



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Самарский государственный технический университет»
(ФГБОУ ВО «СамГТУ»)

Кафедра «Информационно-измерительная техника»

А.В. БОЧКАРЕВ
Р.Т. САЙФУЛЛИН

ПРЕОБРАЗОВАНИЕ ИЗМЕРИТЕЛЬНЫХ СИГНАЛОВ

Практикум

Самара

Самарский государственный технический университет
2022

**Печатается по решению редакционно-издательского совета СамГТУ,
(протокол №12 от 23.05.2022 г.)**

УДК 006.9

К 64

Бочкарев А.В.

**К 64 Преобразование измерительных сигналов: практикум /
A.B. Бочкарев, Р.Т. Сайфуллин. – Самара: Самар. гос. техн. ун-т, 2022. – 150 с.**

Пособие содержит материалы для выполнения практических работ по дисциплине «Преобразование измерительных сигналов» и может быть использовано как в очном, так и дистанционном формате обучения. Все используемое студентами программное обеспечение бесплатное или имеет бесплатные аналоги (в случае с Excel).

Предназначено для студентов бакалавриата высших технических учебных заведений специальности 12.03.01.

**Рецензенты: д-р техн. наук П.К. Ланге,
канд. техн. наук И.Ю. Занозин**

© А.В. Бочкарев, Р.Т. Сайфуллин, 2022

© Самарский государственный
технический университет, 2022

ВВЕДЕНИЕ

Данный практикум содержит материалы для выполнения практических работ по дисциплине «Преобразование измерительных сигналов» и **может быть использовано как в очном, так и дистанционном формате обучения**, поскольку все используемое студентами программное обеспечение бесплатное или имеет бесплатные аналоги (в случае с Excel).

В рамках пособия рассматриваются наиболее важные по мнению авторов разделы курса «Преобразование измерительных сигналов», среди которых спектральный и корреляционный анализ сигналов, преобразование Фурье и вейвлет-преобразование, основы цифровой фильтрации, свертка и деконволюция. Работы построены таким образом, чтобы приобретаемые навыки позволяли в дальнейшем студентам самостоятельно анализировать измерительные сигналы с применением рассмотренных в настоящем пособии преобразований, наиболее важные моменты подчеркнуты, а сложные для усвоения аспекты опущены или существенно упрощены.

Пособие содержит 16 работ, 6 из которых выполняются вручную, 10 – с применением бесплатной системы компьютерной алгебры, разрабатываемой в России, Smath Studio. Для некоторых работ за компьютером также потребуется Excel или любой иной табличный процессор. Предназначено для студентов бакалавриата высших технических учебных заведений специальности 12.03.01. Данное пособие предназначено для курса практических занятий до 34 академических часов.

ОБЩИЕ ПОЛОЖЕНИЯ ВЫПОЛНЕНИЯ РАБОТ В РАМКАХ ЗАНЯТИЙ

Каждая тема состоит из одного или нескольких заданий. В ходе выполнения работы поощряется использование лекционного материала. Допускается в ходе работы также пользование сторонними источниками. По каждой теме необходим 1 отчет.

Существуют работы, требующие вычислений на ПК, они помечены символом  на первой странице работы, а также те, что выполняются вручную, они помечены символом .

Отчеты по работам  выполняются исключительно в электронном текстовом редакторе, выполненные от руки отчеты не рассматриваются. Такой отчет состоит из следующих структурных элементов:

- титульный лист (с указанием номера темы и его названием), шаблон в данном разделе ниже;
- подробное решение всех заданий в виде скриншотов из Smath Studio, Excel и т.п. с указанием соответствующих номеру варианта заданных условий для каждого задания.

Отчеты  могут выполняться как в печатном варианте на листах А4, так и в ручном на листах любого формата. Форму отчета выбирает студент, от формы отчета не зависит оценка преподавателем результатов работы. Такой отчет состоит из следующих структурных элементов:

- титульный лист (с указанием номера темы и его названием), шаблон в данном разделе ниже (если работаете от руки – логотип СамГТУ исключается, текст справа от него пишете по центру);
- подробное пронумерованное решение задания (без записи условия) с указанием номера варианта и соответствующих ему заданных условий для каждого задания.

При работе в дистанционном формате все отчеты в электронном

виде выгружаются в соответствующую систему университета либо передаются преподавателю иным образом (по предварительной с ним договоренности). Вручную выполненные  отчеты при работе в дистанционном формате сканируются/фотографируются и объединяются в один документ PDF, размер которого не должен превышать 10 МБ. Для фотографирования со смартфона сразу в PDF можно воспользоваться freeware утилитой Mobile Doc Scan или ее аналогами, но результат занимает, как правило, существенно более 10 МБ, для сжатия также ищите ПО либо онлайн-сервисы, коих предостаточно. При возникновении проблем свяжитесь с преподавателем!



Самарский государственный
технический университет
Кафедра Информационно-
измерительная техника

Практическая работа №_
«Название работы»
Вариант №_

Выполнил:
Студент 4-ИАИТ-5
Фамилия И.О.
Проверил:
Бочкарев А.В.

Самара. 20__ г.

РАБОТА №1. АВТОКОРРЕЛЯЦИОННАЯ И ВЗАИМНОКОРРЕЛЯЦИОННАЯ ФУНКЦИИ

Цель работы: закрепление навыков вычисления АКФ и ВКФ на примере простейших дискретных сигналов.

Планируемая продолжительность: 2 академических часа.

Тип работы:  без использования компьютерных средств.

Теоретические основы

АКФ дискретного сигнала, состоящего из N отсчетов может быть найдена с применением следующего выражения:

$$AC_m = \sum_{k=1}^{N-m} S_k \cdot S_{k-m}, \quad -(N-1) \leq m \leq N-1, \quad (4.1)$$

где m – сдвиг между массивами (а также номер элемента в ВКФ);

k – номер элемента в первом массиве.

То есть каждый n -й элемент АКФ находится суммированием произведений отсчетов сигнала на отсчеты его сдвинутой на n отсчетов копии. Данную процедуру можно пошагово проиллюстрировать следующим образом для простейшего дискретного сигнала, состоящего из трех отсчетов.

Таблица 4.1

Пример вычисления АКФ

№ точки	m	S=[1, 2, 3] N=3 -2≤m≤2
1	-2	$AC_{-2} = S_1 \cdot S_3 + S_2 \cdot \underline{S_4} + S_3 \cdot \underline{S_5} = S_1 \cdot S_3 = 3$
2	-1	$AC_{-1} = S_1 \cdot S_2 + S_2 \cdot S_3 + S_3 \cdot \underline{S_4} = S_1 \cdot S_2 + S_2 \cdot S_3 = 8$
3	0	$AC_0 = S_1 \cdot S_1 + S_2 \cdot S_2 + S_3 \cdot S_3 = 14$
4	1	$AC_1 = S_1 \cdot \underline{S_0} + S_2 \cdot S_1 + S_3 \cdot S_2 = S_2 \cdot S_1 + S_3 \cdot S_2 = 8$
5	2	$AC_2 = S_1 \cdot \underline{S_{-1}} + S_2 \cdot \underline{S_0} + S_3 \cdot S_1 = S_3 \cdot S_1 = 3$

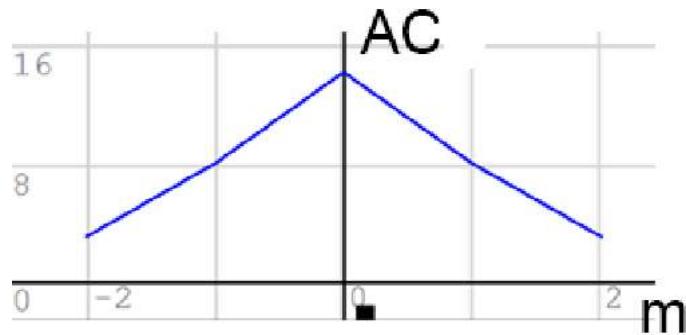


Рис. 4.1. Пример графика АКФ

В таблице выделенные элементы не существуют и поэтому приравниваются нулю (из-за чего члены, умножаемые на них, также равны нулю). Также очевидно, что АКФ симметрична относительно оси ординат.

ВКФ двух сигналов можно найти с помощью выражения (4.1), если в нем в качестве сдвинутого сигнала использовать другой сигнал:

$$CC_m = \sum_{k=1}^{N-m} S1_k \cdot S2_{k-m}, \quad -(N-1) \leq m \leq N-1. \quad (4.2)$$

Процедуру нахождения ВКФ можно проиллюстрировать тем же примером выше, что и АКФ, если сменить в нем сдвигаемый сигнал на другой, не равный исходному.

Таблица 4.2
Пример вычисления ВКФ

№ точки	m	S1=[1, 2, 3]	S2=[11, 1, 22, 3, 3]	N=3	-2≤m≤2
1	-2		CC ₋₂ = S1 ₁ · S2 ₃ + S1 ₂ · <u>S2₄</u> + S1 ₃ · <u>S2₅</u> = S1 ₁ · S2 ₃ =3,3		
2	-1		CC ₋₁ = S1 ₁ · S2 ₂ + S1 ₂ · S2 ₃ + S1 ₃ · <u>S2₄</u> = S1 ₁ · S2 ₂ + S1 ₂ · S2 ₃ =28,6		
3	0		CC ₀ = S1 ₁ · S2 ₁ + S1 ₂ · S3 ₂ + S1 ₃ · S2 ₃ = 140		
4	1		CC ₁ = S1 ₁ · <u>S2₀</u> + S1 ₂ · S2 ₁ + S1 ₃ · S2 ₂ = S1 ₂ · S2 ₁ + S1 ₃ · S2 ₂ =80		
5	2		CC ₂ = S1 ₁ · <u>S2₋₁</u> + S1 ₂ · <u>S2₀</u> + S1 ₃ · S2 ₁ = S1 ₃ · S2 ₁ =30		

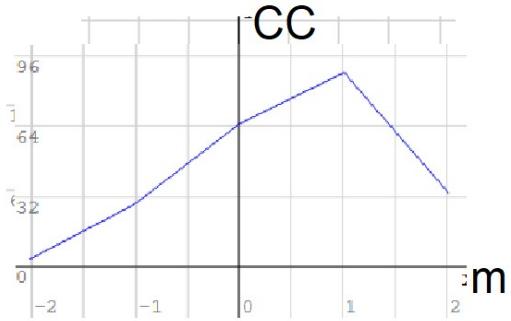


Рис. 4.3. Пример графика ВКФ

ВКФ может быть как симметрично (к примеру, если оба сигнала линейно растут), так и не обладать симметрией.

Задание 1

Найти автокорреляционную функцию заданного дискретного сигнала S в 11 точках при $-5 \leq m \leq 5$. Построить графики исходного сигнала и полученной АКФ.

Таблица 4.3

Варианты к первому заданию

Вар.	1,11,21	2,12,22	3,13,23	4,14,24	5,15,25
S	0,5,0,1,2,3,0	3,2,5,0,1,-1,-3	1,2,1,-1,1,-1	0,5,1,-1,1,1,-1	1,2,-2,5,4,5,4
Вар.	6,16,26	7,17,27	8,18,28	9,19,29	10,20,30
S	7,-7,6,-6,5,-5	1,3,5,7,9,7	0,1,0,2,0,3	-1,2,-1,1,-1,0	0,4,2,1,3,1

Задание 2

Найти взаимную корреляцию двух сигналов (в 11 точках при $-5 \leq m \leq 5$) – первый принимается тем же, что и в прошлом задании ($S_1=S$), второй S_2 задан в таблице ниже. Построить графики исходных сигналов и полученной ВКФ.

Таблица 4.4

Варианты ко второму заданию

Вар.	1,11,21	2,12,22	3,13,23	4,14,24	5,15,25
S_2	0,5,1,-1,1,1,-1	1,2,-2,5,4,5,4	0,5,0,1,2,3,0	3,2,5,0,1,-1,-3	1,2,1,-1,1,-1

Bap.	6,16,26	7,17,27	8,18,28	9,19,29	10,20,30
S2	-1,2,-1,1,-1,0	0,4,2,1,3,1	7,-7,6,-6,5,-5	1,3,5,7,9,7	0,1,0,2,0,3

РАБОТА №2. РАЗЛОЖЕНИЕ СИГНАЛА В РЯД ФУРЬЕ ПО ГАРМОНИКАМ

Цель работы: формирование навыков разложения сигналов в ряд Фурье по гармоникам простых сигналов, заданных дискретными отсчетами, а также построение частотного спектра на основе данного разложения.

Планируемая продолжительность: от 1 до 2 академических часов.

Тип работы:  без использования компьютерных средств.

Теоретические основы

Ряд Фурье позволяет разложить произвольный периодический сигнал по гармоническим составляющим посредством выражения:

$$s(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(n \cdot \omega_1 \cdot t) + b_n \sin(n \cdot \omega_1 \cdot t)], \quad (7.1)$$

где ω_1 – основная частота,

$$a_0 = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t) dt, \quad (7.2)$$

$$b_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t) \sin(n \cdot \omega_1 \cdot t) dt. \quad (7.3)$$

$$a_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t) \cos(n \cdot \omega_1 \cdot t) dt. \quad (7.4)$$

Таким образом, сигнал состоит из постоянной составляющей (ее вклад характеризует a_0), которая не зависит от времени, а также бесконечного набора гармонических колебаний (их вклад характеризуют a_n и b_n) с частотами $\omega_n = n \cdot \omega_1$, ($n = 1, 2, \dots$), кратными основной частоте.

В случае дискретного сигнала S , состоящего из N (конечного числа) элементов коэффициенты разложения можно найти путем замены интеграла в (7.2)-(7.4) на сумму:

$$a_0 = \frac{1}{N} \sum_{k=1}^N S_k, \quad (7.5)$$

$$a_n = \frac{1}{N} \sum_{k=1}^N S_k \cos\left(\frac{2\pi n k}{N}\right), \quad (7.6)$$

$$b_n = \frac{1}{N} \sum_{k=1}^N S_k \sin\left(\frac{2\pi n k}{N}\right), \quad (7.7)$$

где k – номер текущего элемента дискретного сигнала S .

Ряды Фурье позволяют выделить не только амплитудно-частотный спектр, но и фазо-частотный. Для нахождения спектров периодического сигнала можно использовать следующие выражения:

- n -я составляющая амплитудно-частотного спектра

$$A_n = \sqrt{a_n^2 + b_n^2}, \quad (7.8)$$

- n -я составляющая фазо-частотного спектра

$$\varphi_n = \arctan\left(\frac{b_n}{a_n}\right). \quad (7.9)$$

Для нахождения дискретного спектра периодического сигнала следует применить формулу (7.10) относительно коэффициентов (7.8) и (7.9), причем результат следует умножать на 2 и отбрасывать элементы с индексами $i > N/2$. Это объясняется тем, что разложение производится относительно положительных и отрицательных частот, что приводит к получению «зеркального» спектра, в котором амплитуды распределены пополам между коэффициентами левой и правой частей.

Таким образом, частотный спектр сигнала представляет из себя массив, соответствующий следующему виду:

$$A = \{A_1, A_2, \dots, A_{N/2}\}. \quad (7.10)$$

Задание

1. Построить график исходного сигнала.
2. Найти коэффициенты разложения a_n по формуле (7.6) и a_0 по формуле (7.5).
3. Найти коэффициенты разложения b_n по формуле (7.7).
4. Найти массив коэффициентов разложения A по формуле (7.10).
5. Построить график спектра сигнала A .

Разрешается округлять значения исходного сигнала до второго знака после запятой при расчетах и построении графиков (к примеру, 0,1097546 можно округлить до 0,11).

Таблица 7.1

Исходный сигнал для каждого из вариантов (10 отсчетов, N=10)

Вар.	Сигнал
1,11, 21	0, 1.53884, 1.53884, 0.363271, -0.363271, 0, 0.363271, -0.363271, -1.53884, -1.53884
2,12, 22	0, 1.53027, 1.23965, -0.288593, -0.942481, 0, 0.942481, 0.288593, -1.23965, -1.53027
3,13, 23	0, 1.7119, 0.470228, -1.05801, -0.760845, 0, 0.760845, 1.05801, -0.470228, -1.7119
4,14, 24	0, 2.37764, -0.293893, -1.46946, 0.475528, 0, -0.475528, 1.46946, 0.293893, -2.37764
5,15, 25	0, 0.864527, 0.828199, 0.122857, -0.276741, 0, 0.276741, -0.122857, -0.828199, -0.864527
6,16, 26	0, 0.646564, -0.0951057, 1.04616, -0.0587785, 0, 0.0587785, -1.04616, 0.0951, -0.646564
7,17, 27	0, 0.637988, -0.394298, 0.394298, -0.637988, 0, 0.637988, -0.394298, 0.394298, -0.637988
8,18, 28	0, 0.855951, 0.176336, -0.529007, -0.285317, 0, 0.285317, 0.529007, -0.176336, -0.855951
9,19, 29	0, 1.42658, -0.293893, -0.881678, 0.475528, 0, -0.475528, 0.881678, 0.293893, -1.42658
10,20, 30	0, 0.484104, -0.593085, 0.357971, -0.103681, 0, 0.103681, -0.357971, 0.593085, -0.484104

РАБОТА №3. РЕШЕНИЕ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ СВЕРТКИ

Цель работы: изучение возможности использования свертки для решения задач.

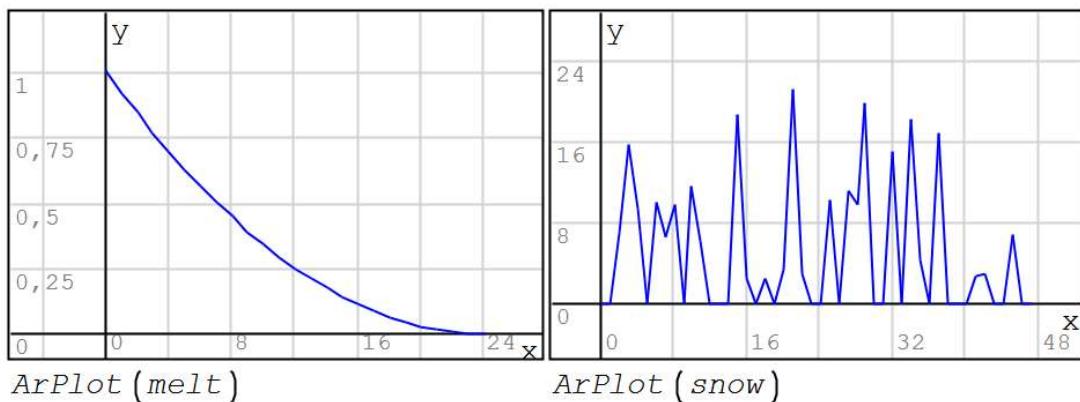
Планируемая продолжительность: от 1 до 2 академических часов.

Тип работы:  без использования компьютерных средств.

Теоретические основы

Свертка может быть применена и для обработки сигналов, которые достаточно легко видеть на практике. В качестве примера возьмем распространенную задачу, в которой два сигнала задают, соответственно, количество выпавшего в тот или иной момент времени снега, а также модель таяния этого снега. Сама задача заключается в том, что по этим известным двум сигналам требуется создать модель, которая позволяла бы в любой момент времени найти количество нерастаявшего снега в килограммах, а также определить по данной модели количество нерастаявшего снега в 14:00 первого и второго дня измерений.

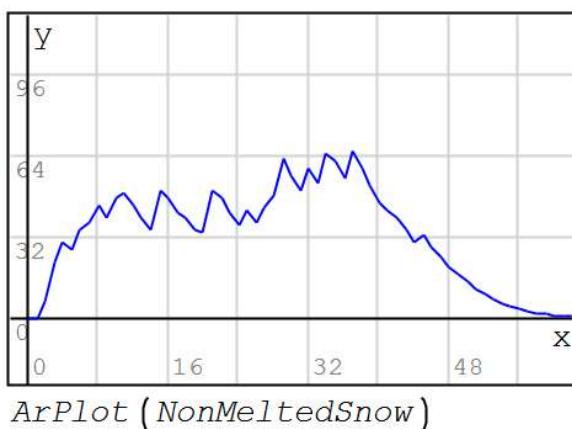
Эта задача, на первый взгляд простая, не может быть решена простым суммированием массы всех выпавших осадков, так как подразумевается, что выдерживается температура, при которой снег тает, причем по модели, заданной вторым сигналом. Рассмотрим эти сигналы.



Слева – график таяния снега при данной температуре (в течение 24 часов), справа – график выпадения снега за двое суток (в течение 48 часов). График таяния снега по оси ординат задает процент нерастаявшего снега ($1=100\%$, $0=0\%$), график выпадения снега – количество выпавшего снега в килограммах.

С точки зрения обработки измерительных сигналов на данную задачу можно взглянуть следующим образом: имеется импульсный сигнал `snow`, который задает величину входного воздействия на систему, имеющую импульсную характеристику `melt`. Такая формулировка условия оправдана, поскольку при выпадении новых осадков прежние продолжают таять по закону `melt` (или, иначе говоря, каждый последующий пик `meltne` может быть просто сложен с предыдущим из-за ослабления вклада предыдущего пика с течением времени). Как было обговорено ранее, свертка в измерительной технике может довольно часто встречаться, как искажающее воздействие импульсной характеристики прибора (или системы) на входной сигнал. В контексте рассматриваемой задачи все идентично, но физического воплощения системы нет, вместо нее принимаются физические свойства снега и текущий температурный режим, которым в совокупности можно приписать характеристику `melt`.

Следовательно, для решения задачи можно применить свертку, используя `melt` в качестве импульсной характеристики. При этом получим следующий график, однозначно определяющий количество нерастаявшего снега в зависимости от времени.



Модель получена, остается найти количество нерастаявшего снега в час дня каждого из дней (сигнал записи осадков имеет длительность 48 часов). Соответственно, требуется взять 13 и $24+13=37$ значения полученного массива.

$$NonMeltedSnow_{13} = 44,3262$$

$$NonMeltedSnow_{37} = 55,5528$$

Задача решена, количество снега в килограммах получено.

Для ручных расчетов можно воспользоваться выражением свертки:

$$(f * g)_{\tau} = \sum_{i=0}^{N+2(M-1)} f_i \cdot g_{\tau-i},$$

где τ – сдвиг,

f – сворачиваемый сигнал,

g – сворачивающий сигнал (импульсная характеристика);

N – число элементов в сигнале f ,

M – число элементов в сигнале g .

Сигналы									Сдвиг	Свертка
0	0	1	2	3	4	5	0	0		
3	2	1	0	0	0	0	0	0	1	$1 \cdot 1 = 1$
0	3	2	1	0	0	0	0	0	2	$1 \cdot 2 + 2 \cdot 1 = 4$
0	0	3	2	1	0	0	0	0	3	$1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1 = 10$
0	0	0	3	2	1	0	0	0	4	$2 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 = 16$
0	0	0	0	3	2	1	0	0	5	$3 \cdot 3 + 4 \cdot 2 + 5 \cdot 1 = 22$
0	0	0	0	0	3	2	1	0	6	$4 \cdot 3 + 5 \cdot 2 = 22$
0	0	0	0	0	0	3	2	1	7	$5 \cdot 3 = 15$

При этом, если элемент при заданных i или $\tau-i$ не существует, его принимают равным нулю. Если обратиться к более ранней задаче с ВКФ, то данная задача будет идентична нахождению ВКФ, с той лишь разницей, что сворачивающий сигнал следует «перевернуть». К

примеру, если $f=\{1,2,3,4,5\}$, а $g=\{1,2,3\}$, то для нахождения их свертки следует проделать процедуру, описанную ниже. При этом, в отличие от ВКФ, для свертки мы используем только положительный сдвиг.

