
            print(f"Average diff is {np.mean(np.abs(fbanks_lib - fbanks))}")
            return -1

    print("TEST PASSED")
    return 0

assert test_fbank(wav[:sr*1], sr) <= 0.0003, "1 sec wav test failed"
assert test_fbank(wav, sr) <= 0.0003, "All wav test failed"
print("All ok!")

window_ms = 25
step_mc = 10
n_fbanks = 40
n_fft = window_ms * sr//1000
hop_length = step_mc * sr//1000

# Визуализируем полученные фильтрбанки самописными функциями и функциями
либросы
fbanks = get_fbanks(wav, sr)
plt.figure(figsize=(15,10))
librosa.display.specshow(fbanks, sr=sr, x_axis='time')
plt.ylabel("Filter number")
plt.show()
fbanks_lib = librosa.feature.melspectrogram(wav, sr, n_fft=n_fft, hop_length=h
op_length, n_mels=n_fbanks, htk=True)
plt.figure(figsize=(10,5))
librosa.display.specshow(librosa.power_to_db(fbanks_lib), sr=sr, x_axis='time'
)
plt.ylabel("Filter number")
plt.show()

# ЗАДАНИЕ 4
```

```
def get_mfcc(wav: np.ndarray, sr: int, window_ms=25, step_mc=10, n_mfcc=13):
    # Реализовать вычисление mfcc
    # wav - input signal
    # sr - sample rate
    # window_ms - window length in milliseconds
    # step_ms - stft step in milliseconds
    # n_mfcc - number of filters
    # return mfcc matrix [n_mfcc, time]
    raise NotImplementedError()


def test_mfcc(wav, sr, window_ms=25, step_mc=10, n_mfcc=13):
    n_fft = window_ms * sr//1000
    hop_length = step_mc * sr//1000
    mfcc_lib = librosa.feature.mfcc(
                wav,
                sr,
                n_fft=n_fft,
                hop_length=hop_length,
                n_mfcc=n_mfcc,
                htk=True
                )
    mfcc = get_mfcc(
                wav,
                sr,
                window_ms=window_ms,
                step_mc=step_mc,
                n_mfcc=n_mfcc
                )

    if mfcc_lib.shape != mfcc.shape:
        print("TEST PASSED BUT WITH CALCULATION ERROR")
```

```
        print(f"Shape {mfcc_lib.shape} != {mfcc.shape}")
    if not np.allclose(mfcc_lib, mfcc):
        print("TEST PASSED BUT WITH CALCULATION ERROR")
        print(f"Average diff is {np.mean(np.abs(mfcc_lib - mfcc))}")
        return -1
    print("TEST PASSED")
    return 0
assert test_mfcc(wav[:sr*1], sr) <= 0.0005, "1 sec wav test failed"
assert test_mfcc(wav, sr) <= 0.0005, "All wav test failed"
print("All ok!")

mfcc = get_mfcc(wav, sr)
plt.figure(figsize=(15,10))
librosa.display.specshow(mfcc, sr=sr, x_axis='time')
plt.ylabel("Filter number")
plt.show()

# КЛАССИФИКАЦИЯ СЛОВ

# Загрузим весь датасет
def load_yn_dataset(directory):
    X, labels = [], []
    bad_files = set(["0_1_0_1_0_0_0_0"])
    for f in glob(directory +"/*.wav"):
        name = os.path.basename(f)[:-4]
        if name in bad_files:
            continue
        y = [int(l) for l in name.split("_")]
        x, _ = librosa.load(f)
        X.append(x)
        labels.append(y)
```

```
    return X, labels

X, Y = load_yn_dataset("waves_yesno/")

# Послушаем одну вавку
wav, sr = librosa.load("waves_yesno/0_1_0_1_0_0_0_0.wav")
ipd.Audio(wav, rate=sr)


# Отделим 20% для теста
X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(
                                        X,
                                        Y,
                                        test_size=0.2,
                                        random_state=1
                                        )

# ЗАДАНИЕ 5

# Реализуйте VAD

# train_VA = # List[np.ndarray]. 1 - Voice, 0 - silence
# test_VA = # List[np.ndarray]. 1 - Voice, 0 - silence

def test_VAD(VOICE, VA, Y, min_acc=0.95):
    def check_ali(ali, num_words):
        diff = ali[1:]  - ali[:-1]
        if diff.sum()!=0:
            print("VAD detected speech at the beginning (or end) of audio")
            return -1
        if not (diff > 0).sum() == (diff > 0).sum() == num_words:
            return -2
```

```
        return 0


    bad_i = []
    err =0
    for i, (ali, y) in enumerate(zip(VA, Y)):
        assert len(VOICE[i]) == len(ali), "shapes doesn't match"
        if check_ali(ali, len(y)) != 0:
            bad_i.append(i)
            err+=1
    acc = (len(Y) - err)/len(Y)
    print(f"Accuracy {acc}")
    if len(bad_i) >0:
        print(f"ids with error {bad_i}")
    assert acc >= min_acc , f"Accuracy must be >= {min_acc}"
    print("Test passed")

test_VAD(X_train, train_VA, Y_train)
test_VAD(X_test, test_VA, Y_test)
print("VAD ok!")
```

Мещеряков Илья Денисович
Волохов Владимир Андреевич
Шуранов Евгений Витальевич
Махныткина Олеся Владимировна

Методические указания к выполнению лабораторных работ по курсу «Цифровая обработка сигналов»

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А