Задание

Для четных и нечетных вариантов задачи отличаются. Для всех вариантов: построить графики заданных сигналов, провести необходимые для решения задачи расчеты, построить требуемую зависимость и отдельно выписать требуемое значение.

Для нечетных. В химической лаборатории в раствор засыпают строго определенное количество слаборастворимых солей раз в минуту. Известна зависимость растворения данных солей в используемом растворе с течением времени. Следует построить модель определения количества нерастворенных солей в растворе в любой момент времени, а также определить массу нерастворенных солей на пятой минуте.

Вариант	Масса засыпанных солей по минутам	Растворение солей по минутам
1	(0,15.9,0,22.1,0,4.9,0,14.5,0,7.1,0,12.1,0,17.7,0,3.9)	(2,1,0)
3	(0,9.3,0,3.3,0,6.7,0,11.3,0,20.9,0,13.9,0,20.9,0,19.9)	(3.9,1.1,0)
5	(0,17.7,0,12.7,0,20.1,0,20.3,0,19.9,0,14.3,0,10.1,0,10.7)	(2.7,0.8,0)
7	(0,9.3,0,18.3,0,14.5,0,5.5,0,19.5,0,8.9,0,16.9,0,9.3)	(2,1,0)
9	(0,16.7,0,16.1,0,8.9,0,9.5,0,6.1,0,16.3,0,11.1,0,9.3)	(3.9,1.1,0)
11	(0,12.7,0,6.9,0,8.7,0,18.3,0,15.5,0,18.9,0,20.9,0,15.7)	(2.7,0.8,0)
13	(0,16.1,0,5.1,0,11.5,0,18.7,0,6.1,0,2.7,0,11.9,0,20.3)	(2,1,0)

Для четных. В опоре измерительном приборе имеется упругий узел, который сжимается в ответ на входное изменение нагрузки в соответствии с некоторой характеристикой (масштаб времени - секунды). Требуется при заданном ряде значений величины посекундной нагрузки построить модель текущей величины сжатия данного узла, а также определить величину сжатия на седьмой секунде.

Вариант	Входные воздействия по секундам	Распрямление узла по секундам
2	(0,15.1,0,4.9,0,10.3,0,5.5,0,18.7,0,21.5,0,13.7,0,0)	(2,1,0)
4	(0,20.7,0,13.7,0,14.5,0,14.1,0,0,19.1,0,20.9,0,20.7,0)	(3.9,1.1,0)
6	(0,8.1,0,14.7,0,0,13.7,0,10.3,0,11.9,0,9.7,0,9.9,0)	(2.7,0.8,0)
8	(0,17.9,0,9.1,0,21.7,0,14.9,0,17.9,0,13.1,0,8.9,0,7.3)	(2,1,0)
10	(0,18.3,0,12.9,0,17.7,0,7.5,0,22.3,0,0,17.9,0,6.9,0)	(3.9,1.1,0)
12	(0,8.3,0,22.3,0,20.3,0,3.1,0,20.5,0,3.9,0,10.5,0,9.5)	(2.7,0.8,0)
14	(0,11.9,0,8.3,0,20.1,0,16.1,0,5.9,0,15.9,0,19.5,0)	(2,1,0)

РАБОТА №4. АВТОКОРРЕЛЯЦИОННАЯ И ВЗАИМНОКОРРЕЛЯЦИОННАЯ ФУНКЦИЯ ДИСКРЕТНЫХ И НЕПРЕРЫВНЫХ СИГНАЛОВ

Цель работы: изучение АКФ и ВКФ дискретных и непрерывных сигналов.

Планируемая продолжительность: 2 академических часа.

Тип работы:  с использованием компьютерных средств.

В данной работе в качестве основного инструмента используется язык программирования Python. Общие принципы решения задач описаны только для Python вне зависимости от какого-либо окружения, поэтому вы можете использовать любой редактор кода, например JupyterLab, Jupyter Notebook, VSCode и т.д.

Теоретические основы

В общем случае взаимная корреляция сигналов $f(t)$ и $g(t)$ описывается выражением:

$$(f \cdot g)(\tau) = \int_{-\infty}^{\infty} \bar{f}(t)g(t - \tau) dt,$$

а в некоторых источниках встречается другое обозначение, отличное знаком перед t :

$$(f \cdot g)(\tau) = \int_{-\infty}^{\infty} \bar{f}(t)g(t + \tau) dt,$$

где $\bar{f}(t)$ обозначает комплексное сопряжение и в случае используемых нами действительных сигналов может быть опущен.

Эта функция характеризует степень «похожести» $f(t)$ на $g(t)$ при текущем сдвиге τ .

То есть, по функции ВКФ двух сигналов можно построить график со значениями сдвига τ по оси абсцисс. На данном графике видно, где сигналы более похожи – при смещении второго на положительный сдвиг (часть графика при $\tau > 0$) или же при его смещении на отрицательный сдвиг (часть графика при $\tau < 0$). Пример графика ВКФ

приведен на рисунке ниже. Здесь мы видим, что сигналы максимально похожи при сдвиге второго из них на $\tau = 2$.

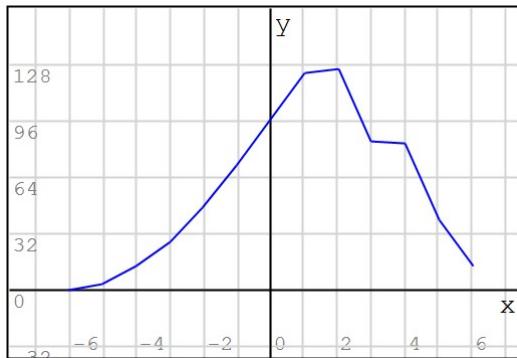


Рис. 4.1. Пример ВКФ

Автокорреляционная функция (АКФ) — это частный случай ВКФ, в котором в качестве второго сигнала используется первый (то есть ВКФ сигнала с самим собой):

$$(f \cdot f)(\tau) = \int_{-\infty}^{\infty} f(t)^* f(t - \tau) dt.$$

Функция АКФ симметрична относительно $\tau = 0$ и обязательно имеет максимум в $\tau = 0$, поскольку сигнал не может быть более похож сам на себя, чем при нулевом сдвиге. В случае с АКФ можно строить только половину графика (обычно ту, что соответствует $\tau \geq 0$) из-за его симметричности. Пример АКФ представлен на рисунке ниже.

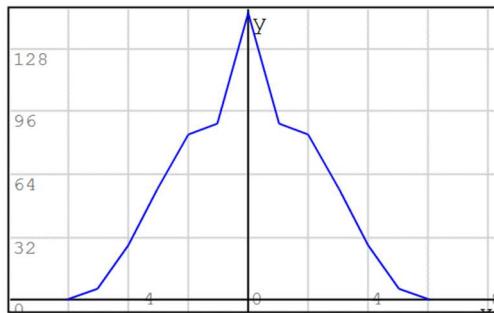


Рис. 4.2. Пример АКФ

АКФ используют для выявления зависимостей в сигналах, не поддающихся визуальному анализу. К примеру, АКФ случайного сигнала имеет максимум при $\tau = 0$ и не имеет никаких выраженных зависимостей на графике при $\tau \neq 0$. Если же, как на рисунке ниже, визуально случайный сигнал является результатом сложения

периодического сигнала (в данном случае синусоиды) и шума, периодическая составляющая будет четко видна на графике АКФ.

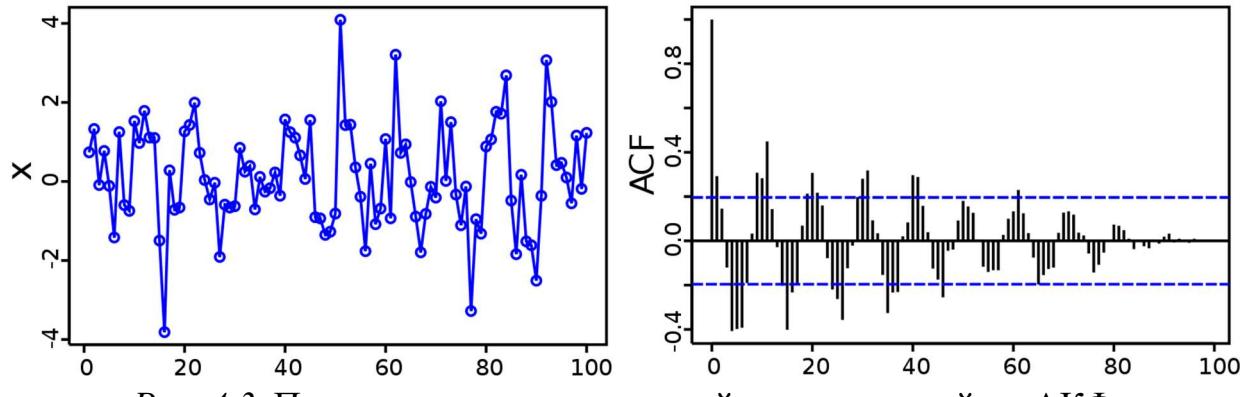


Рис. 4.3. Пример выявления полезной составляющей по АКФ

ВКФ и АКФ дискретного сигнала

Подготовим окружение. Через pip установим все необходимые пакеты.

```
1 ! pip install -U pip
2 ! pip install -U matplotlib
3 ! pip install -U numpy
4 ! pip install -U scipy
5 ! pip install -q ipympl
```

Рис. 4.4. Установка пакетов

И импортируем следующее:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from typing import Callable
4 %matplotlib widget
5 # from google.colab import output
6 # output.enable_custom_widget_manager()
```

Рис. 4.5. Импортирование модулей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

Напишем заранее метод для отображения графика для данных вида $y = f(x)$.

```
1 def plot(x, y):
2     plt.plot(x, y)
3     plt.grid(True)
4     plt.ylabel('y')
5     plt.xlabel('x')
6     plt.show()
```

Рис. 4.6. Функция построения графика

Зададим два дискретных сигнала, в виде массивов чисел с одинаковым размером:

```
1 y_1 = [1, 2, 3, 4, 5, 6, 7]
2 y_2 = [2, 4, 7, 2, 8, 3, 0]
```

Рис. 4.7. Задание массивов

Для осуществления различных преобразований сигналов удобно задавать эти преобразования в общем виде функцией и вызывать для обработки конкретных массивов. Зададим функцию для вычисления ВКФ при текущем сдвиге, ориентируясь на следующее выражение:

$$\sum_{k=0}^{K-n} s_k s_{k-n},$$

где n – сдвиг между массивами (а также номер элемента в ВКФ),

k – номер элемента в первом массиве.

То есть, ВКФ двух массивов также будет массивом, причем

каждый из его элементов будет рассчитываться как сумма произведений всех перекрывающихся элементов. Чтобы задать функцию вычисления текущего (k-го) элемента ВКФ, можно реализовать следующий метод.

```
1 def cross_corelation_element(
2     array_1: np.ndarray,
3     array_2: np.ndarray,
4     lag: int) -> float:
5     """
6         При отрицательном lag
7             становится меньше размера одного из исходных массивов на
8             значение lag, что не позволяет выйти за пределы второго массива.
9         """
10    last_index = 0
11    if lag < 0:
12        last_index = len(array_1) + lag
13    else:
14        last_index = len(array_1)
15
16    """
17        При положительном сдвиге данное условие заставляет
18            суммирование начать не с первого элемента
19            массивов, а с lag + 1, поскольку иначе будут перемножаться не
20            перекрывающиеся элементы. Для первого массива всегда будут взяты
21            элементы, начиная с нулевого.
22    """
23    begin_index = 0
24    if lag >= 0:
25        begin_index = lag + 1
26
27    result = []
28    for i in range(begin_index, last_index):
29        result.append(array_1[i] * array_2[i - lag])
30    return sum(result)
```

Рис. 4.8. Алгоритм вычисления текущего (k-го) элемента ВКФ

Пояснения в шести кавычках писать не нужно, это всего лишь комментарий к тому, как работает алгоритм. Также необязательно писать типизацию к входным и выходным параметрам, это, например `np.ndarray` или `-> float`. Такая типизация может помочь интегрированным средам разработки, понять, что за объект будет на

входе и выходе, и вызвать подсказки к методам в соответствии с типом, если среда опять же эти подсказки поддерживает.

Теперь нужно задать сдвиг. Как было сказано в рамках прошлого занятия, для дискретных отсчетов функция корреляции может быть найдена в диапазоне сдвига от $-(N - 1)$ до $(N - 1)$, где N – число отсчетов в сигнале. Напишем для этого функцию, приведённую ниже.

```
1 def lags(array):
2     return np.arange(-(len(array) - 1), len(array), 1, dtype=int)
```

Рис. 4.9. Функция создания смещений на основе существующего массива

Чтобы вручную не задавать пределы массива смещений на вход будем передавать некоторый массив и на его основе получаем N встроенной функцией `len`. При помощи функции `np.arange` создаём массив смещений от $-(\text{len}(\text{array}) - 1)$ до $-(\text{len}(\text{array}) - 1)$ (второй аргумент в `np.arange` берёт предел не включительно).

Теперь осталось скомбинировать наши функции, при каждом сдвиге нужно записать результат вычисления текущего (k -го) элемента ВКФ в массив.

```
1 def cross_corelation(
2     array_1: np.ndarray,
3     array_2: np.ndarray) -> list:
4     result = []
5     for lag in lags(array_1):
6         result.append(cross_corelation_element(array_1, array_2, lag))
7     return result
```

Рис. 4.10. Функция создания смещений на основе существующих массивов

Вызовем эти функции и нарисуем полученные графики при помощи реализованного в самом начале метода `plot`.

```
1 cc = cross_corelation(y_1, y_2)
2 plot(np.arange(0, len(y_1)), y_1)
3 plot(np.arange(0, len(y_1)), y_2)
4 plot(lags(y_1), cc)
```

Рис. 4.11. Вызов функций

Таким образом получим следующие графики.

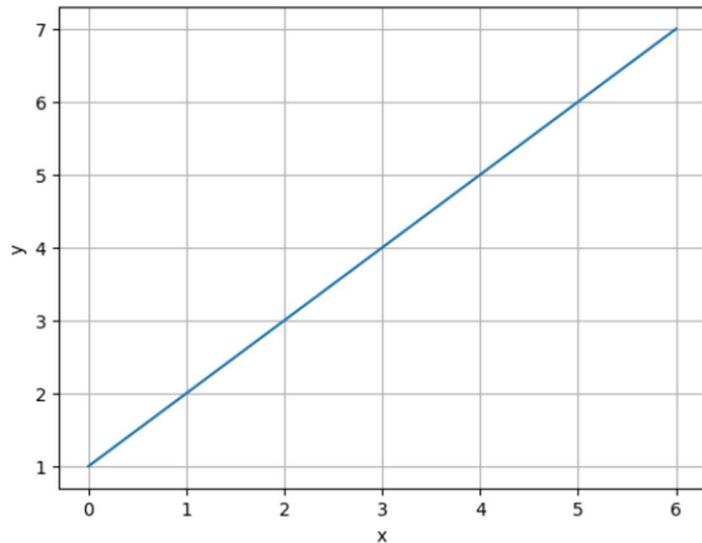


Рис. 4.12. График сигнала y_1

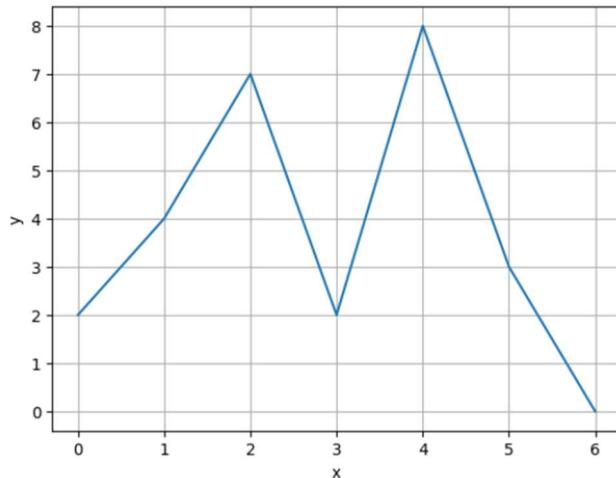


Рис. 4.13. График сигнала y_2

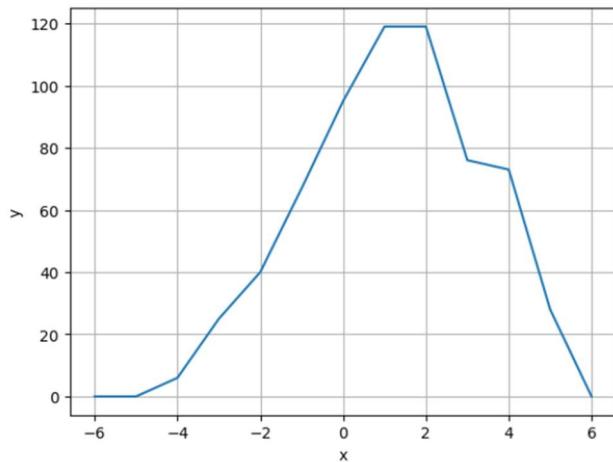


Рис. 4.14. График ВКФ y_1 и y_2

Как видно из последнего графика наиболее похожей функция y_1 похожа на y_2 при смещении равном 2.

Теперь найдем АКФ для сигнала, заданного некоторой функцией. Для этого сначала реализуем функцию $y = \sin(\frac{x\pi}{12})$ по которой будем создавать массив данных.

```
1 def func_1(x):
2     return math.sin((x * math.pi) / 12)
```

Рис. 4.15 Функция для генерации данных

Также зная то, что автокорреляционная функция (АКФ) – это частный случай ВКФ, в котором в качестве второго сигнала используется первый (то есть ВКФ сигнала с самим собой) напишем функцию АКФ.

```
1 def auto_corelation(
2     array: np.ndarray
3 ) -> list:
4     return cross_corelation(array, array)
```

Рис. 4.16 Функция АКФ

Теперь создадим массив данных по `func_1` при помощи функции `np.linspace`, которая создаст массив от 1 до 100 на 100 элементов с равными промежутками между элементами и выведем её.

```
1 x = np.linspace(1, 100, 100)
2 y_1 = [func_1(value) for value in x]
3 plot(np.linspace(1, 100, 100), y_1)
```

Рис. 4.17. Алгоритм создания данных

Получим график нашей функции на рисунке 4.18.

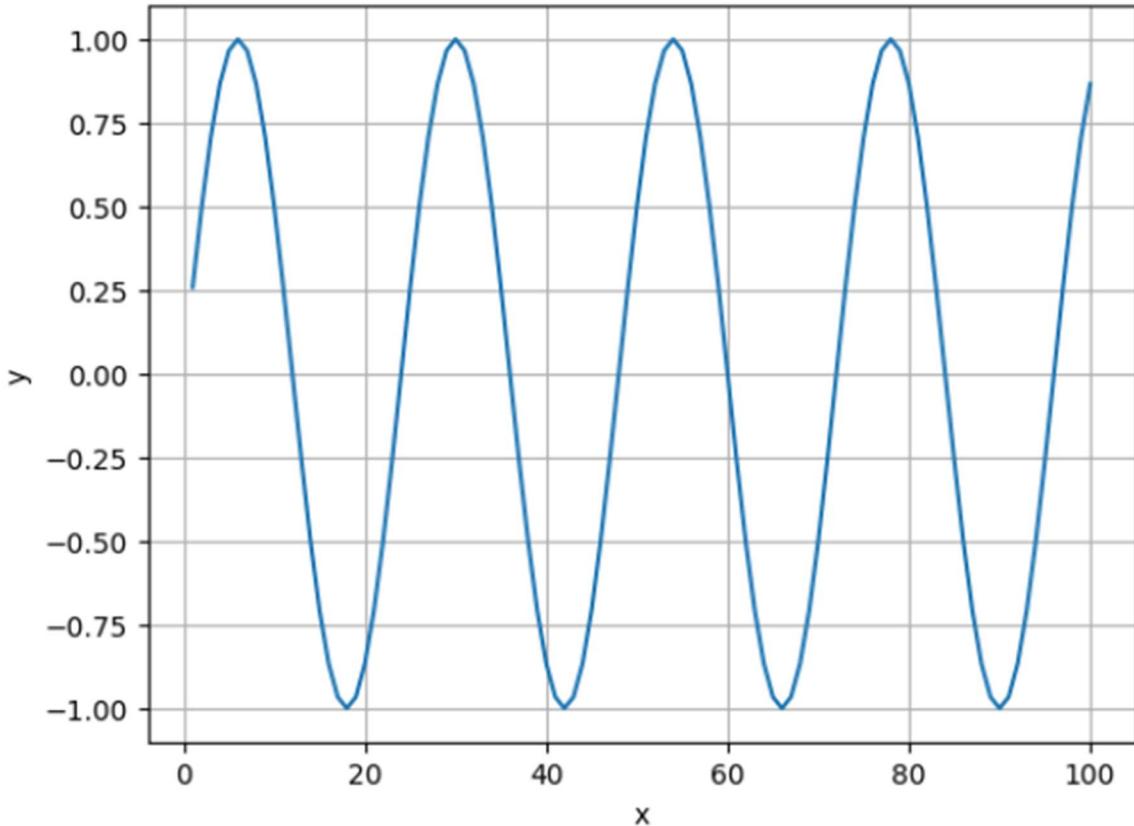


Рис. 4.18. График функции синуса

Осталось рассчитать АКФ y_1 и вывести полученные данные в график.

```
1 plot(lags(y_1), auto_corelation(y_1))
```

Рис. 4.19. Вызов графика функции АКФ с расчётом

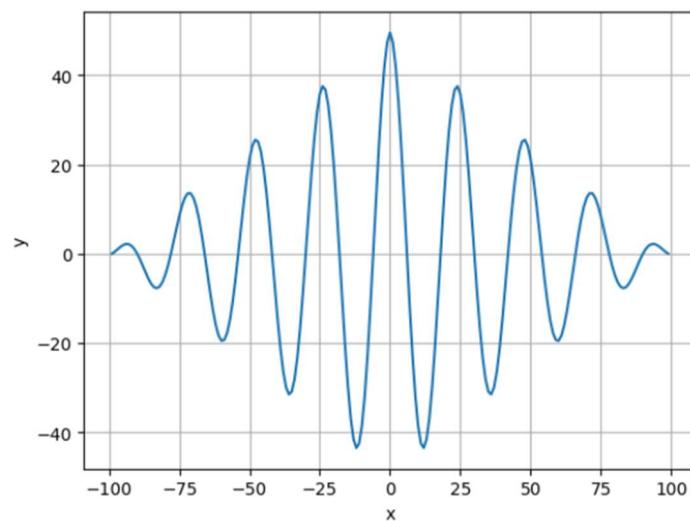


Рис. 4.20. График АКФ

АКФ и ВКФ непрерывных сигналов

Теперь рассмотрим возможность нахождения аналитического выражения АКФ для простых примеров. В нашем случае АКФ будет вычисляться согласно выражению:

$$\int_0^{10} y(x) y(x - T) dx,$$

где бесконечные пределы заменены на интервал $[0;10]$ для простоты вычислений.

Обратите внимание, для самостоятельной работы при работе с АКФ сигналы в таблице с вариантами обозначаются как $y(x)$ и $y2(x)$.

Реализуем функцию `contigious_corelation`. Обратите внимание, теперь функции не являются массивами данных, теперь это Callable объект, иными словами, сама функция без её вызова. Помимо этого, теперь в нашу функцию передаём переменные a и b – это пределы интегрирования. Также стоит упомянуть внутреннюю функцию `mul_func`, для интегрирования нам не нужен результат работы функции, нам нужна сама функция, поэтому при помощи внутренней функции `mul_func`, которая обладает областью видимости данных до входных переменных от `contigious_corelation` указываем их перемножение при вызове `mul_func`, указывая во входных параметрах x и параметр смещения T . Для интегрирования используем функцию `quad`, во входных параметрах передаём функцию `mul_func`, пределы интегрирования и в дополнительный аргументах `args` выставляем смещение. Функция `quad` возвращает кортеж из двух элементов, а именно значение интеграла и абсолютную ошибку. Абсолютная ошибка нам не нужна, поэтому пропускаем её, выставляя прочерк вторым аргументом

```

1 def contiguous_corelation(
2     func_1: Callable,
3     func_2: Callable,
4     laggs: np.ndarray,
5     a: int,
6     b: int,
7 ) -> list:
8     def mul_func(x, T):
9         return func_1(x) * func_2(x - T)
10    result = []
11    for lag in laggs:
12        integral, _ = quad(mul_func, a, b, args=(lag,))
13        result.append(integral)
14    return result

```

Рис. 4.21. Функция ВКФ для непрерывного сигнала

Создаём вторую функцию, от которой будем брать ВКФ.

```

1 def func_2(x):
2     return math.sin(x) + 1

```

Рис. 4.22. Вторая функция для ВКФ

Теперь осталось задать смещения вышеупомянутой функцией `np.linspace` вызвать реализованную функцию `contiguous_corelation` и отобразить график результата.

```

1 laggs = np.linspace(-50, 50, 100)
2 cc = contiguous_corelation(func_1, func_2, laggs, 0, 10)
3 plot(laggs, cc)

```

Рис. 4.23. Вызов ВКФ для непрерывного сигнала и графика для него

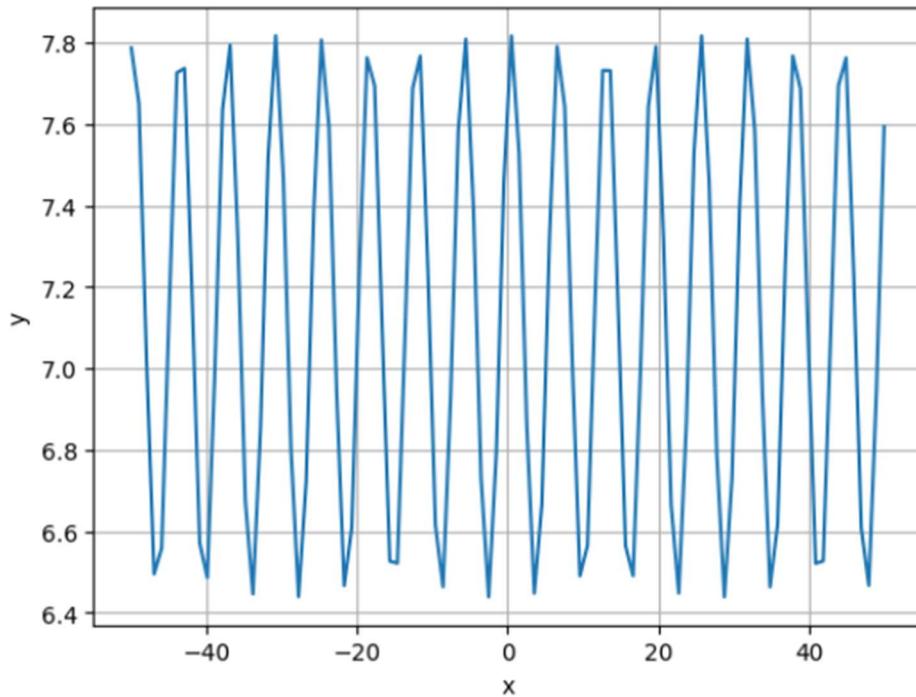


Рис. 4.24. График ВКФ

По аналогии с дискретным сигналом получить АКФ сигнала можно вызвав ВКФ с одинаковыми функциями. Реализуем данный подход и получим метод `auto_contigious_corelation`.

```

1 def auto_contigious_corelation(
2     func: Callable,
3     laggs: np.ndarray,
4     a: int,
5     b: int
6 ) -> list:
7     return contiguous_corelation(func, func, laggs, a, b)

```

Рис. 4.25. АКФ
для непрерывного сигнала

Как и до этого создаём смещения, вызываем функцию АКФ и строим график.

```

1 laggs = np.linspace(-50, 50, 100)
2 cc = auto_contigious_corelation(func_1, laggs, 0, 10)
3 plot(laggs, cc)

```

Рис. 4.26. Вызов АКФ

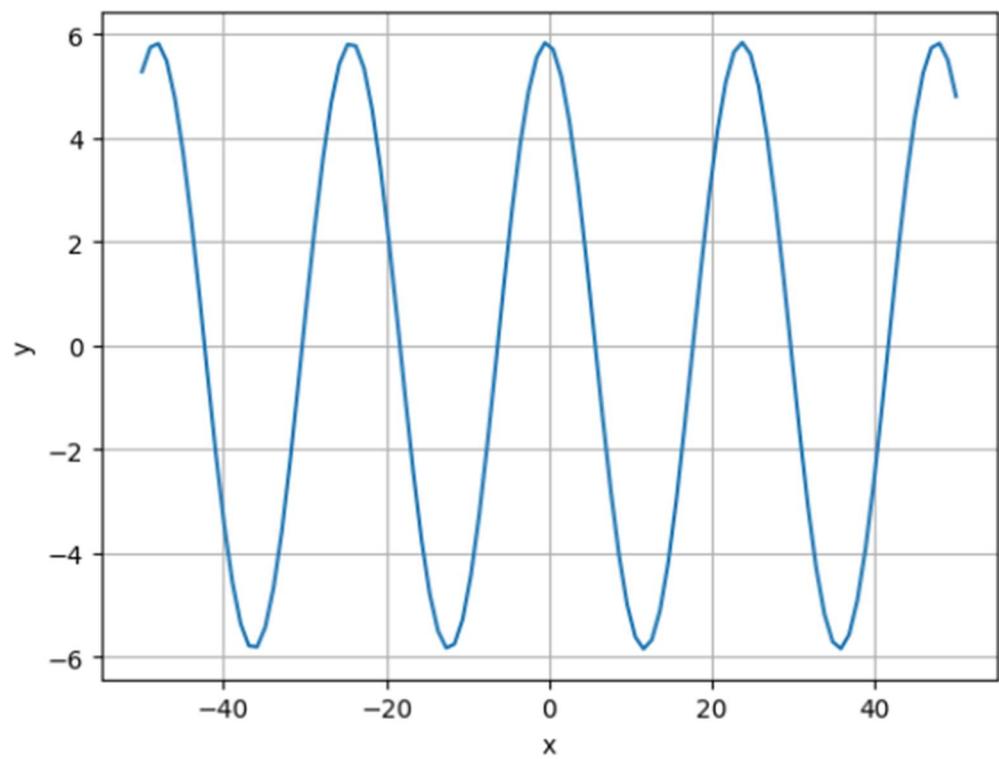


Рис. 4.27. График АКФ для непрерывной функции

Задание

Указанные значения заменяют собой соответствующие параметры для раздела, посвященного дискретным отсчетам и АКФ непрерывной функции.

Таблица 4.1

Задания для второй работы

Var.	1,6,11, 16,21,26	2,7,12, 17,22,27	3,8,13, 18,23,28	4,9,14, 19,24,29	5,10,15, 20,25,30
y(x)	$2*(\text{Sin}(x/10))$	$2*(\text{Cos}(x/10))$	$0.5*(\text{Sin}(x/7))$	$0.5*(\text{Cos}(x/7))$	$1.3*(\text{Sin}(x/3))$
s1	0.5,0,1,2,3,0	3,2.5,0,1,-1,-3	1,2,1,-1,1,-1	0.5,1,-1,1,1,-1	1,2,-2,5,4.5,4
s2	7,-7,6,-6,5,-5	1,3,5,7,9,7	0,1,0,2,0,3	-1,2,-1,1,-1,0	0,4,2,1,3,1
y2(x)	y(x-3)	y(x-1)	y(x-5)	y(x-4)	y(x+2)

РАБОТА №5. ДИСКРЕТНОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

Цель работы: формирование навыков применения прямого и обратного преобразования Фурье для дискретных сигналов, а также их использования для получения частотного спектра сигнала.

Планируемая продолжительность: 2 академических часа.

Тип работы:  с использованием компьютерных средств.

Основные понятия преобразования Фурье для дискретных сигналов

В данной работе будут употреблены следующие сокращения:

ДПФ – дискретное преобразование Фурье (преобразование Фурье для сигналов, заданных массивами чисел);

ОДПФ – обратное дискретное преобразование Фурье (восстановление исходного массива из коэффициентов, полученных с помощью ДПФ);

БПФ – быстрое преобразование Фурье (является разновидностью ДПФ, значительно ускоренной, но только для массивов, количество элементов которых равна 2^m , где $m=1,2,\dots$);

ОБПФ – обратное быстрое преобразование Фурье (восстановление исходного массива из коэффициентов, полученных с помощью БПФ);

Преобразование Фурье переводит дискретный или непрерывный сигнал из временной в частотную область (по оси x вместо времени t следует откладывать частоту ω), причем полученные коэффициенты имеют комплексный вид. Модуль этих коэффициентов показывает распределение амплитуд по частотам в сигнале, а аргумент – распределение фазовых сдвигов по частотам.

Начнем с рассмотрения ДПФ. Формула для его реализации известна (задает k -й коэффициент Фурье на основе n -го элемента исходного массива x , который состоит из N элементов):

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i n k}{N}}. \quad (5.1)$$

Восстановить же сигнал можно согласно формуле ОДПФ:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i n k}{N}}. \quad (5.2)$$

БПФ является частным случаем ДПФ, когда $N=2^m$, где m – произвольное натуральное число. Выигрыш достигается за счет отдельного расчета четных и нечетных коэффициентов. После разделения ДПФ на четные и нечетные коэффициенты можно к следующему выражению:

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{4\pi i n}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{2\pi i (2n+1)k}{N}}. \quad (5.3)$$

которое и задает БПФ.

Число математических операций при этом значительно уменьшается по сравнению с ДПФ. ОБПФ можно получить аналогично ОДПФ:

$$X_n = \sum_{k=0}^{\frac{N}{2}-1} x_{2k} e^{\frac{4\pi i nk}{N}} + \sum_{k=0}^{\frac{N}{2}-1} x_{2k+1} e^{-\frac{2\pi i (2k+1)n}{N}}. \quad (5.4)$$

Реализация преобразования Фурье в Python

Начнём подготавливать окружение. Установим необходимые библиотеки.

```
1 ! pip install -U pip
2 ! pip install -U matplotlib
3 ! pip install -U numpy
4 ! pip install -U scipy
5 ! pip install -q ipympl
```

Ruc. 5.1. Установка библиотек

Импортируем следующие зависимости:

```
1 import numpy as np
2 import math
3 import scipy
4
5 from matplotlib import pyplot as plt
6
7 from scipy.fft import fftfreq, fft
8 from typing import Callable
9 from random import randint
10
11 %matplotlib widget
12 # from google.colab import output
13 # output.enable_custom_widget_manager()
```

Ruc. 5.2. Импортирование зависимостей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

Из предыдущей работы возьмём метод `plot` и немного изменим, чтобы можно было отрисовывать на графике N сигналов.

```
1 def plot(*args):
2     ax = plt.figure()
3     for idx in range(0, len(args), 2):
4         x, y = args[idx], args[idx + 1]
5         plt.plot(x, y)
6     plt.grid(True)
7     plt.ylabel('y')
8     plt.xlabel('x')
9     plt.show()
```

Рис 5.3. Функция отрисовки графиков

Реализуем функцию сигнала. Пусть это будет простейший $y(x)=\sin(2\pi x)$.

```
1 def y_1(x):
2     return math.sin(2*x*math.pi)
```

Рис 5.4. Функция сигнала

Теперь нужно создать дискретные отсчёты на основе этой функции, для этого нужно задать количество дискретных отсчётов и диапазон отсчётов, при помощи функции [np.arange](#).

```
1 DISCRETE_COUNT_MAX = 32
2 discrete_range = np.arange(stop=DISCRETE_COUNT_MAX)
```

Рис. 5.5. Дискретные отсчёты

На основе полученного диапазона и нашей функции создадим массив дискретных значений нашей функции через отдельный метод. То есть каждое значений диапазона передаём в функцию, разделив на общее количество значений.

```

1 def discrete_values(
2     func: Callable,
3     discrete_range: np.ndarray,
4 ) -> np.ndarray:
5     result = np.zeros(discrete_range.shape)
6     max_value = discrete_range.size
7     for idx, value in enumerate(discrete_range):
8         result[idx] = func((value) / max_value)
9     return result

```

Рис 5.6. Функция дискретных значений

Также можете заметить, `result` является `np.ndarray` (потому что сгенерирован через `np.zeros`), в отличие от предыдущей работы, где в качестве результата был нативный `list`. Вам никто не запрещает использовать списковое включение (`list comprehension`), но помните, работать это будет медленнее, python для списков выделяет память динамически, и не всегда блок памяти будет в одной области, в отличие от `np.zeros`, где память резервируется под размер нашего массива. Как пример из жизни, нативный список – это как набитая полка с книгами, куда вам нужно положить ещё и свои книги, вы определяете, где свободное место, если места нет, то какие книги можно выбросить. NumPy массив – грубо говоря выкидывает все книги, и сразу же их складывает.

Теперь вызовем нашу функцию `discrete_values`, напечатаем значения и нарисуем график данных

```

1 dv = discrete_values(y_1, discrete_range)
2 print(dv)
3 plot(discrete_range, dv)

```

Рис 5.7. Вызов и отрисовка дискретных значений

Получим следующее:

```
[ 0.0000000e+00  1.95090322e-01  3.82683432e-01  5.55570233e-01
 7.07106781e-01  8.31469612e-01  9.23879533e-01  9.80785280e-01
 1.0000000e+00  9.80785280e-01  9.23879533e-01  8.31469612e-01
 7.07106781e-01  5.55570233e-01  3.82683432e-01  1.95090322e-01
 1.22464680e-16 -1.95090322e-01 -3.82683432e-01 -5.55570233e-01
-7.07106781e-01 -8.31469612e-01 -9.23879533e-01 -9.80785280e-01
-1.0000000e+00 -9.80785280e-01 -9.23879533e-01 -8.31469612e-01
-7.07106781e-01 -5.55570233e-01 -3.82683432e-01 -1.95090322e-01]
```

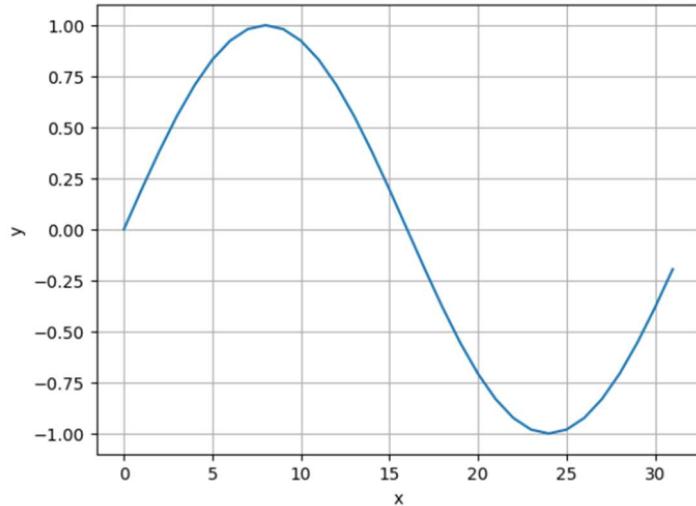


Рис. 5.8. Данные и их график

Реализуем метод расчёта ДПФ в соответствии с формулой 5.1. Как видим, функция должна принимать номер выходного коэффициента k и имя входного массива в качестве параметров возвращает мнимую сумму

```
1 def dft_func(
2     array: np.ndarray,
3     k: int
4 ) -> np.imag:
5     def F(item, n):
6         return item * math.e ** (
7             (-1) * ((2 * math.pi * k * n * 1j) /
8                 array.size - 1)
9         )
10
11     result = np.zeros(array.size, dtype=np.complex_)
12     for idx in range(array.size):
13         result[idx] = F(array[idx], idx)
14     return np.sum(result)
```

Рис. 5.9. Метод дискретного преобразования Фурье

Теперь используя реализованный метод `dft_func` создадим ещё один метод формирования массива ДПФ.

```

1 def count_discrete_fourier_on(
2     array: np.ndarray,
3 ) -> np.ndarray:
4     result = np.zeros(array.shape, dtype=np.complex_)
5     for idx in range(array.size):
6         result[idx] = discrete_fourier_transform(array, idx)
7     return result

```

Рис. 5.10. Метод формирования массива ДПФ

Нарисуем по абсолютному значению результат работы этого массива для наших дискретных значений.

```

1 fourier_array = count_discrete_fourier_on(discrete_values)
2 plot(discrete_range, np.abs(fourier_array))

```

Рис 5.11. – Вызов и отрисовка функции
формирования массива Фурье

Получим следующий график:

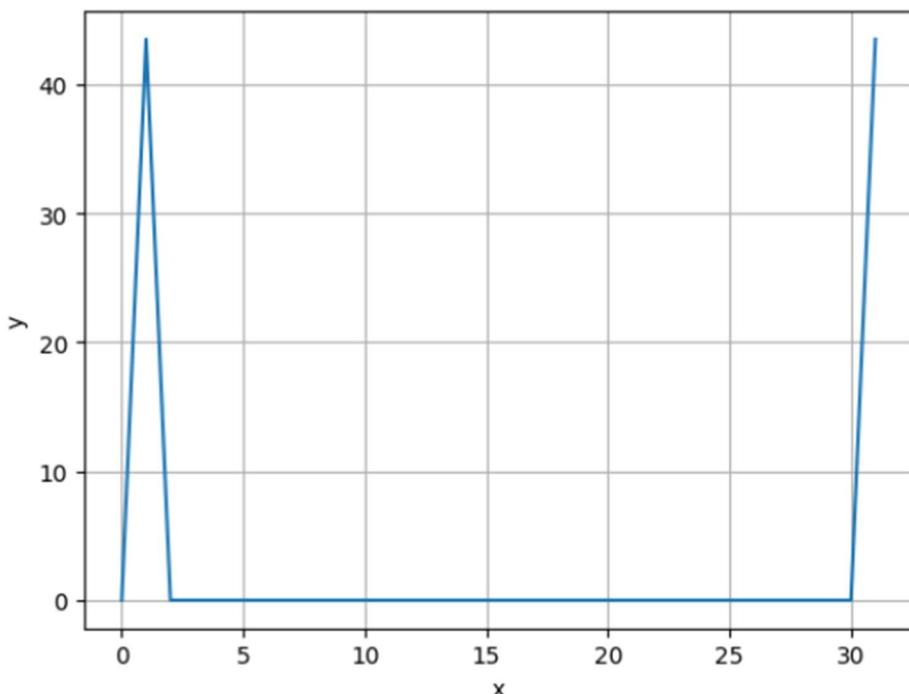


Рис. 5.12. Спектр сигнала

Как видим, мы получили спектр сигнала. Спектр сигнала симметричен (если не считать самый первый коэффициент, который характеризует наличие постоянной составляющей). Это объясняется тем, что преобразование Фурье, в общем случае, осуществляется в

пределах от $-\infty$ до ∞ , и правая половина спектра неинформативна.

В данном случае, максимальным информативным коэффициентом является $(N/2+1)$ -й, поскольку он соответствует половине частоты дискретизации, выше этого значения физически полезные данные не могут быть найдены.

При помощи инструмента масштабирования приблизим первый пик на графике.

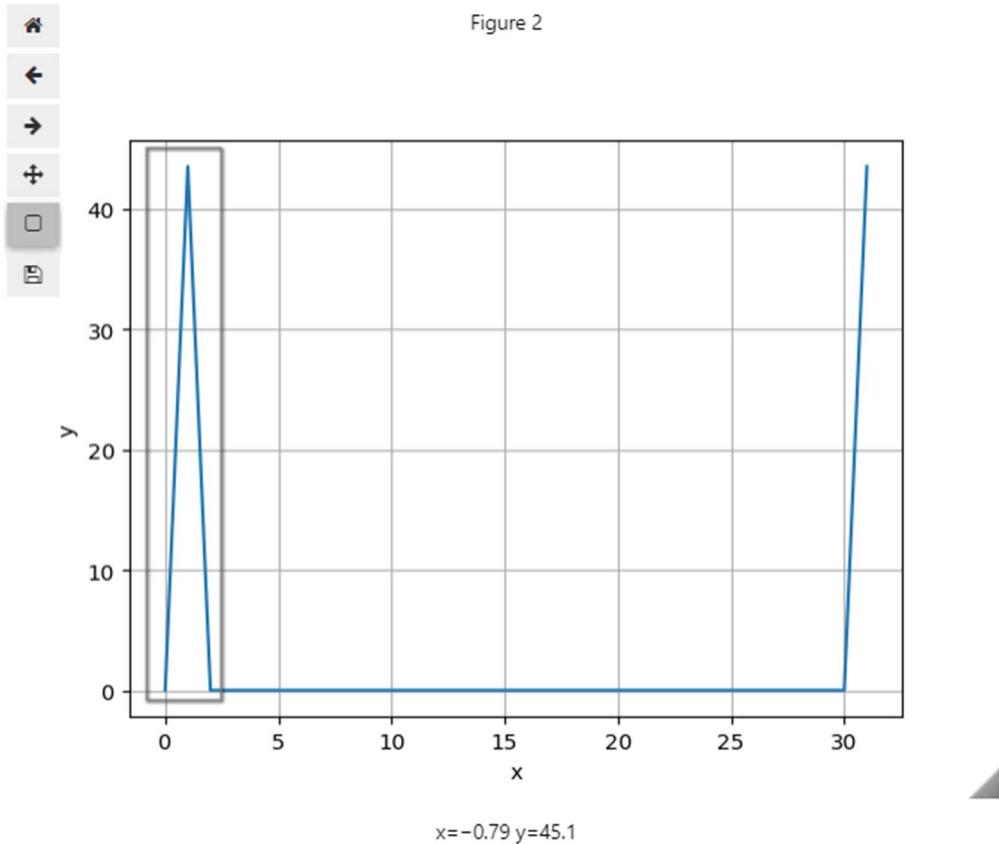


Рис. 5.13. Масштабирование графика

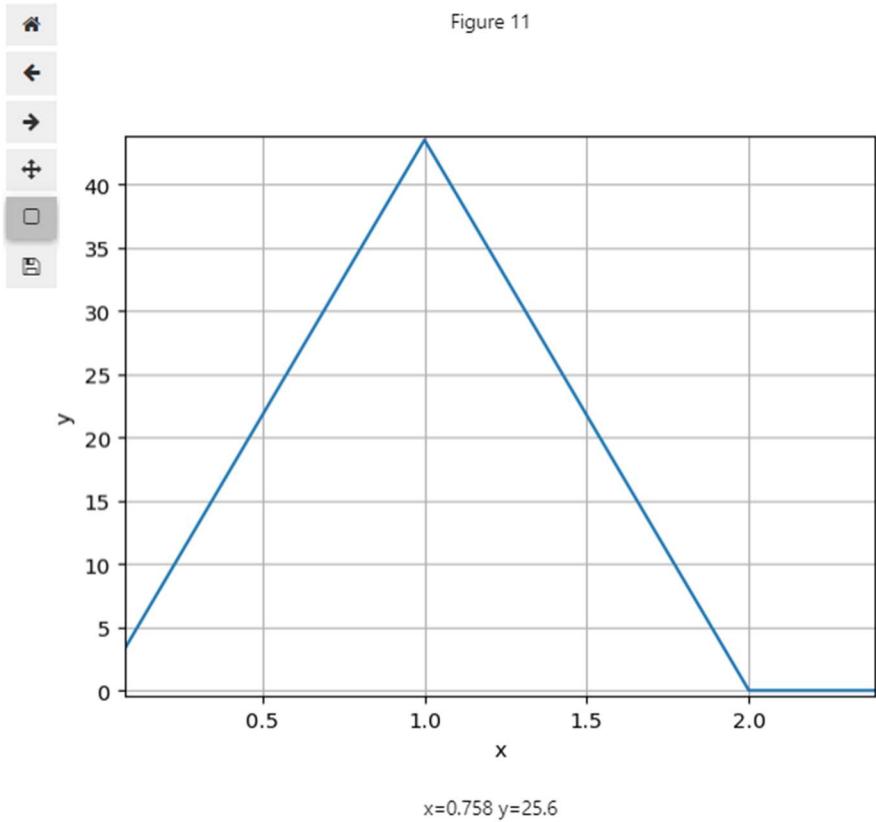


Рис. 5.14. Первый пик спектра

Анализируя спектр, можно сказать, что в сигнале присутствует только одна частота, соответствующая второму коэффициенту Фурье (первому, если обозначить коэффициент, соответствующий постоянной составляющей, как нулевой). Это суждение корректно, поскольку исходный сигнал сформирован с помощью функции $\sin(2\pi x)$, то есть содержит одну частоту, равную 1 Гц.

Теперь реализуем обратную задачу, а именно ОДПФ по формуле 5.2. Повнимательнее приглядимся к функциям 5.1 и 5.2, у 5.2 суммирующая часть ничем не отличается от 5.1, кроме знака у степени, поэтому мы можем использовать ту же функцию `dft_func`, только на вход передавать отрицательный индекс и дописать деление

```

1 def reverse_fourier_transform(
2     array: np.ndarray
3 ) -> np.array:
4     result = np.zeros(array.shape, dtype=np.complex_)
5     for idx in range(array.size):
6         result[idx] = (1 / len(array)) * \
7             (discrete_fourier_transform(array, (-1) * idx))
8     return result

```

Рис. 5.15. Метод формирования массива ОДПФ

Вызовем эту функцию на полученном ранее массиве ДПФ и выведем график:

```

1 reversed_fourier_array = reverse_fourier_transform(fourier_array)
2 plot(discrete_range, reversed_fourier_array)

```

Рис. 5.16. Вызов ОДПФ с вызовом отрисовки графика

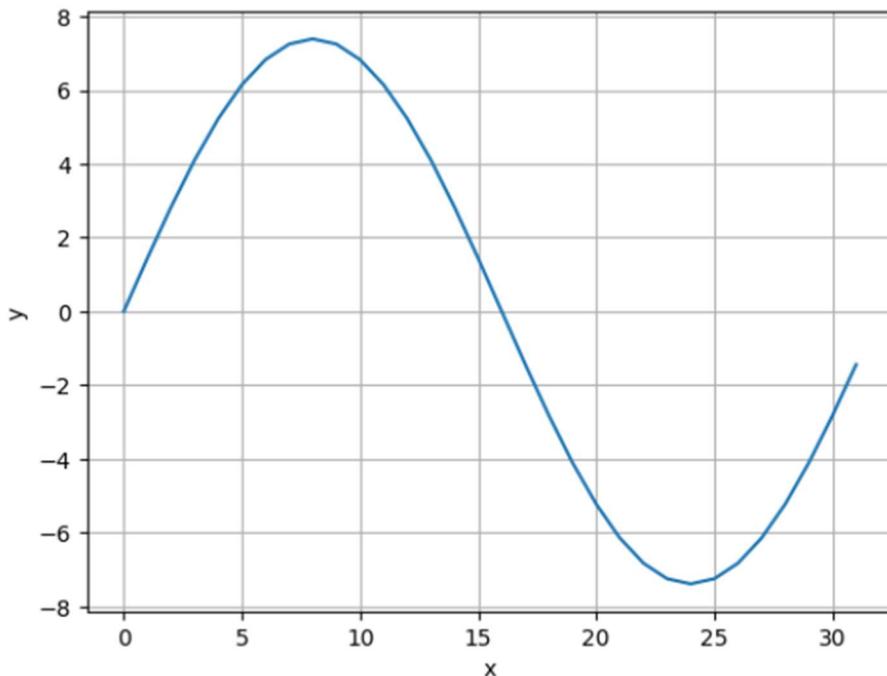


Рис. 5.17. График восстановленной по ОДПФ функции

Таким образом получили график исходной функции на основе ряда Фурье.

Чаще всего, в математических пакетах уже существует реализация БПФ, как одного из наиболее значимых инструментов анализа сигналов. Для этой задачи будем использовать SciPy и его функции `fft` и `fftfreq`.

Зададим новый сигнал, причем состоящий из двух синусоид различных частот и шумовой компонент.

```
1 def y_2(x):
2     return np.sin(2 * math.pi * 50 * x) + np.sin(2 * math.pi * 120 * x) + (randint(0, 100) / 1000)
```

Рис. 5.18. Функция сигнала

Введём переменные количество точек сигнала – SIGNAL_POINTS, частоту дискретизации – SAMPLE_SPACING, x – в данном случае будет массив отсчётов, этот массив мы передаём в нашу функцию, выведем график функции. В fft передаём массив данных получаем массив из БПФ, далее на основе этого массива создаём массив частот при помощи fftfreq. Так как полезная часть находится в первой половине – делим массив частот по полам.

```
1 SIGNAL_POINTS = 600
2 SAMPLE_SPACING = 1.0 / 800
3 x = np.arange(0.0, stop=SIGNAL_POINTS*SAMPLE_SPACING, step=SAMPLE_SPACING)
4 y = y_2(x)
5 plot(x, y)
6 yf = fft(y)
7 xf = fftfreq(SIGNAL_POINTS, SAMPLE_SPACING)[:SIGNAL_POINTS//2]
8 plot(xf, 2.0/SIGNAL_POINTS * np.abs(yf[0:SIGNAL_POINTS//2]))
```

Рис. 5.19. БПФ со спектром сигнала

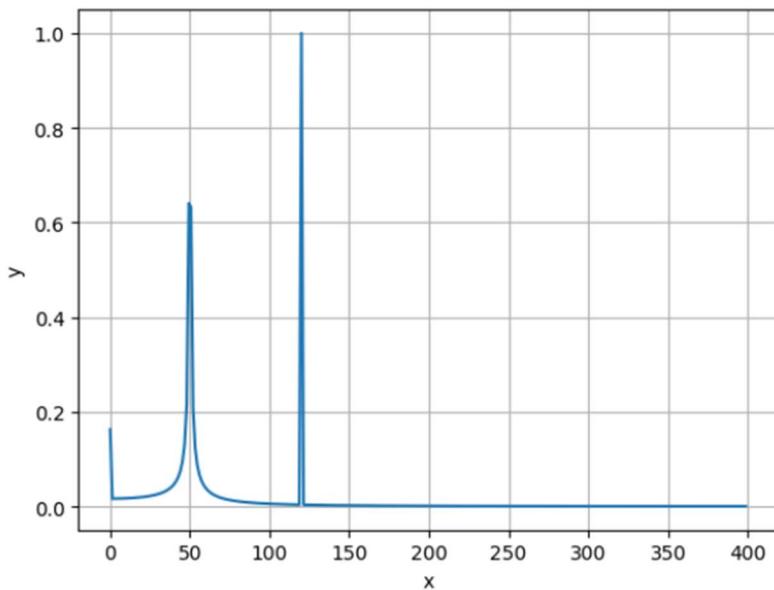
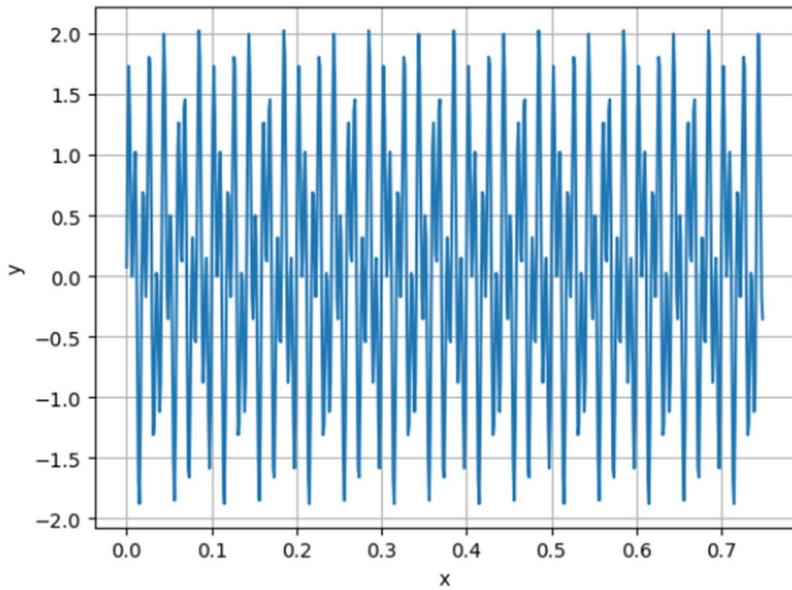


Рис. 5.20. График сигнала и его спектра

Можно заметить, что на исходном сигнале визуально едва ли заметны доминирующие частоты, тогда как на спектре сигнала четко представлены два пика, соответствующие этим частотам.

Задание 1

В соответствии с описанным материалом реализуйте ДПФ, ОДПФ и БПФ, построив соответствующие графики. При этом, используйте в качестве $y(x)$ и $f(x)$ указанные в таблице ниже функции, в соответствии с вариантом. Для всех вариантов в $f(x)$ присутствует случайная аддитивная помеха (`randint(0, 100) / 1000`).

Таблица 5.1

Варианты заданий

Варианты	$y(x)$	$f(x)$
1,6,11, 16,21,26	$2 * (\sin[4\pi x])$	$\sin(2 \cdot \pi \cdot 14 \cdot x) + \sin(2 \cdot \pi \cdot 70 \cdot x) + \frac{\text{random}(100)}{1000}$
2,7,12, 17,22,27	$\cos[3\pi x]$	$\cos(2 \cdot \pi \cdot 25 \cdot x) + 0,5 \cdot \sin(2 \cdot \pi \cdot 70 \cdot x) + \frac{\text{random}(100)}{1000}$
3,8,13, 18,23,28	$0.5 * (\sin[8\pi x])$	$0,3 \cdot \cos(2 \cdot \pi \cdot 31 \cdot x) + 1,5 \cdot \cos(2 \cdot \pi \cdot 39 \cdot x) + \frac{\text{random}(100)}{1000}$
4,9,14, 19,24,29	$0.5 * (\cos[10\pi x])$	$0,1 \cdot \sin(2 \cdot \pi \cdot 11 \cdot x) + \cos(2 \cdot \pi \cdot 100 \cdot x) + \frac{\text{random}(100)}{1000}$
5,10,15, 20,25,30	$1.3 * (\sin[14\pi x])$	$\sin(2 \cdot \pi \cdot 73 \cdot x) + \sin(2 \cdot \pi \cdot 83 \cdot x) + \frac{\text{random}(100)}{1000}$

Применение корреляционной функции и преобразования фурье для исследования сигналов

Рассмотренные ранее преобразования крайне часто встречаются на практике для анализа различных сигналов — как детерминированных, так и стохастических.

Для анализа будем использовать сигнал, заданный выражением

```
1 def signal(x):
2     return ((np.exp(9.24 * x * 1j) - np.exp((-1j) * 9.24 * x)) / (
3         2 * 1j
4     ))
```

На его основе создадим дискретный массив из 64 значений с шагом dx и наложенными на них случайными значениями из массива rnd и выведем .

```
1 SIGNAL_LEN = 64
2 STEP = 0.0312
3 rnd = np.random.randint(low=-1, high=1, size=SIGNAL_LEN)
4 x = np.arange(start=0, stop=SIGNAL_LEN) * STEP
5 y_1 = signal(x) + rnd
6 plot(x, y_1)
```

Рис. 5.21. Создание массива значений сигнала

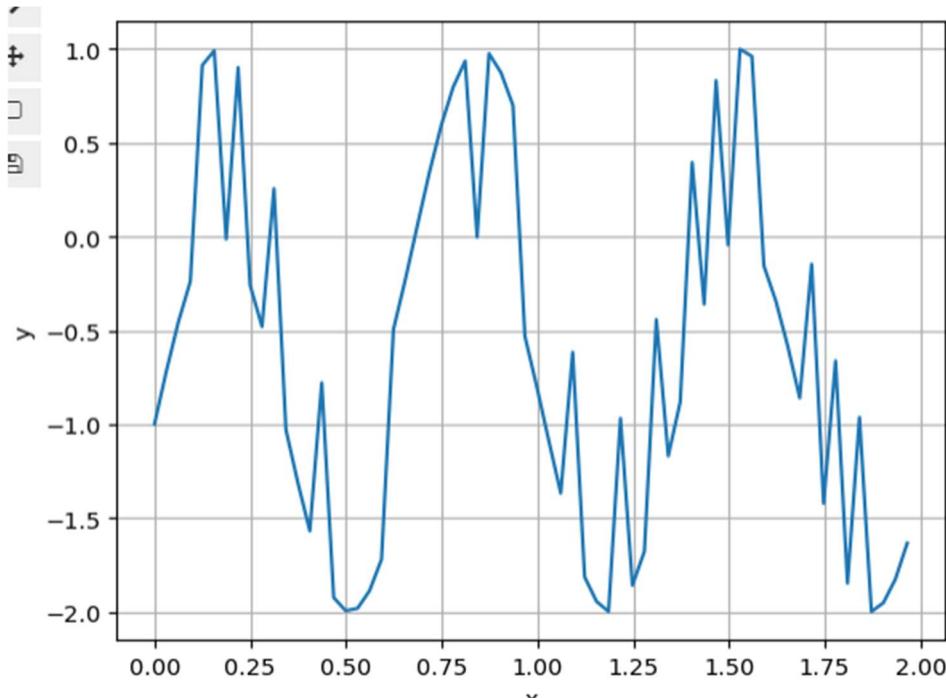


Рис. 5.22. График сигнала

При поиске полезной информации в такого рода сигналах (визуально идентичных шуму), чаще всего исходят из предположения, что полезная информация имеет периодический характер. Для выявления периодических составляющих сигнала следует воспользоваться АКФ, поскольку данное преобразование, за счет «сдвигания» сигнала относительно самого себя позволяет подчеркнуть повторяющиеся области.

Как было сказано в предыдущей работе, АКФ нельзя применять к сигналу, смешенному относительно оси x. Для устранения постоянной составляющей, вычтем из каждого значения массива y_1 его математическое ожидание, сформировав массив y_2 .

```
1 mean = np.mean(y_1)
2 y_2 = y_1 - mean
```

Рис. 5.23. Устранение постоянной составляющей

Для формирования массива АКФ воспользуемся функцией [corelate](#) из модуля `numtr` после чего выведем график получившийся график.

```
1 auto_corelation_y_2 = np.correlate(y_2, y_2, mode='full')
2 auto_corelation_x = np.arange(start=-SIGNAL_LEN + 1, stop=SIGNAL_LEN)
3 plot(auto_corelation_x, auto_corelation_y_2)
```

Рис. 5.24. Создание АКФ и его отображение на графике

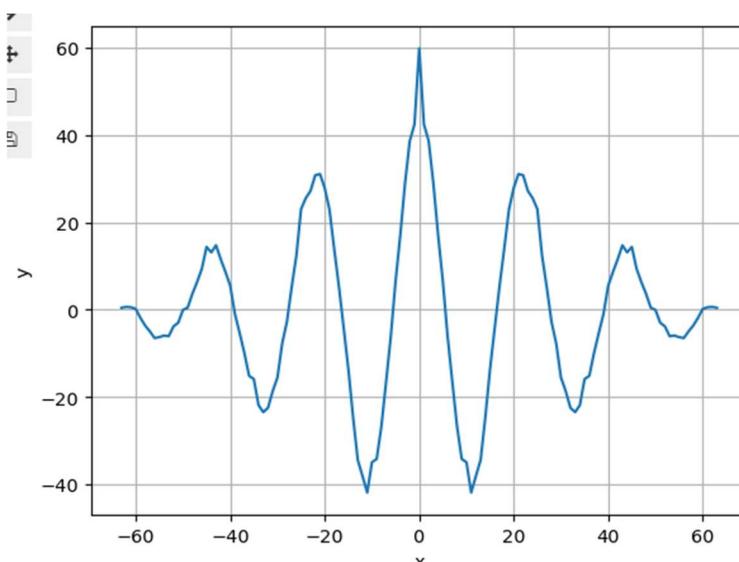


Рис. 5.24. График АКФ

На основе полученного графике можно сделать исследование. В общем случае, если уровень шума не чрезмерно высок, а в сигнале присутствует периодическая составляющая, то на графике АКФ различить ее будет очень просто. Если же уровень шума значительный (что может приводить к потере полезной информации), либо периодических составляющих в сигнале не содержится – график АКФ будет представлять собой затухающий по обе стороны от нуля импульс с заполнением реализацией случайной величины (затухающий от центра к краям шум). И только в нулевой точке будет наблюдаться пик, поскольку реализация случайного процесса значительно коррелирует сама с собой только при отсутствии сдвига. По графику АКФ ниже очевидно, что сигнал содержит периодическую полезную составляющую.

Выявив, имеется ли полезная составляющая в сигнале, следует определить ее свойства, при наличии. Поскольку в данной работе под полезной понимается периодическая составляющая, задача определения свойств сводится к вычислению характерной частоты сигнала. Характерной частотой (или характерными частотами) называют доминирующую частоту или группу частот, сопровождающих весь исследуемый процесс. В данном случае, для выявления характерной частоты лучше всего воспользоваться исследованием спектра как модуля ДПФ.

```
1 SIGNAL_SIDE = np.arange(0, (SIGNAL_LEN - 1) / 2)
2 fs = SIGNAL_SIDE * (1 / (SIGNAL_LEN * STEP))
3 fourier_array = np.abs(count_discrete_fourier_on(y_1))[:SIGNAL_SIDE.size]
```

Рис. 5.25. Массивы частоты и спектра по ДПФ

Нулевая частота есть постоянная составляющая, поэтому ее учитывать при выявлении характерной частоты не следует. Чтобы выявить характерную частоту достаточно найти максимум на графике (не соответствующий постоянной составляющей). Обычно характерная частота сильно, в два и более раз выделяется на фоне прочих, шумовых составляющих.

Построив график спектра, видим присутствие в нем характерной частоты между 0 и 4 Гц.

```
1 plot(fs, fourier_array)
```

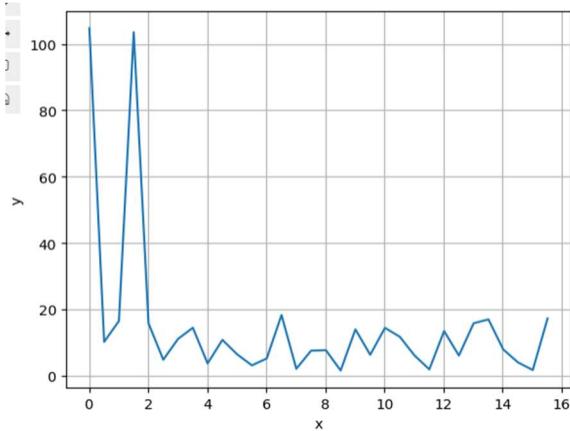


Рис. 5.26. Спектр сигнала

Приближаем график в данной области, для определения характерной частоты. Становится ясно, что она равна 1,5 Гц.

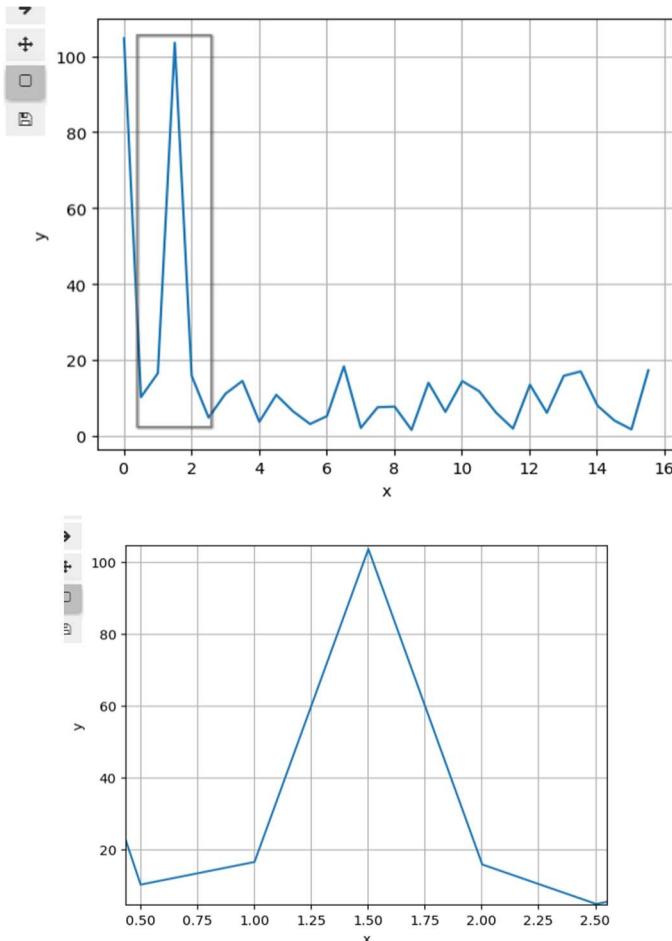


Рис. 5.27. Фрагмент спектра

Для иллюстрации результата задаем сигнал с такой частотой, $\sin(1.5 \cdot 2\pi \cdot x)$, и построим его график вместе с графиком исходных данных.

```
1 sin_func = np.sin(1.5 * 2 * math.pi * x)
2 plot(x, y_1, x, sin_func)
```

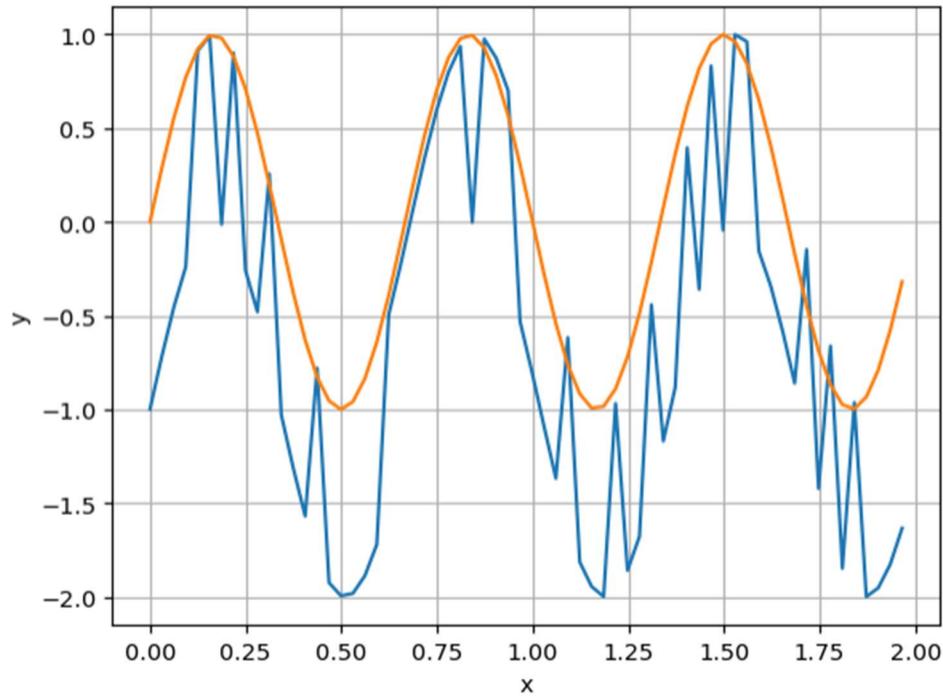


Рис. 5.28. Исследуемый сигнал с наложенной полезной составляющей

Задание 2

1. Задать указанные параметры из собственного варианта.
2. Построить график получившегося сигнала y_1 .
3. Выявить наличие/отсутствие периодической составляющей в заданном сигнале.
4. При наличии периодической составляющей определить значение характерной частоты для заданного сигнала.
5. Построить график сигнала с найденной характерной частотой вместе с графиком исходных данных.

Варианты ниже меняют вид выражения `signal`, а также параметр `dx`. Прочие переменные из примера выше остаются прежними.

Таблица 8.1

Варианты задания

Вариант	signal(x)	dx
1,6,11, 16,21,26	$\frac{\exp(i \cdot 25,1327 \cdot x) - \exp(-i \cdot 25,1327 \cdot x)}{2 \cdot i}$	0.0117
2,7,12, 17,22,27	$\frac{\exp(i \cdot 31,4159 \cdot x) - \exp(-i \cdot 31,4159 \cdot x)}{2 \cdot i}$	0.0094
3,8,13, 18,23,28	$\frac{\exp(i \cdot 6,2832 \cdot x) - \exp(-i \cdot 6,2832 \cdot x)}{2 \cdot i}$	0.0469
4,9,14, 19,24,29	$\frac{\exp(i \cdot 18,8496 \cdot x) - \exp(-i \cdot 18,8496 \cdot x)}{2 \cdot i}$	0.0156
5,10,15, 20,25,30	$\frac{\exp(i \cdot 12,5664 \cdot x) - \exp(-i \cdot 12,5664 \cdot x)}{2 \cdot i}$	0.0234

РАБОТА №6. ЧАСТОТНО-ВРЕМЕННЫЕ СПОСОБЫ АНАЛИЗА СИГНАЛОВ С ПРИМЕНЕНИЕМ ПРЕОБРАЗОВАНИЯ ФУРЬЕ

Цель работы: изучение возможностей частотно-временного анализа сигналов на базе ДПФ.

Планируемая продолжительность: от 2 до 4 академических часов.

Тип работы:  с использованием компьютерных средств.

Коротко-временное преобразование Фурье в Smath Studio

Проблема ДПФ и НПФ в том, что результаты этих преобразований не зависят от времени, то есть не могут показать изменение частотного/фазового спектра на протяжении длительности сигнала. Для решения этой проблемы на базе ДПФ реализуют коротко-временное преобразование Фурье (КВПФ), являющееся **частотно-временным** способом анализа сигналов. Результатом этого преобразования является двумерный массив, каждый столбец которого представляет собой частотный спектр на основе ДПФ для какого-то короткого участка исследуемого сигнала, причем каждый последующий столбец также является частотным спектром, но для последующих участков сигнала той же длительности. КВПФ используется, к примеру, в кодировщике звуковых файлов в формат MP3 (файл MP3 не содержит значения амплитуд звука, в нем хранятся спектры коротких фрагментов исходного звукового файла).

Импортируем и сконфигурируем следующие зависимости.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib widget
4 # from google.colab import output
5 # output.enable_custom_widget_manager()
```

Рис. 6.1. Импортирование и конфигурирование зависимостей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

Из предыдущей работы возьмём функцию построения простых графиков.

```
1 def plot(*args):
2     ax = plt.figure()
3     for idx in range(0, len(args), 2):
4         x, y = args[idx], args[idx + 1]
5         plt.plot(x, y)
6     plt.grid(True)
7     plt.ylabel('y')
8     plt.xlabel('x')
9     plt.show()
```

Рис. 6.2. Функция построения графиков

```
1 def dft(array: np.ndarray) -> np.ndarray:
2     """ Discrete Fourier Transform"""
3     def discrete_fourier_function(
4         array: np.ndarray,
5         k: int
6     ) -> np.imag:
7
8         def F(item, n):
9             return item * np.e ** (
10                 (-1) * ((2 * np.pi * k * n * 1j) /
11                     array.size - 1)
12             )
13
14         result = np.zeros(array.size, dtype=np.complex_)
15         for idx in range(array.size):
16             result[idx] = F(array[idx], idx)
17         return np.sum(result)
18
19     result = np.zeros(array.shape, dtype=np.complex_)
20     for idx in range(array.size):
21         result[idx] = discrete_fourier_function(array, idx)
22     return result
```

Рис. 6.3. Реализация ДПФ

Спектр сигнала – это модуль его преобразования Фурье. Зададим функцию получения спектра, правой половины (половина массива модуля ДПФ, каждый элемент умножен на двое).

```
1 def spectrum(array: np.ndarray) -> np.ndarray:
2     tmp_freqs = np.abs(array)
3     tmp_freqs = tmp_freqs[tmp_freqs.size // 2:]
4     return 2 * tmp_freqs / array.size
```

Рис. 6.4. Формирование спектра

Так как КВПФ вычисляется для отдельных фрагментов сигнала, зададим функцию фрагментирования на части размера size массива сигнала array.

```
1 def fragmentize(
2     array: np.ndarray,
3     size: int) -> np.ndarray:
4     fragments = np.array_split(array, range(size, array.size, size))
5     fragments = np.array(fragments)
6     return fragments
```

Рис. 6.5. Функция создания фрагментов

На основе этих функций можно вычислить КВПФ, как двухмерный массив, содержащий «слепленные» в виде вертикальных полос спектры для каждого из последовательно идущих фрагментов.

```
1 def stft(array: np.ndarray):
2     """Short Timed Fourier Transform"""
3     tmp = []
4     for part in array:
5         tmp.append(spectrum(part))
6     return np.array(tmp)
```

Рис. 6.6. Функция вычисления КВПФ

Зададим сигнал у.

```
1 def y(x):
2     if x < np.pi:
3         return np.sin(0.5 * 2 * np.pi * x)
4     else:
5         return np.sin(0.8 * 2 * np.pi * x)
```

Рис. 6.7. Сигнал

И на основе этого сигнала у создадим 50 значений

```
1 SAMPLES_COUNT = 50
2 signal_array = np.zeros(SAMPLES_COUNT)
3 for i in range(1, SAMPLES_COUNT):
4     signal_array[i - 1] = y((2 * np.pi * i) / SAMPLES_COUNT)
```

Рис. 6.8. Сигнал

Теперь нужно реализовать функцию попадания (или не попадания) в поддиапазон диапазона частот, отрисовывать график мы будем при помощи `pcolormesh`, поэтому диапазоны будем создавать с учётом его особенностей. Возвращать данная функция будет две переменные: равномерно разделённый диапазон, и двумерный массив `binned[y][x]`, где элементы по `y` – это поддиапазон частот, а элементы по `x` – это значения спектра для частей, обращение к элементу по `y x` вернёт либо 0, либо 1, что соответствует попаданию или непопаданию в поддиапазон для соответственных гармоник спектра.

```

1 def make_bins(values: np.ndarray):
2     """
3         Создаём диапазоны для частей,
4         где за горизонтальное значение берётся
5         индекс части, а по вертикали не/попадание
6         в диапазон данных
7         """
8     bins = np.linspace(np.min(values), np.max(values), num=100)
9     binned = np.zeros(shape=(bins.shape[0], values.shape[0]), dtype=np.int8)
10
11    for line_idx in range(len(values)):
12        for el_idx in range(values[line_idx].size):
13            """Итерируемся по значениям частот"""
14            current_value = values[line_idx, el_idx]
15
16            for bin_idx in range(0, bins.size, 2):
17                high_value = bins[bin_idx + 1]
18                """
19                    Если частота попала в диапазон относительно
20                    большего значения поддиапозона, то выставляем попадание
21                    """
22                if high_value >= current_value:
23                    binned[bin_idx - 1, line_idx] = 1
24                    break
25
26    """Добавляем ещё одно значение для pcolormesh (иначе график не будет отрисовываться)"""
27    bins = np.append(bins, np.max(values) + bins[0] - bins[1])
28    return bins, binned

```

Рис. 6.9. Функция попадания в диапазон

На основе всех реализованных функций теперь можно создать несколько графиков с разными размерами фрагментов (или же размерами окон). В данном случае будем использовать размеры, при которых массив из 50 значений поделится на равные части. Затем итерируемся по этим размерам и создаём на их основе фрагменты массива сигнала, для этих фрагментов высчитываем КВПФ. Создаём диапазоны и выводим всё полученное на графики

```
1 SIZES = [1, 5, 10, 25, 50,]
2 fig, axs = plt.subplots(nrows=1, ncols=len(SIZES), figsize=(15, 10))
3 fig.tight_layout(pad=2.5)
4 for idx, size in enumerate(SIZES):
5     fragment_size = size
6     fragments = fragmentize(signal_array, fragment_size)
7     stft_array = stft(fragments)
8
9     bins, values = make_bins(stft_array)
10    x_graph = np.arange(1, fragments.shape[0] + 2)
11    y_graph = bins
12    z_graph = values
13
14    axs[idx].set_title(f'Размер фрагмента: {fragment_size}')
15    axs[idx].set_ylabel(f'Частота')
16    axs[idx].set_xlabel(f'Номер фрагмента')
17    axs[idx].pcolormesh(
18        x_graph,
19        y_graph,
20        z_graph,
21        shading='flat',
22        vmin=stft_array.min(),
23        vmax=stft_array.max())
```

Рис. 6.10. Алгоритм построения графиков спектров

Итого получаем следующие графики

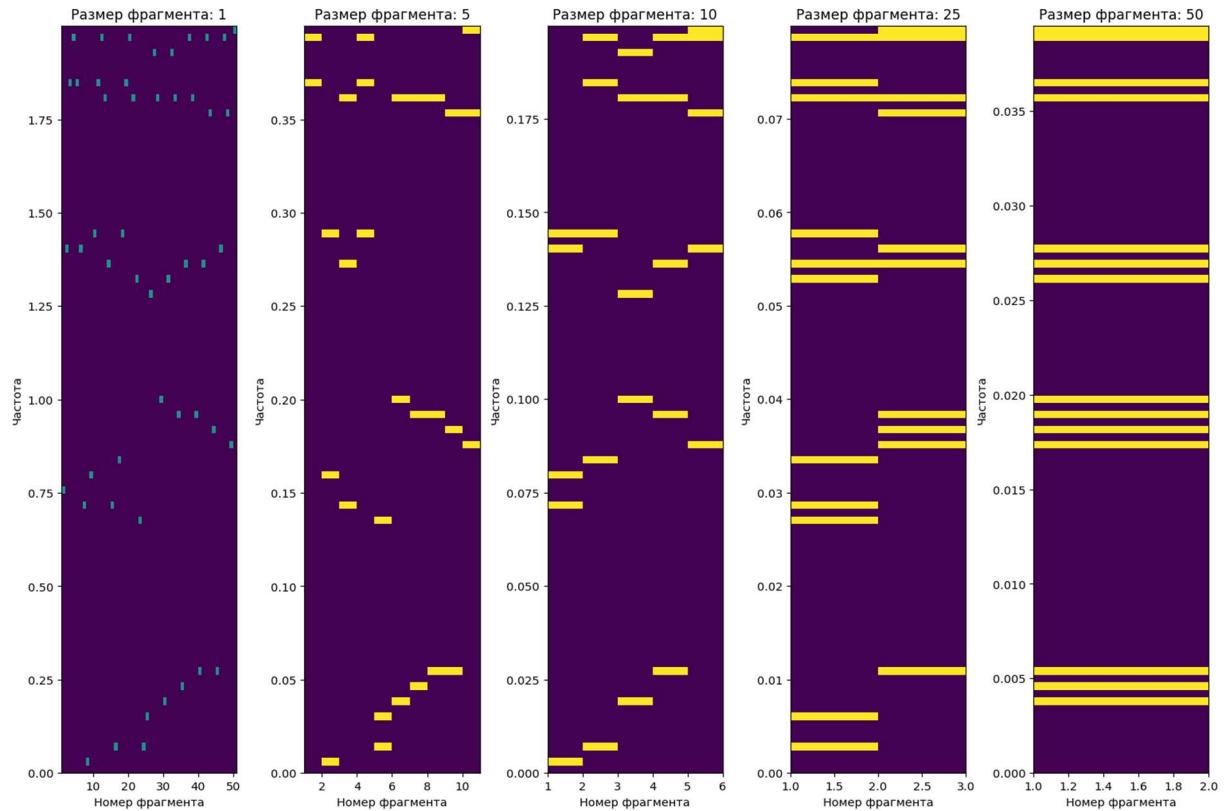


Рис. 6.11. Графики КВПФ для фрагментов разного размера

На основе этих графиков подтверждается основное свойство КВПФ: **при увеличении размера окна, увеличивается разрешение по частоте, но уменьшается разрешение по времени.** Иными словами, чем больше значений в фрагменте, тем больше частот мы можем увидеть в нём, но при этом точно сказать какая частота изменилась в определённый момент времени будет затруднительно, и наоборот для узкого окна, мы видим частоту в конкретный момент времени, но на большой дистанции мы не видим преобладающих частот.

Оконное преобразование Фурье

Еще одним способом анализа сигналов является оконное преобразование Фурье (ОКПФ). Его можно получить, если в выражении НПФ умножить под интегралом исследуемый сигнал на некоторое окно $W()$:

$$F(t, \omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(\tau) W(\tau - t) e^{-i\omega\tau} d\tau, \quad (6.1)$$

или для дискретного сигнала S:

$$F(m, \omega) = \frac{1}{\sqrt{2\pi}} \sum_{n=0}^N S_n \cdot W_{n-m} \cdot e^{-i\omega}. \quad (6.2)$$

В таком виде выражение F() для непрерывного и дискретного случаев задает, очевидно, ВКФ исследуемого сигнала с функцией, заданной произведением комплексной экспоненты на окно.

Назначение оконной функции – сгладить края анализируемого фрагмента сигнала, снизив выбросы. В качестве W выбирают функцию, которая на краях плавно приближается к нулю, и поскольку она умножается на сигнал, сам сигнал на краях также будет стремиться к нулю, из-за чего и сгладятся его края. Это необходимо, в первую очередь, когда речь идет о пофрагментном ДПФ – по сути, рассмотренное выше КВПФ является подобием ОКПФ (без плавного сдвига по времени), в котором используется прямоугольное окно (равно 0 вне фрагмента и 1 внутри фрагмента). Также очевидно, что оконная функция сдвигается вдоль сигнала, а значит **ОКПФ является частотно-временным способом анализа сигналов.**

Реализация ОКПФ

Обратите внимание, здесь используются функции из раздела КВПФ, если данный раздел вы делаете в отдельном документе – не забудьте скопировать в него зависимые функции.

Сперва выберем оконную функцию. Для простоты возьмем распространенное на практике окно Хэмминга.

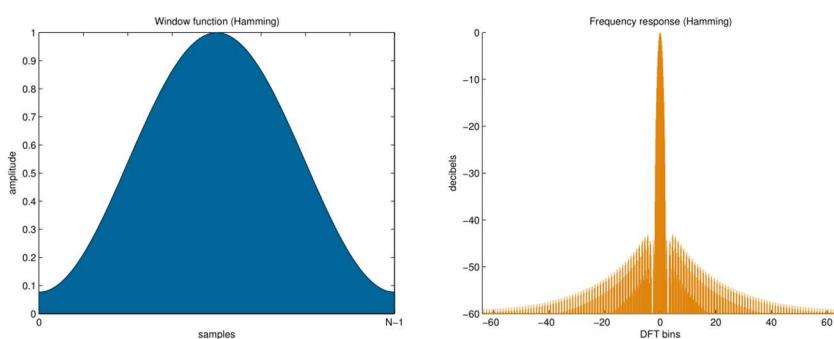


Рис. 6.12. Окно Хэмминга и его спектр

Окно задается выражением:

$$W_n = 0.5 \left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right),$$

где N - ширина окна. Чем шире окно, тем лучше разрешение по частоте, но хуже по времени и наоборот. В примере ниже N задает также число элементов в массиве окна.

```
1 def hem_window(N: int) -> np.ndarray:
2     result = np.arange(start=1, stop=N + 1)
3     result = 0.5 * (1 - np.cos(2 * ((np.pi * result) / (N - 1))))
4     return result
5 N = 10
6 plot(np.arange(N), hem_window(N))
```

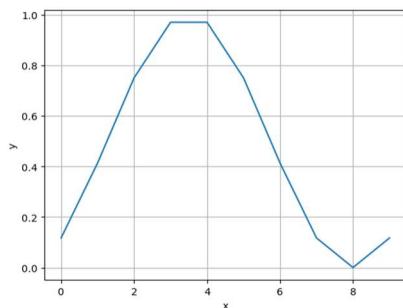


Рис. 6.13. Окно Хэмминга

Из предыдущей работы возьмём функцию `dft` и немного её видоизменим. Добавим внутреннею функцию `window`, и в функции `F` значение элемента будем умножать на значения окна Хэмминга, если они попадают в диапазон этого окна, если они не попадают, то умножаем на 0.

```

1 def wft(array: np.ndarray,
2         m: int,
3         hem_window_array: np.ndarray) -> np.ndarray:
4     """ Windowed Fourier Transform.
5
6     Positional arguments
7     array -- массив всех частей
8     m -- индекс текущей части
9     hem_window_array -- массив окна Хэмминга
10    """
11    def _wft(
12        k: int,
13    ) -> np.imag:
14
15        def window(n: int, m: int):
16            """ Обнуление окна, если индекс выходит за границы массива"""
17            if (
18                n - m >= 0
19                and
20                n - m < hem_window_array.size
21            ):
22                return hem_window_array[n - m]
23            return 0
24
25        def F(item, n):
26            return item * window(n, m) * np.e ** (
27                (-1) * ((2 * np.pi * k * n * 1j) /
28                         array.size - 1)
29            )
30
31        result = np.zeros(array.size, dtype=np.complex_)
32        for idx in range(array.size):
33            result[idx] = F(array[idx], idx)
34        return np.sum(result)
35
36        result = np.zeros(array.shape, dtype=np.complex_)
37        for idx in range(array.size):
38            result[idx] = _wft(idx)
39
40    return result

```

Рис. 6.14. Алгоритм ОКПФ

На основе данных ОКПФ нужно получить спектр фрагментов, для этого реализуем функцию wsp.

```

1 def wsp(array: np.ndarray,
2         window_size: int):
3     """ Windowed Spectrum """
4
5     def _wsp(part_idx):
6         tmp = wft(array, part_idx, hem_window(window_size))
7         tmp = np.abs(tmp)
8         tmp = tmp[tmp.size // 2:]
9         return 2 * tmp / array.size
10
11    result = []
12    for idx in range(array.size):
13        result.append(_wsp(idx))
14    return np.array(result)

```

Рис. 6.15. Алгоритм создания массива спектrogramмы

Из предыдущей работы возьмём функцию сигнала y и для него построим спектrogramмы с разным размером окна Хэмминга, только на этот раз уменьшим количество элементов до 30 (чтобы повышение трудоемкости вычислений не вызывало существенное повышение времени исполнения).

```

1 SAMPLES_COUNT = 30
2 WINDOW_STEP = 5
3 fig, axs = plt.subplots(ncols=2, nrows=(SAMPLES_COUNT // WINDOW_STEP) // 2, figsize=(12, 10))
4 fig.tight_layout(pad=1)
5 for idx, ax in enumerate(axs.ravel()):
6     window_size = (idx + 1) * WINDOW_STEP
7     signal_array = np.zeros(SAMPLES_COUNT)
8     for i in range(1, SAMPLES_COUNT):
9         signal_array[i - 1] = y((2 * np.pi * i) / SAMPLES_COUNT)
10
11    spectrums = wsp(signal_array, window_size)
12    spectrums = spectrums.T
13    x_range = np.arange(start=1, stop=SAMPLES_COUNT - 1)
14    y_bins = np.linspace(start=spectrums.min(), stop=spectrums.max(), num=spectrums.shape[0])
15    y_bins = np.flip(y_bins)
16    y_bins = np.round(y_bins, decimals=2)
17
18    im = ax.imshow(spectrums)
19    ax.set_xticks(np.arange(len(x_range)), labels=x_range)
20    ax.set_yticks(np.arange(len(y_bins)), labels=y_bins)
21    ax.set_title(f"Размер окна Хэмминга: {window_size}")
22 plt.show()

```

Рис. 6.16. Создание спектrogramм для разных размеров окна Хэмминга

Получим следующие графики:

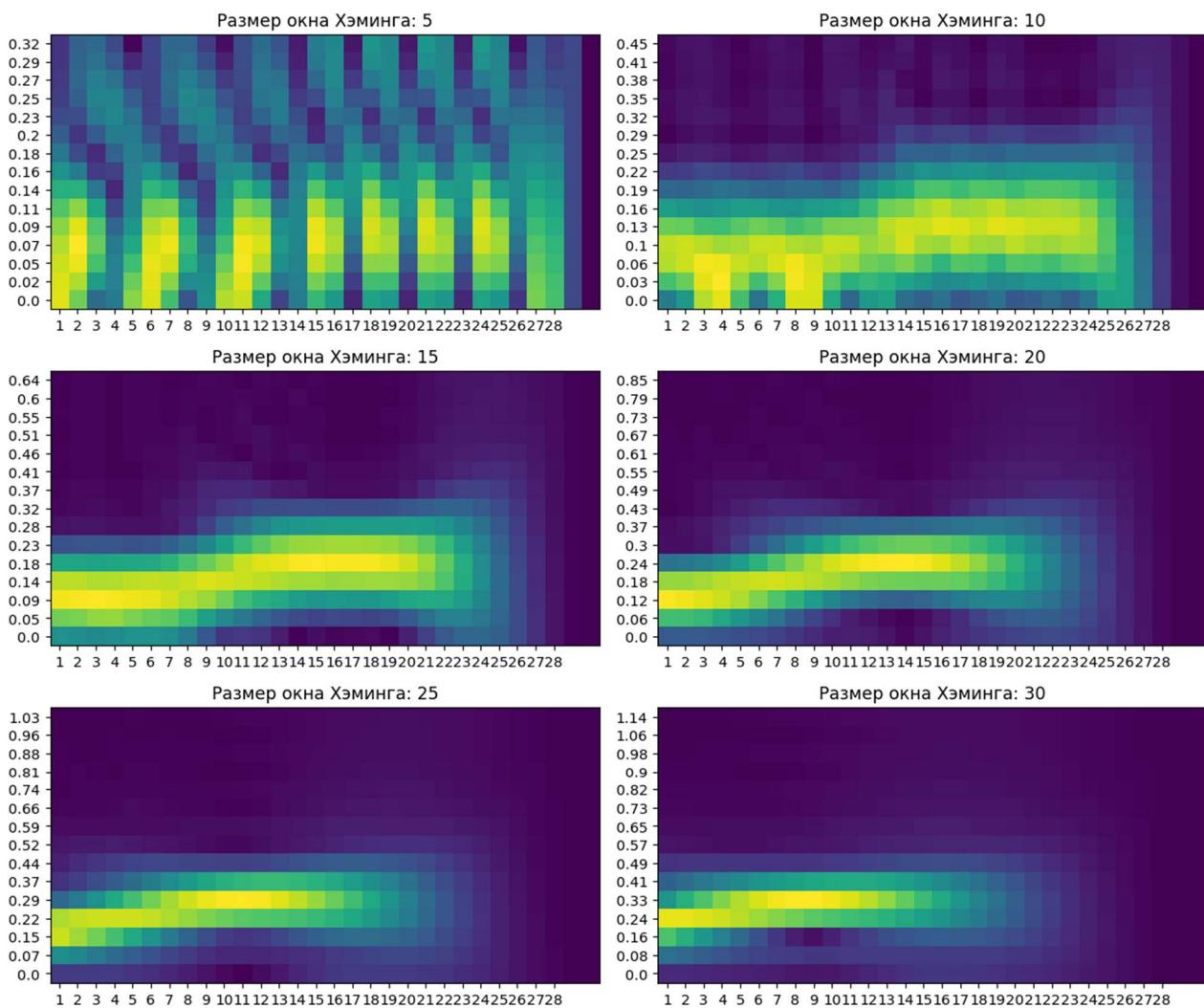


Рис. 6.17. Создание спектрограмм для разных размеров окна Хэмминга

Как раз эти графики и подтверждают, что **чем шире окно, тем лучше разрешение по частоте, но хуже по времени и наоборот.**

Задание

1. Для заданного вариантом сигнала (50 значений) реализуйте КВПФ при трех различных размерах фрагментов (size).
2. Выбрать size, при котором достигается наилучшее разрешение по времени и чистоте.
3. Для заданного сигнала реализуйте ОПФ (30 значений) для трех различных размеров окна (N)
4. Выбрать N, при котором достигается наилучшее разрешение по времени и частоте.
5. Сравнить полученные в пунктах КВПФ и ОКПФ спектрограммы. Привести в конце отчета эти два графика в одну строку. Сделать вывод о простоте визуального интерпретирования частотного состава сигнала (по какому из преобразований легче увидеть переход с одной частоты на другую).

Таблица 6.1

Варианты заданий

Вариант	Сигнал
1,6,11, 16,21,26	$y(x) := \begin{cases} \sin(0,5 \cdot 2 \cdot \pi \cdot x) + \sin(0,8 \cdot 2 \cdot \pi \cdot x) & \text{if } x < \pi \\ \sin(0,8 \cdot 2 \cdot \pi \cdot x) & \text{else} \end{cases}$
2,7,12, 17,22,27	$y(x) := \begin{cases} \sin(0,5 \cdot 2 \cdot \pi \cdot x) & \text{if } x < \pi \\ \sin(0,8 \cdot 2 \cdot \pi \cdot x) + \sin(0,5 \cdot 2 \cdot \pi \cdot x) & \text{else} \end{cases}$
3,8,13, 18,23,28	$y(x) := \begin{cases} \cos(0,5 \cdot 2 \cdot \pi \cdot x) & \text{if } x < \pi \\ \cos(0,8 \cdot 2 \cdot \pi \cdot x) & \text{else} \end{cases}$
4,9,14, 19,24,29	$y(x) := \begin{cases} \cos(0,5 \cdot 2 \cdot \pi \cdot x) + \cos(0,8 \cdot 2 \cdot \pi \cdot x) & \text{if } x < \pi \\ \cos(0,8 \cdot 2 \cdot \pi \cdot x) & \text{else} \end{cases}$
5,10,15, 20,25,30	$y(x) := \begin{cases} \cos(0,5 \cdot 2 \cdot \pi \cdot x) & \text{if } x < \pi \\ \cos(0,8 \cdot 2 \cdot \pi \cdot x) + \cos(0,5 \cdot 2 \cdot \pi \cdot x) & \text{else} \end{cases}$

РАБОТА №7. СВЕРТКА И ДЕКОНВОЛЮЦИЯ

Цель работы: изучение свертки и обратной ей операции на примере простых сигналов и их суммы.

Планируемая продолжительность: от 2 до 4 академических часов.

Тип работы:  с использованием компьютерных средств.

Теоретические основы

Сверткой сигналов называют интеграл произведения двух сигналов, причем один из сигналов сдвигается с опережением относительно другого. То есть, если взаимная корреляция сигналов $f(t)$ и $g(t)$ описывается выражением:

$$(f \times g)(\tau) = \int_{-\infty}^{\infty} \bar{f}(t)g(t - \tau) dt,$$

то свертка может быть описана похожим выражением:

$$(f \times g)(t) = \int_{-\infty}^{\infty} \bar{f}(\tau)g(t - \tau) d\tau.$$

исходя из чего, можно выявить связь взаимной корреляции и свертки:

$$(f \times g)(t) = f(t) \times g(t) = f(-t) * g(t),$$

где $\bar{f}(t)$ и $\bar{f}(\tau)$ - комплексно сопряженные, соответственно, $f(t)$ и $f(\tau)$ сигналы, и в случае используемых нами действительных сигналов можно заменить самими сигналами их комплексно сопряженные копии.

Графически связь ВКФ и свертки представлена ниже. Если оба сигнала являются четными, то их свертка равна их же ВКФ. Упрощенно график свертки двух функций можно интерпретировать также, как и ВКФ – локальная «похожесть» двух функций при соответствующем сдвиге, но, в отличие от ВКФ, при свертке второй сигнал «переворачивается», то есть, для дискретных сигналов, первый элемент свертки определяется как $f_1 \cdot g_1$, (при максимальном отрицательном сдвиге), второй – как $f_1 \cdot g_2 + f_2 \cdot g_1$, третий – как $f_1 \cdot g_3 + f_2 \cdot g_2 + f_3 \cdot g_1$ и т.д.

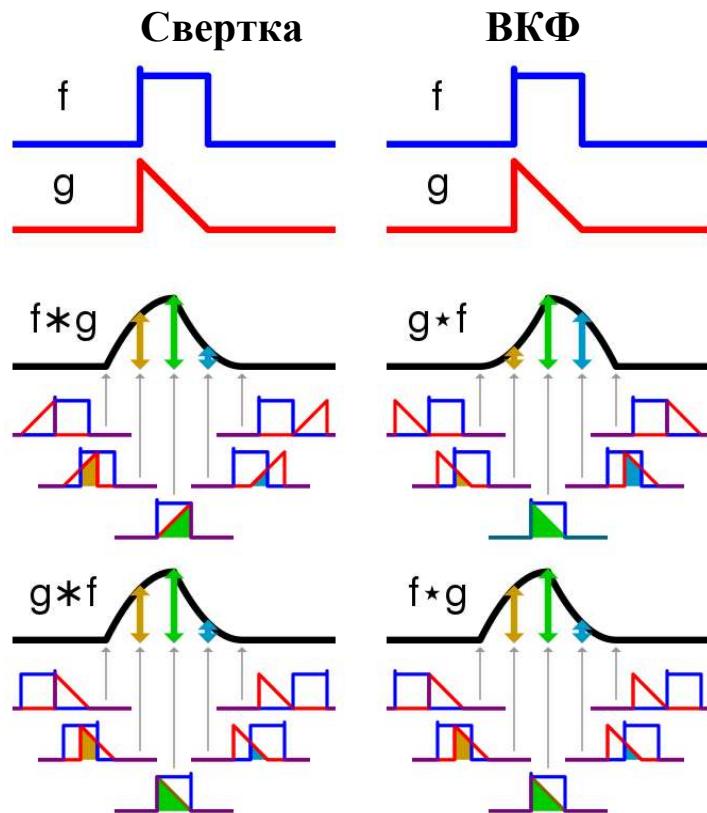


Рис. 7.1. Связь свертки и ВКФ

Часто исследуется случай, когда свертке с одним сигналом фиксированной формы и длительности подвергаются различные по форме и длительности сигналы, при этом фиксированный сигнал, как правило, значительно короче произвольных. В этом случае фиксированный сигнал называют ядром свертки. Также ядром свертки часто называют сдвигаемый сигнал (в выражении свертки), то есть, в текущих обозначениях, $g(t - \tau)$.

Также $g(t - \tau)$ в практических задачах измерительной техники характеризует свойства системы, через которую пропускается сигнал $f(t)$, в результате чего получается искаженная его копия (результат свертки). В этом случае $g(t - \tau)$ называют **импульсной характеристикой** данной системы (причем это может быть как некоторая измерительная система или блок, так и сама среда или канал связи, искажающие воздействия которых и задаются импульсной характеристикой).

Если f и g – дискретные сигналы, свертку для них можно, в общем случае записать выражением:

$$(f * g)_k = \sum_{m=1}^M f_m \cdot g_{k-m}, \quad (9.1)$$

где m – сдвиг;

k – номер элемента в массиве сигнала f и в массиве свертки;

M – число элементов в сигнале M .

Очевидно, что в сигнале g по выражению (6.1) можно ссылаться на несуществующие элементы. Для исключения вызванных этими ошибок следует изменить (6.1):

$$(f * g)_k = \sum_{m=k-N}^{k-1} f_m \cdot g_{k-m}, \quad (9.2)$$

а также проверять выход за границы f :

$$f_m = 0, \text{ если } (m < 1) \vee (m > M),$$

где N – число элементов в массиве g .

Результат свертки будет иметь размер $K=M+N-1$, так как g будет пересекаться с f левым краем при k , близких к M , поэтому (9.2) может не быть равным нулю при $k>M$.

Свертка в Python

Импортируем следующие зависимости.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4 from random import randint
5 %matplotlib widget
6 # from google.colab import output
7 # output.enable_custom_widget_manager()
```

Рис. 7.2. Импортирование зависимостей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

Также из предыдущей работы импортируем функцию plot.

```
1 def plot(*args):
2     ax = plt.figure()
3     for y in args:
4         x = np.arange(y.size)
5         plt.plot(x, y)
6     plt.grid(True)
7     plt.ylabel('y')
8     plt.xlabel('x')
9     plt.show()
```

Рис. 7.3. Функция отрисовки графиков

Чтобы задать выражение для свертки двух дискретных сигналов, можно прибегнуть к алгоритму, похожему на задание ВКФ.

```
1 def convolve_element(
2     f_array: np.ndarray,
3     g_array: np.ndarray,
4     k: int) -> np.ndarray:
5     """ Свёртка элемента """
6
7     def get_f(idx: int):
8         """
9             Получаем значение f
10            Если индекс выходит за пределы массива, то возвращаем 0
11        """
12
13         if idx >= 0 and idx < f_array.size:
14             return f_array[idx]
15         return 0
16
17     result = np.zeros(g_array.size)
18     for idx in range(k - g_array.size, k):
19         result[g_array.size - (k - idx)] = get_f(idx) * g_array[k - idx - 1]
20
21     return np.sum(result)
22
23
24 def convolve(
25     array_1: np.ndarray,
26     array_2: np.ndarray
27 ) -> np.ndarray:
28     """ Свёртка двух массивов """
29     result = np.zeros(array_1.size + array_2.size)
30     for idx in range(1, result.size):
31         result[idx] = convolve_element(array_1, array_2, idx)
32
33     return result
```

Рис. 7.4. Алгоритм свёртки двух массивов

При этом в данной работе отдельно задавать массив сдвигов не требуется, так как все значения сдвига – положительные целые числа, отдельный массив является избыточной величиной, достаточно просто верно задать итератор в цикле `for`.

Зададим два сигнала, построим их графики и графики свертки (скобку на подписи графика для вставки двух массивов можно найти на панели Функции, в самом конце). Стоит обратить внимание, что сигналы могут отличаться по размеру (что чаще всего наблюдается в рамках практических задач).

```
1 y_1 = np.array([1., 1.2, 0., 1.1, 1.3, 0, 1.1, 0])
2 x_1 = np.arange(stop=y_1.size)
3 y_2 = np.array([0.5, 2, 0.5])
4 x_2 = np.arange(stop=y_2.size)
5 plot(y_1, y_2)
```

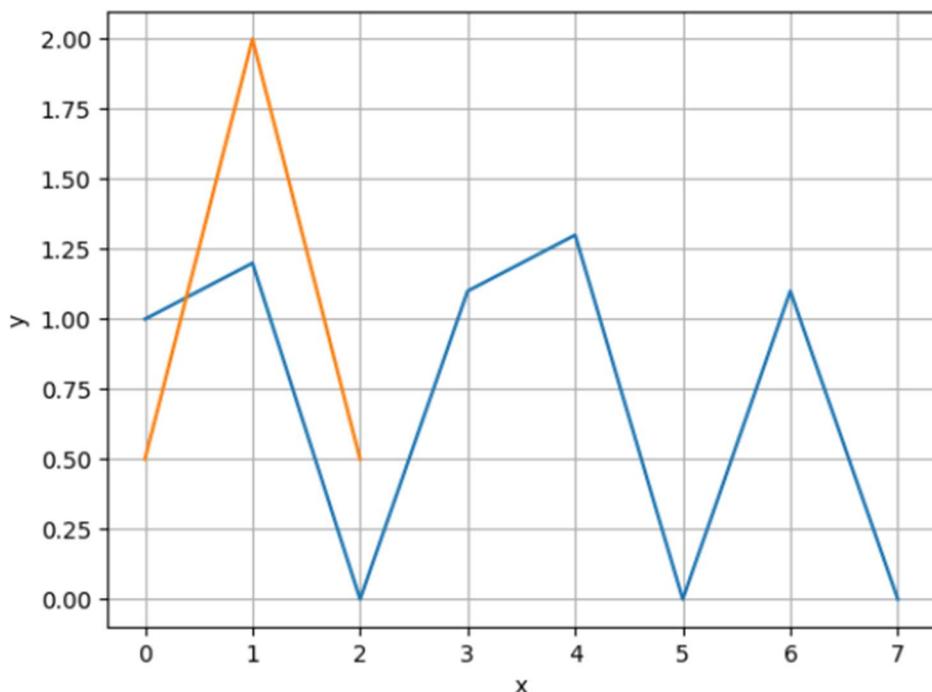


Рис. 7.5. Сигналы

```
1 convolved = convolve(y_1, y_2)
2 expected = np.zeros(convolved.size)
3 expected[1:] = signal.convolve(y_1, y_2, mode='full')
4 np.testing.assert_array_equal(convolved, expected)
5 plot(convolved)
```

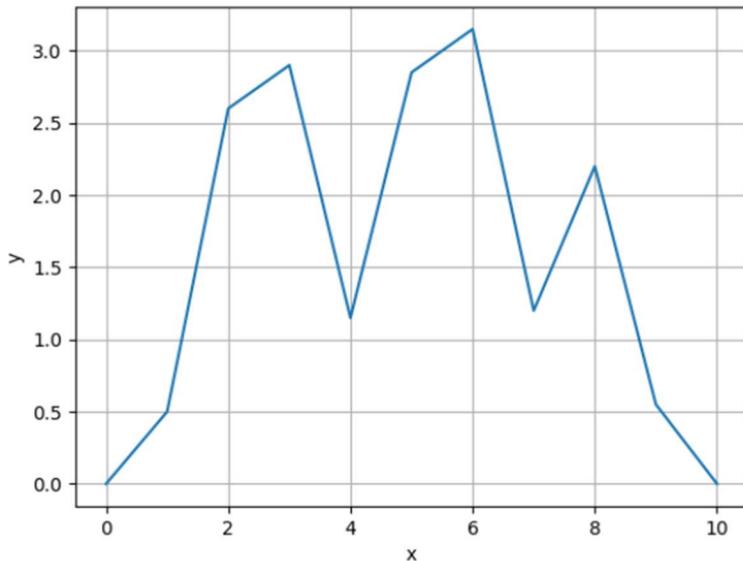


Рис. 7.6. Результат свёртки

Слева синим цветом показан исходный сигнал, жёлтым – функция, с которой он сворачивается. Такая процедура часто имеет место в измерительной технике – многие приборы обладают собственной импульсной характеристикой (например, фильтры), и при прохождении через них сигналов происходит свертка сигналов с импульсной характеристикой. В данном случае исходный сигнал задан тремя простыми импульсами, пример, который может быть использован в качестве импульсной характеристики – треугольным одиночным импульсом.

Деконволюция в частотной области

Если продолжить рассматривать свертку в данной работе, как результат взаимодействия некоторого сигнала и импульсной характеристики прибора, следует считать, что сдвигу подвергается импульсная характеристика, поскольку она на практике практически во всех случаях меньше по длительности, чем сам сигнал. А из выражений выше видно, что размер массива свертки определяется как $2(\text{rows}(ar1)-1)$, поэтому для сигналов разной длины использование в качестве ar1 самого короткого из них даст неполный результат.

Следовательно, считая неподвижным исходный сигнал, получаем график, на котором еще различимы (но очень слабо) три исходных импульса.

Задача деконволюции – обратная свертке задача. В сущности, в русскоязычной литературе данный термин часто встречается именно как обратная свертка (реже развертка). Задача деконволюции в измерительной технике – восстановить исходный сигнал, искаженный прибором, а именно его импульсной характеристикой, которая должна быть известна (бывают и иные случаи, которые здесь не рассматриваются).

Восстановить исходный сигнал можно, если прибегнуть к преобразованию Фурье, поскольку известно следующее свойство свертки:

$$F[f(t) * g(t)] = F[f(t)] \cdot F[g(t)],$$

где $F[]$ – преобразование Фурье.

То есть, получить исходный сигнал $f(t)$ можно, выразив его из выражения выше:

$$f(t) = F^{-1} \left[\frac{F[f(t) * g(t)]}{F[g(t)]} \right],$$

где $F^{-1}[]$ – обратное преобразование Фурье.

Реализация деконволюции в Python

Воспользуемся библиотекой SciPy для свёртки сигнала, так как данный пакет учитывает особенности работы NumPy. При реализации напрямую выражений из вышеописанных функций, результатом данных функций будут числа выходящий за пределы разрядности чисел, что естественно будет ошибкой.

Создадим массив для свёртки и импульсную характеристику. Функцией `signal.convolve` произведём свёртку, и выведем полученные данные на графике.

```
1 original = np.array([0, 1, 0, 0, 1, 1, 0, 0])
2 impulse_response = np.array([2, 1])
3 convolved = signal.convolve(impulse_response, original)
4 plot(original)
5 plot(convolved)
```

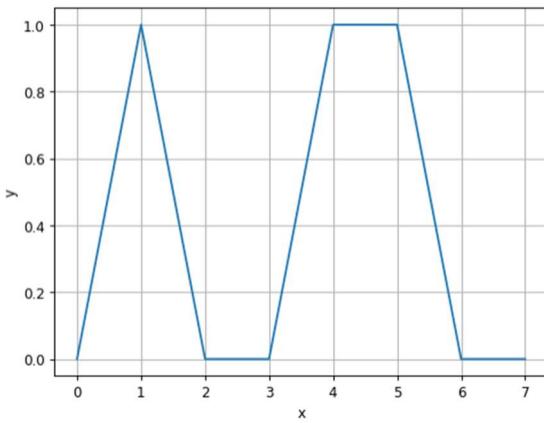


Figure 27

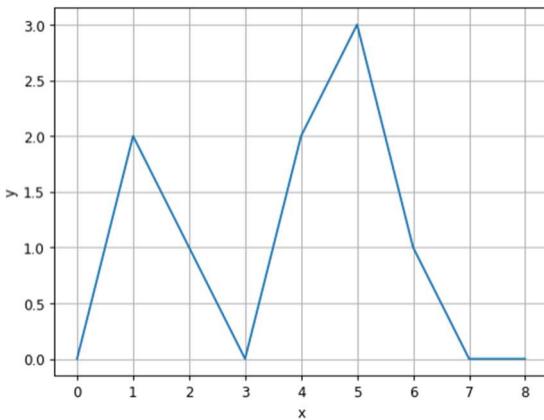


Рис. 7.7. Свёртка сигнала

Важно учитывать, что импульсная характеристика не должна быть существенно более длительная, чем отдельные компоненты сигнала – иначе детали теряются полностью и данный способ восстановить исходный сигнал не позволяет (то есть если имеются проблемы в работе описанного алгоритма – в первую очередь смотрите на длительность второго сигнала; значения равные нулям справа и слева от отличной от нуля области здесь не учитываются).

Деконволюция происходит также функцией `deconvolve` из того же модуля.

```
1 recovered, _ = signal.deconvolve(convolved, impulse_response)
2 plot(recovered)
```

Рис. 7.8. Деконволюция сигнала

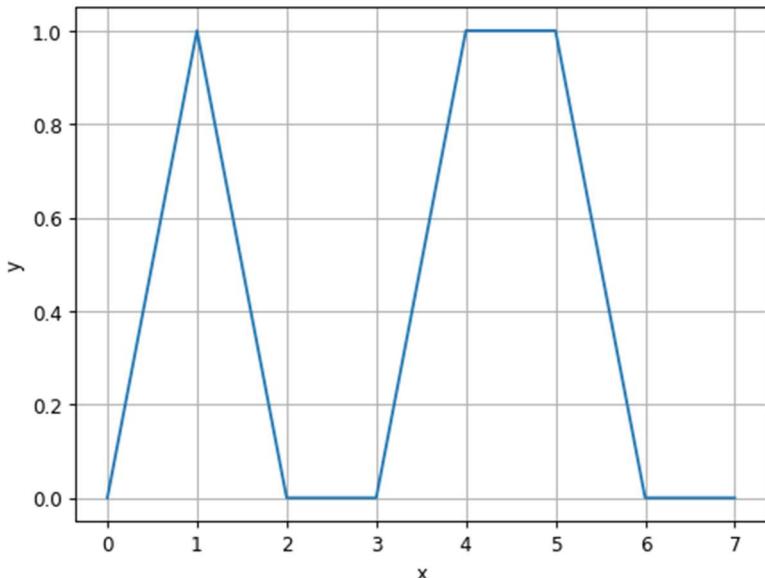


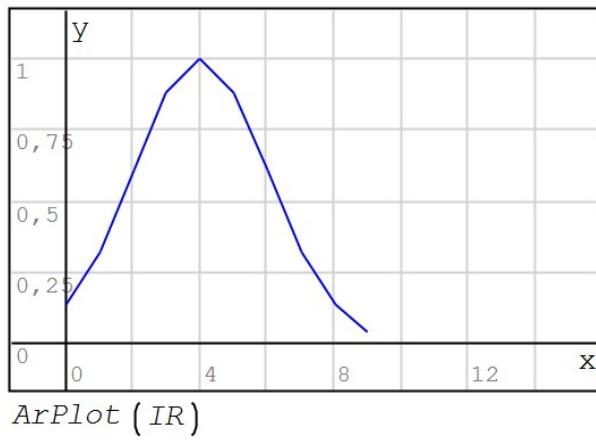
Рис. 7.9. График деконволированного сигнала

Свертка сигнала с функцией Гаусса

Теоретические основы

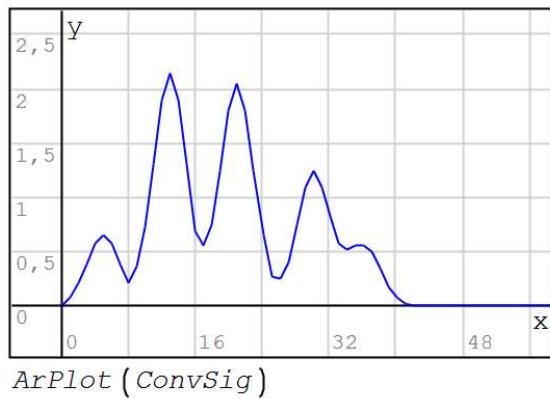
Перейдем к примеру, который куда ближе к практике. Имеется прибор с импульсной характеристикой, заданной функцией $y(x)$.

Эта функция, $y(x)$, называется кривой Гаусса и в данном контексте может говорить о том, что прибор понижает разрешение выходного сигнала, по сравнению с входным, «смазывает» его. Такое можно наблюдать, например, при фотографировании с дефокусом – расфокусировка может быть устранена деконволовацией с использованием функции Гаусса в качестве импульсной характеристики. Также фильтры, размывающие фон на фото в режиме портрета в камере смартфона, а также фильтр Размытие по Гауссу в Adobe Photoshop выполняют именно данную процедуру – свертку исходного изображения или его частей с функцией Гаусса.



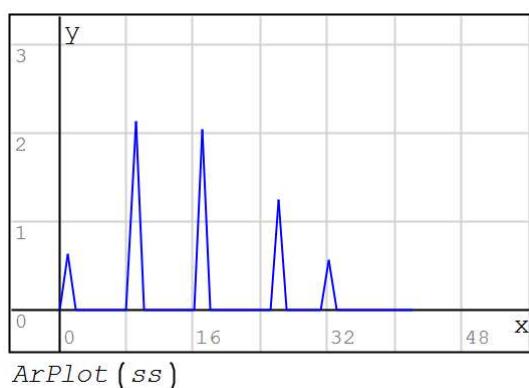
Rис. 7.10. Функция Гаусса

В измерительной технике такая импульсная характеристика свойственна многим приборам и связана с их инерционностью. К примеру, в спектрометрии можно встретить спектр вида, представленного на рисунке.



Rис. 7.11. Свернутый спектр

Тогда как действительный (или истинный) спектр имеет вид, показанный ниже.



Rис. 7.12. Действительный спектр

Можно заметить, что помимо растяжения и сглаживания четких одиночных пиков, свертка с импульсной характеристикой вызывает смещение максимальных значений пиков вправо. Если представленные выше графики – спектры частот, то смещение положения пиков по оси абсцисс вызывает погрешность в определении частоты исследуемого процесса, что куда серьезнее погрешности в определении амплитуды.

Пример реализации

В качестве примера зададим сигнал, подобный приведенному выше, для чего сперва спроектируем функцию, генерирующую массив, в котором содержится заранее заданное число пиков со случайными параметрами. Зададим сигнал по этой функции и выведем его.

```
1 def form_signal(n: int) -> np.ndarray:
2     tmp = [0]
3     for _ in range(n):
4         tmp.append(0.25 + randint(0, 100) / 50)
5         for _ in range(10 - randint(0, 8)):
6             tmp.append(0)
7     return np.array(tmp)

1 signal_s = form_signal(5)
2 plot(signal_s)
```

Рис. 7.13. Задание сигнала

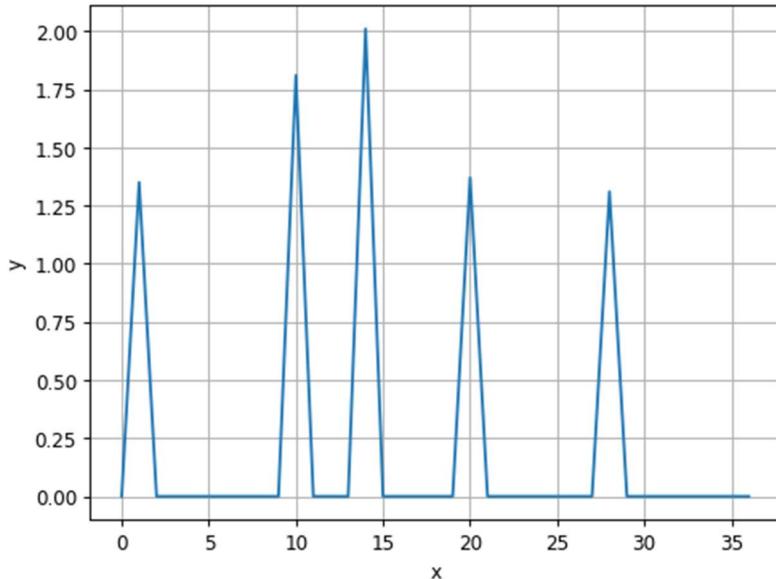


Рис. 7.14. График сигнала

Для ss вычислим его свертку с импульсной характеристикой по функции Гаусса. Построим график свертки (при пересчете будет новый график из-за случайности параметров пиков).

```

1 def y(x, w):
2
3     def numerator():
4         degree = (-1) * (x ** 2) / (2 * (w ** 2))
5         return np.power(np.e, degree)
6
7     def denominator():
8         return np.sqrt(2 * np.pi) * w
9
10    return numerator() / denominator()

1 N = 10
2 DX = 1 / N
3 x = -0.5 + DX * np.arange(start=1, stop=N + 1)
4 IR = y(x, 1 / 5)
5 convolved_signal = signal.convolve(signal_s, IR, mode='full', method='direct')
6 plot(convolved_signal)
```

Рис. 7.15. Алгоритм свёртки функцией Гаусса

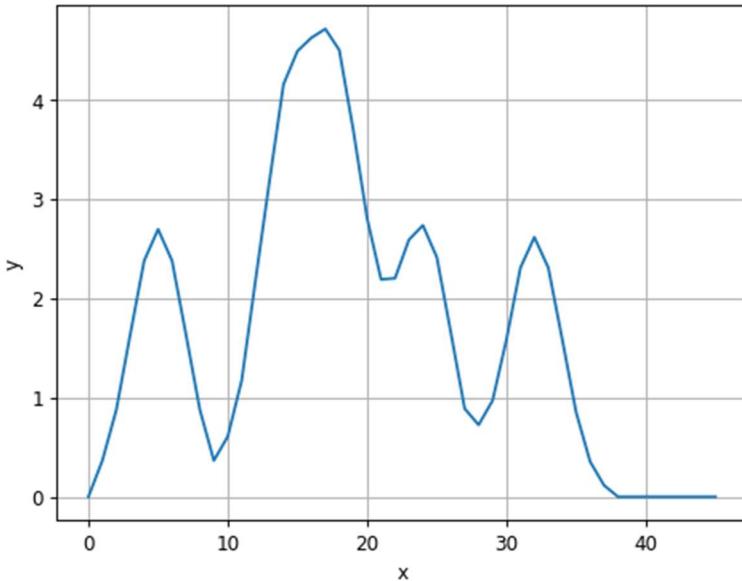


Рис. 7.16. График свёрнутого сигнала

Теперь, поскольку параметры импульсной характеристики заранее известны, вычислить деконволюированный сигнал по приведенному ранее алгоритму не составит труда. Здесь мы вместо IR вызываем функцию `y()` с известным заранее параметром `w` (`1/5`) – сейчас процедура выполняется для примера, в дальнейшем же понадобится для реализации последнего раздела и выполнения соответствующего задания.

```
1 deconvolved_signal, _ = signal.deconvolve(convolved_signal, y(x, 1 / 5))
2 plot(np.array(deconvolved_signal))
```

Рис. 7.17. Деконволовия и вывод графика результата

Построим графики результата деконволовии и подобранный импульсной характеристики. Как видим, график сигнала больше не имеет «сглаженностей», содержит только короткие одиночные импульсы.

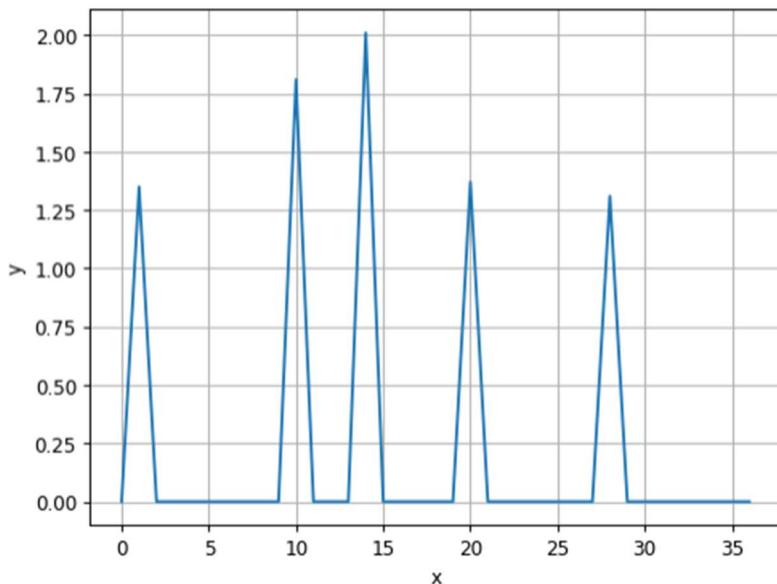


Рис. 7.18. График деконволированного сигнала

Деконволюция сигнала с использованием априорной информации о форме импульсной характеристики и последовательным ручным уточнением ее параметров

Теоретические основы

Для случаев, когда импульсная характеристика заранее неизвестна, ее пытаются оценить, обычно статистическими методами. Для некоторых случаев, например, когда известна функция, задающая импульсную характеристику, но не известны ее параметры.

Чем ближе ширина пика к требуемому значению, тем ближе деконволированный сигнал к исходной форме, в случае, если сигнал состоит из отдельных резких, преимущественно положительных пиков, на графике деконволюции будет все меньше искажений, особенно в отрицательной области.

Такой результат может быть получен, если ширина импульсной характеристики выбрана больше требуемой (справа – деконволюция, слева - исходный).

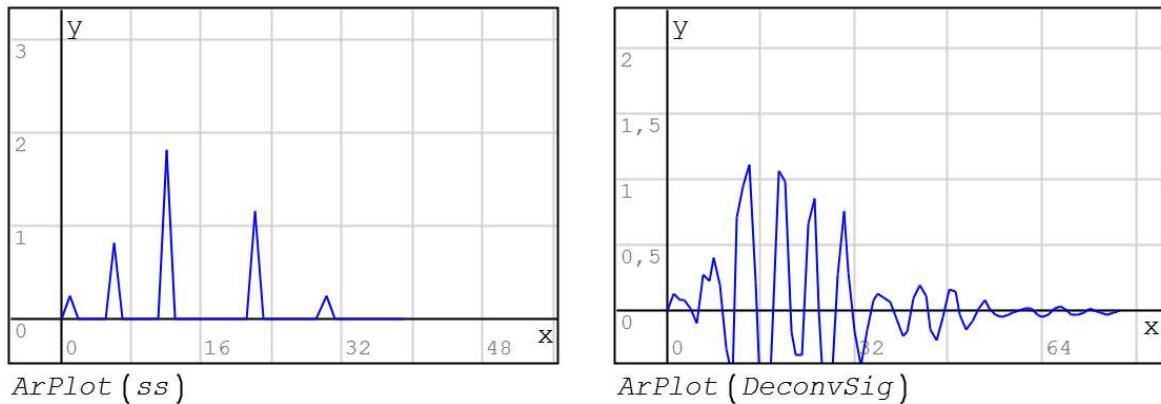


Рис. 7.19. Результат при превышении максимально возможной ширины

Результат ниже – если ширина меньше требуемой.

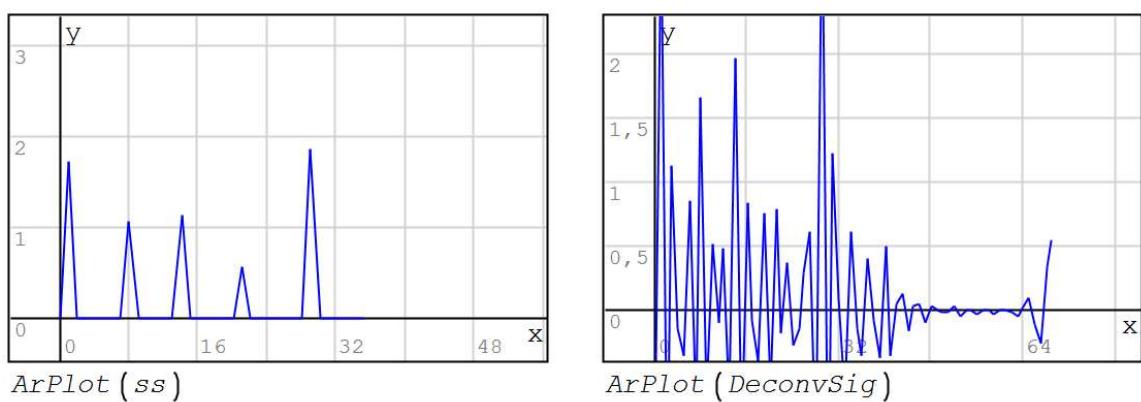


Рис. 7.20. Результат при занижении минимальной ширины

Графики ниже характеризуют близкое к требуемому значение, при его небольшом превышении.

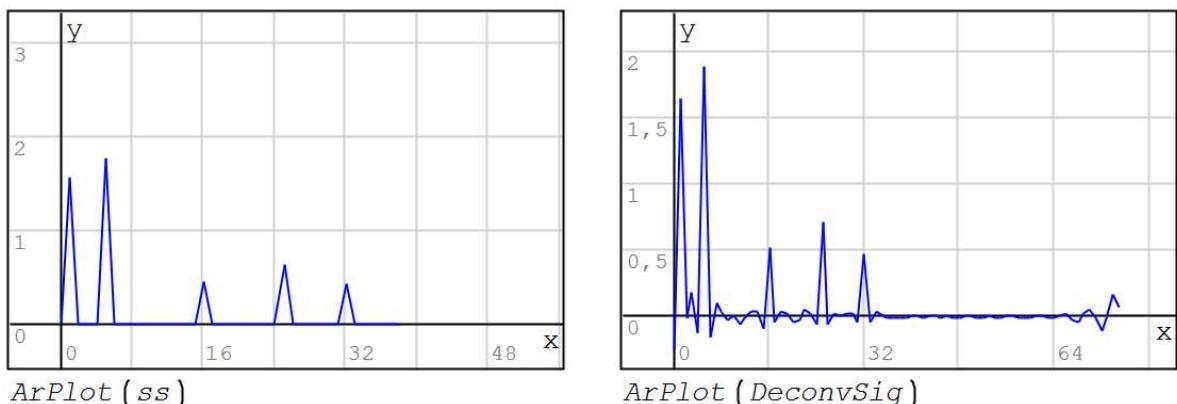


Рис. 7.21. Приближение с большей стороны

Графики ниже характеризуют близкое к требуемому значение, при его небольшом занижении.

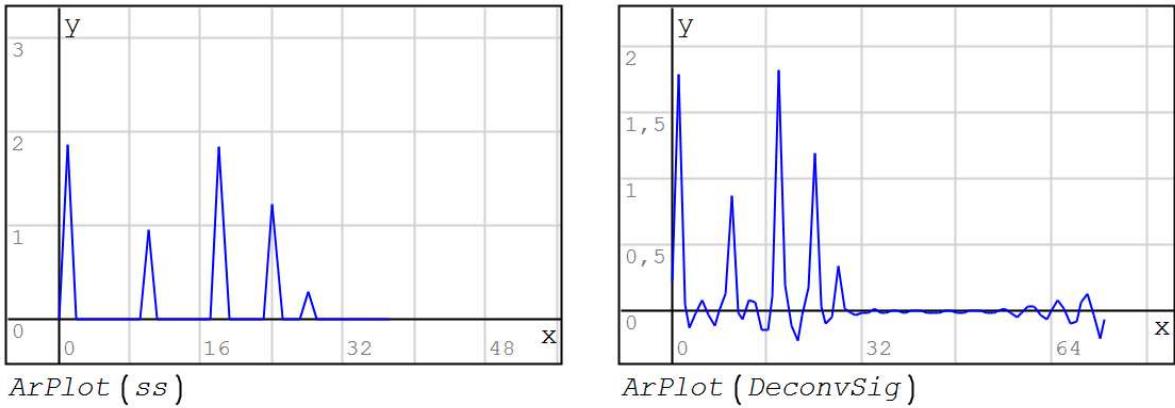


Рис. 7.22. Приближение с меньшей стороны

Таким образом, деконволютировать некоторые простые сигналы можно даже вручную – если известна форма импульсной характеристики прибора, но неизвестны ее параметры.

Пример реализации

Рассмотрим нередкий случай, когда известна в общем виде форма импульсная характеристики, но не ее параметры. Допустим, импульсная характеристика задана функцией Гаусса неизвестной ширины. При этом сигнал, являющийся результатом конволюции исходного сигнала с этой функцией, представлен ниже.

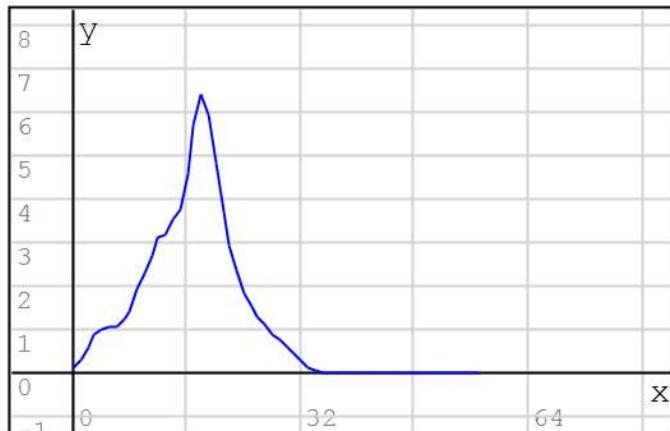


Рис. 7.23. Результат свертки

Для решения задачи воспользуемся уже заданными в разделе «Свертка сигнала с функцией Гаусса». В `resp` заменим параметр ширины функции `y()` с $1/5$ на 1 . Применим спроектированный ранее алгоритм деконволюции к исследуемому сигналу.

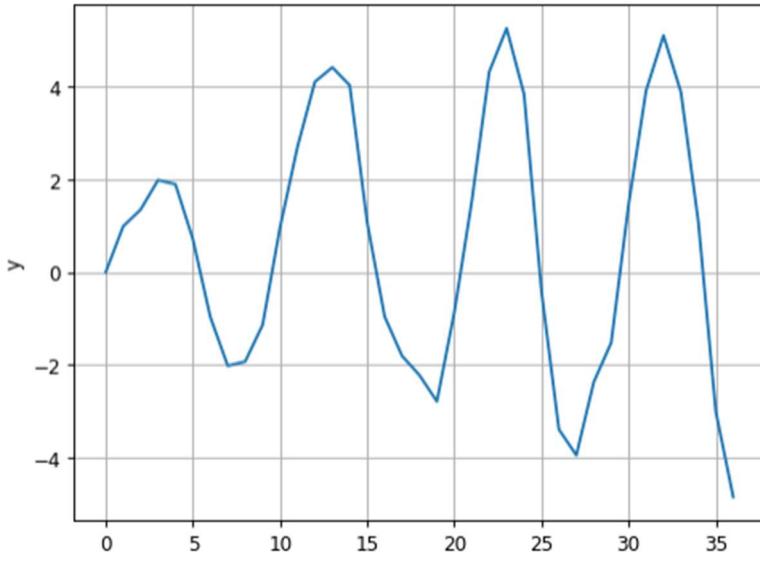


Рис. 7.24. Первичная деконволюция

Видим практически синусоиду на выходе. Учитывая, что входной сигнал был строго положительным после свертки, можно быть уверенным, что до свертки с функцией Гаусса он также был исключительно положительным, а, значит, полученный результат является следствием неверно выбранного параметра ширины импульсной характеристики.

Уменьшим ширину импульсной характеристики вдвое, для чего в выражении resp сменим внутри функции $y()$ параметр w с 1 на $1/2$. Полученная импульсная характеристика и результат деконволюции ниже.

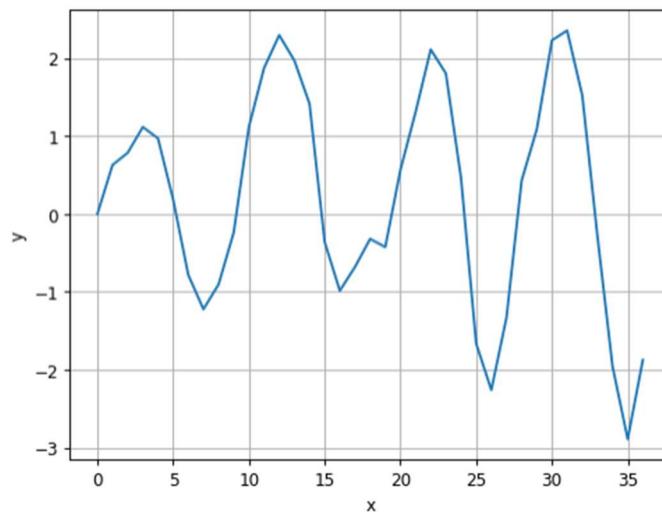


Рис. 7.25. Уточнение деконволюции

Видно, что на графике деконволюции появляются непериодические составляющие, что может указывать на приближение к истинной форме исходного сигнала.

Изменим параметр w на $1/4$.

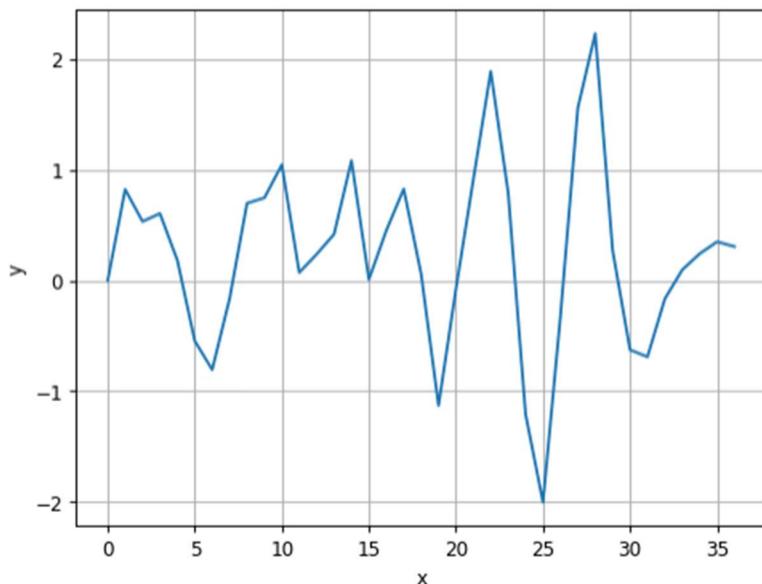


Рис. 7.26. Уточнение деконволюции

Как видим, непериодические детали проявляются все сильнее. Попробуем принять параметр w равный $1/5,5$.

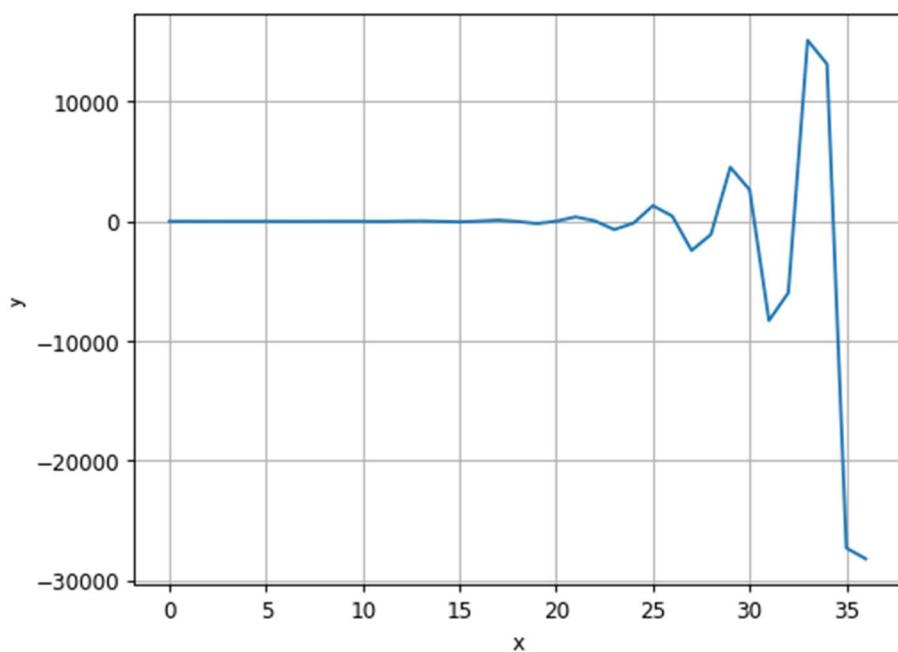


Рис. 7.27. Уточнение деконволюции

В данном случае всплеск в конце указывает, на то, что происходит отдаление от истинной формы сигнала. Этот всплеск является высокочастотной составляющей.

Попробуем уменьшить параметр w до $1/6$, чтобы уточнить результаты.

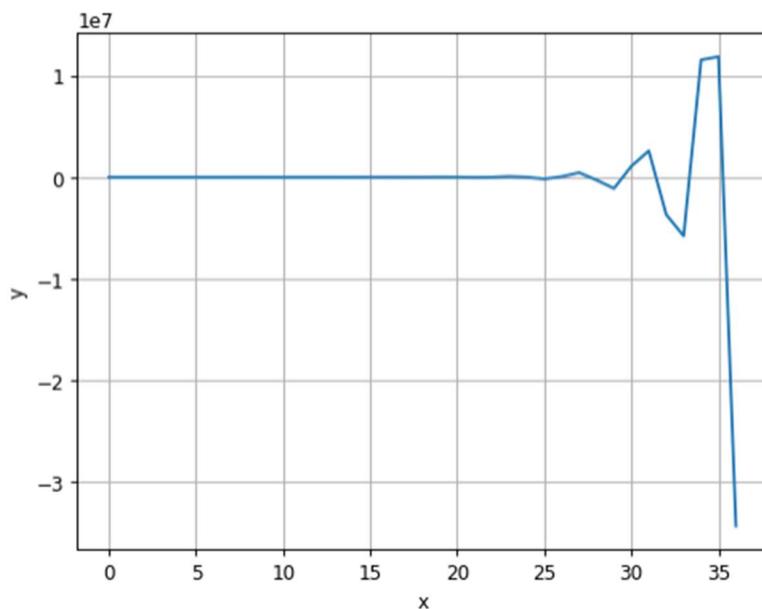


Рис. 7.28. Уточнение деконволюции

Высокочастотная переменная составляющая нарастает, значит, определено требуемый множитель ширины был пропущен. Исходя из анализа приведенных графиков, можно видеть, что исходный сигнал был подвергнут свертке с функцией Гаусса с параметром w в диапазоне $1/5,5\dots1/4$.

Дальнейшее уточнение результата представлено ниже.

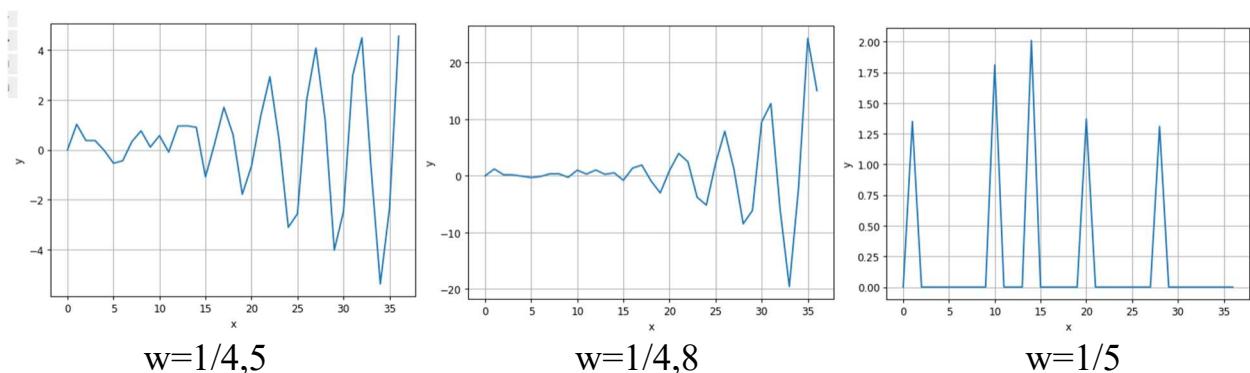


Рис. 7.29. Последовательное уточнение

Таким образом, исходный сигнал представлен на последнем графике, а подобранная импульсная характеристика описывается выражением $\frac{e^{-\frac{x^2}{2(\frac{1}{5})^2}}}{\sqrt{2\pi}(\frac{1}{5})}$ или, упрощенно $\frac{5e^{-\frac{25x^2}{2}}}{\sqrt{2\pi}}$.

Задание

1. Реализовать пример для вычисления свертки и деконволюции (второй и четвертый разделы).
2. Опробовать свертку и деконволюцию для заданных сигнала и импульсной характеристики (таблица 9.1), заменив указанным сигналом тот, что использовался в задании 1.
3. Реализовать пример свертки сигнала с функцией Гаусса и его деконволюции по пятому разделу.
4. Задать вместо сигнала ss в задании 3 тот, что определен вариантом (таблица 9.2).
5. Решить задачу определения параметров импульсной характеристики (функции Гаусса) по примеру в последнем разделе описания данной работы. При описании приводить все промежуточные результаты, обосновывать выбор того или иного множителя от этапа к этапу. Начальное приближение w брать равным 1. Менять только w (все по примеру в последнем разделе), причем:
 6. искомое значение w не может быть менее 1/10 и более 1;
 7. искомое значение w во всех вариантах представляет собой дробь вида $\frac{1}{c}$, где c – некоторое целое или дробное число;
 8. c может иметь не более одного знака после запятой (к примеру, допускаются $c = 5,1$ и $w = \frac{1}{5,1}$, $c = 7$ и $w = \frac{1}{7}$, $c = 4,4$ и $w = \frac{1}{4,4}$ и т.п.).

Таблица 9.1

Массивы для второго задания

Вариант	Сигнал	Импульсная характеристика
1,6,11, 16,21,26	0.3,1.2,4,2.1,3.3,1.2,1.7,0	2,0,1
2,7,12, 17,22,27	0,1,0,2.5,2.6,1.1,0.7,0	1,0,0
3,8,13, 18,23,28	3,2.2,4,2.7,0.3,4.2,2.6,1.4	0,1,0
4,9,14, 19,24,29	1,1,1,3,1.7,2.1,3.7,1	0.5,1,0
5,10,15, 20,25,30	1.4,0.2,1,1.5,1.6,1.7,0.4,1	0.5,1,0

Таблица 9.2

Массивы для четвертого задания

Вариант	Сигнал
1,6,11, 16,21,26	0,1.4678,2.1005,2.7133,3.1638,3.7576,3.7757,3.5037,3.0221,2.4378,1.847 5,1.6234,1.804,1.9676,2.0654,1.89,1.7957,3.3781,3.8225,4.2308,4.4874, 4.17,5.1476,5.0946,4.8222,4.3938,3.8494,2.5557,2.1918,1.6968,1.1857, 0.7479
2,7,12, 17,22,27	0.1435,0.3109,0.54,0.7522,0.84,0.7522,0.54,0.6269,0.8281,1.2424,1.6565, 1.85,1.6565,1.1893,0.6846,0.316,0.1169,0.0427,0.0925,0.1607,0.2239, 0.25,0.4852,0.7269,1.0761,1.4127,1.5458,1.37,0.9836,0.5662,0.4749, 0.5593,0.8036,1.1193,1.25,1.1193,0.8036,0.4626,0.591,0.8969,1.4208, 1.9789,2.21,1.9789,1.4208,0.8179,0.3775,0.1397
3,8,13, 18,23,28	0.6302,0.8292,1.0088,1.1346,1.18,1.1346,2.3173,2.5509,2.7247,2.7987, 2.45,2.3558,2.0944,1.7217,1.3085,2.3989,1.9465,2.368,2.6635,2.77, 2.6635,2.368,1.9465,1.4794,1.0396,0.9667,1.2719,1.5473,1.7404,1.81, 2.3653,2.3695,2.2721,2.0917,1.8493,2.2413,2.4689,2.6089,2.6345,2.5291, 2.0097,1.7867,1.4687,1.1162,0.7844
4,9,14, 19,24,29	0.0358,0.1073,0.235,0.3761,0.44,0.3761,0.235,0.1073,0.0358,0.0584, 0.1487,0.3258,0.5215,0.61,0.5215,0.3258,0.1487,0.0496,0.2049,0.5779, 1.2658,2.0261,2.37,2.0261,1.2658,0.5779,0.1928,0.047,0,0.3002,0.8998, 1.9708,3.1545,3.69,3.1545,1.9708,0.8998,0.3002,0.2823,0.6267,1.3726, 2.197,2.57,2.197,1.3726,0.7935,0.709,1.1459,1.7525,2.05,1.7525,1.0949, 0.4999,0.1668,0.0407
5,10,15, 20,25,30	0.0632,0.2324,0.5891,1.0295,1.24,1.0295,0.5891,0.2324,0.0632,0.1836, 0.6316,1.6011,2.7979,3.37,2.7979,1.6011,0.6316,0.1717,0.0322,0.1901, 0.699,1.7722,3.0968,3.73,3.1278,1.8865,0.9889,0.6965,0.6456,0.5064, 0.2898,0.1143,0.1233,0.345,0.86,1.5027,1.81,1.5786,1.1392,1.0471, 1.3293,1.5073,1.237,0.7079,0.2792,0.0759,0.0142

РАБОТА №8. ПОСТРОЕНИЕ НЕРЕКУРСИВНЫХ ЦИФРОВЫХ ФИЛЬТРОВ МЕТОДОМ ЧАСТОТНОЙ ВЫБОРКИ И ИХ ПРИМЕНЕНИЕ

Цель работы: получение навыков проектирования и применения нерекурсивных цифровых фильтров.

Планируемая продолжительность: от 2 до 4 академических часов.

Тип работы:  с использованием компьютерных средств.

Теоретические основы

Цифровым фильтром называют математическое выражение, применив к некоторому сигналу которое можно получить копию данного сигнала, в которой подавлена та или иная область частот, причем и сигнал, и его копия являются цифровыми величинами (массивы дискретных значений). Такие фильтры можно реализовать и аппаратно, так же, как и аналоговые, но из-за растущей сложности структуры с увеличением порядка чаще прибегают к программной реализации фильтра, и уже программу записывают в некое вычислительное устройство, обрабатывающее входной сигнал.

Цифровые фильтры могут реализовывать те же выражения, что и аналоговые (Баттервортса, Кауэра, Бесселя и т.п.), но, вместе с тем, позволяют получить и фильтры, передаточная характеристика которых задана произвольно (буквально можно вручную по точкам задать идеальную для конкретного случая характеристику и получить выражение для ее реализации).

Цифровые фильтры в самой широкой классификации принято делить на рекурсивные (используют обратную связь) и нерекурсивные (не используют обратную связь). Их также называют БИХ (с бесконечной импульсной характеристикой) и КИХ (с конечной импульсной характеристикой) фильтрами.

В рамках данной работы рассмотрим нерекурсивные цифровые фильтры. Основной характеристикой таких фильтров является импульсная характеристика:

$$h_k = [h_0, h_1, \dots, h_{M-1}],$$

где $[]$ – обозначение массива дискретных значений;

M – число точек в импульсной характеристике.

Применить фильтр с импульсной характеристикой h_k к сигналу s из N точек можно, воспользовавшись следующим алгоритмом работы фильтра:

$$y_m = \sum_{i=0}^{N+2(M-1)} s_i \cdot h_{m-i},$$

который, фактически, является сверткой импульсной характеристики фильтра и входного сигнала.

Проектирование нерекурсивного фильтра в Smath Studio

Для реализации нерекурсивного ЦФ в Smath Studio сперва вновь зададим зависимости и изменим график построения функции из предыдущей работы.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4 from scipy.fft import fftshift, fft, fftfreq
5 from random import randint
6 %matplotlib widget
7 # from google.colab import output
8 # output.enable_custom_widget_manager()
```

Рис. 8.1. Импорт зависимостей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

```

1 def plot(*args, y=None, stem=False):
2     ax = plt.figure()
3     if y is None:
4         for y in args:
5             x = np.arange(y.size)
6             if stem:
7                 plt.stem(x, y)
8             else:
9                 plt.plot(x, y)
10    else:
11        if stem:
12            plt.stem(args[0], y)
13        else:
14            plt.plot(args[0], y)
15    plt.grid(True)
16    plt.ylabel('y')
17    plt.xlabel('x')
18    plt.show()

```

Рис. 8.2. Функция построения графиков

Так как в данной работе она используется для фильтрации, назовем ее filtr, в отличие от прошлой работы.

```

1 def filtr(array_1: np.ndarray,
2           array_2: np.ndarray) -> np.ndarray:
3     return signal.convolve(array_1, array_2, mode="full")

```

Рис.8.3. Задание свертки для фильтрации

Рассмотрим работу фильтра на примере. Зададим сигнал, состоящий из двух частот.

```

1 def y(x: np.ndarray) -> np.ndarray:
2     return 10 * np.sin(9 * 2 * np.pi * x) + 5 * np.sin(2 * 2 * np.pi * x)
3
4
5 N = 33
6 DX = 1 / N
7 x = np.arange(start=1, stop=N)
8 sig = y(x * DX)
9 plot(sig)

```

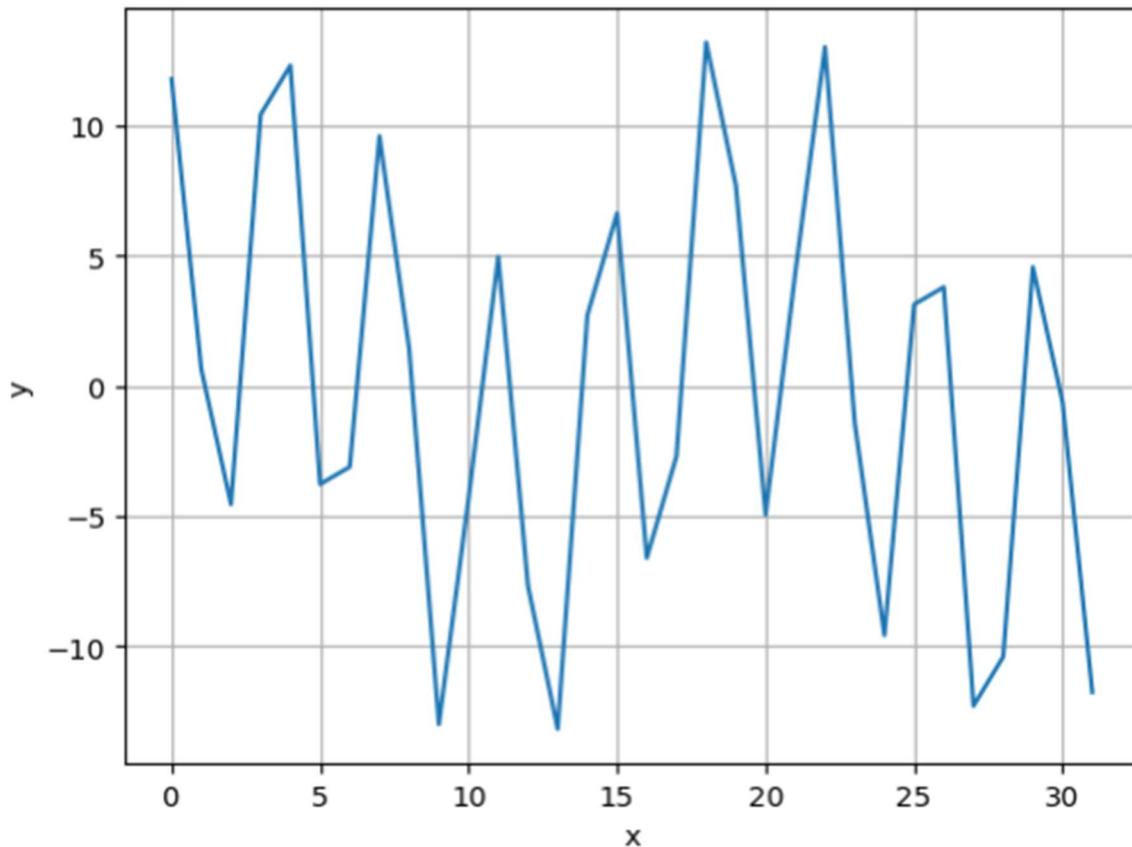


Рис. 8.4. Тестовый сигнал

А также фильтр с импульсной характеристикой, приведенной ниже.

```

1 IR = np.array([0.4366, 0.3943, 0.1416, -0.0793, -0.0924,
2                 0.0327, 0.0889, -0.0068, -0.1109, -0.0311, 0.2264])
3 plot(IR)

```

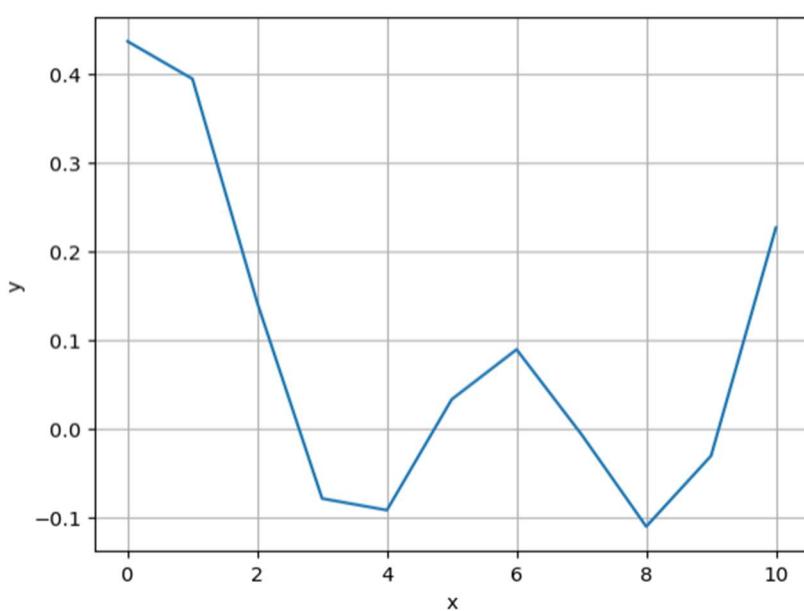


Рис. 8.5. Импульсная характеристика

Теперь построим график сигнала до и после фильтрации.

```
1 plot(sig, filtr(sig, IR))
```

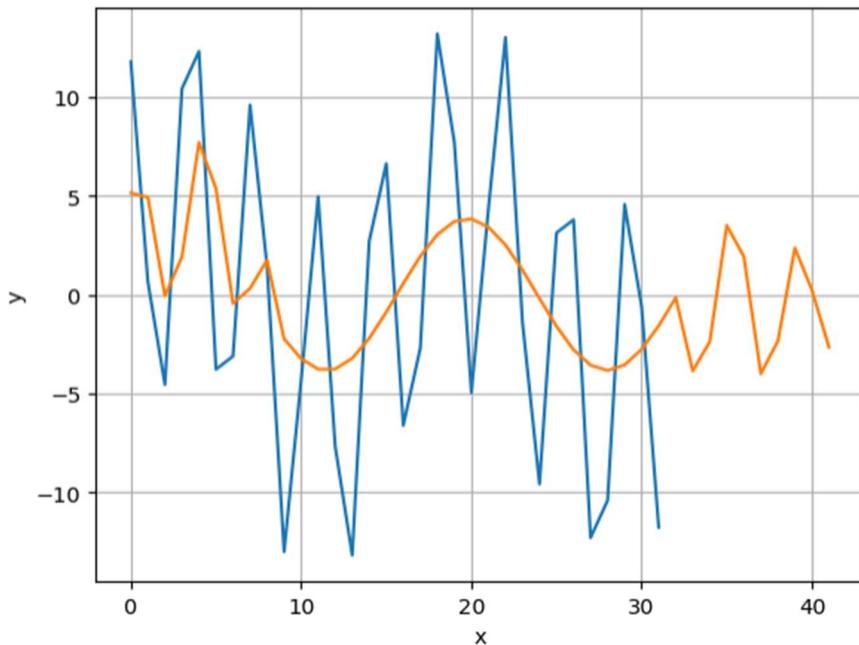


Рис. 8.6. Результат фильтрации

Синим – сигнал до фильтрации, жёлтым – после. Можно видеть, что в средней части сигнал после фильтрации практически идеально гладкий, тогда как по краям имеются существенные искажения. Это обусловлено тем, что по краям не все значения импульсной характеристики фильтра участвуют в свертке, так как они не накладываются на соответствующие значения сигнала, в средней же части участвуют все значения импульсной характеристики.

Зная, как применять фильтр к тому или иному сигналу, логично поставить вопрос о том, как же формировать импульсную характеристику фильтра, чтобы иметь возможность получить подавление на нужной частоте, ведь сама по себе импульсная характеристика не несет информации о частоте среза. Для этого можно поступить достаточно просто: задать желаемую частотную характеристику фильтра и определить на ее основе требуемую импульсную.

Алгоритм работы здесь следующий: задать дискретные отсчеты

требуемой амплитудно-частотной (АЧХ) и фазо-частотной характеристики (ФЧХ) фильтра, после чего подставить ФЧХ в качестве аргумента в комплексную экспоненту, перемножить АЧХ на комплексную экспоненту в степени ФЧХ и найти ОДПФ данного произведения. Полученный результат и будет требуемой импульсной характеристикой.

$$h_k = iF_k(|H(j\omega)| \cdot e^{j\phi}),$$

где h_k - k -ое значение импульсной характеристики фильтра;

$iF_k(\dots)$ - k -ое значение ОДПФ;

$|H(j\omega)|$ - АЧХ;

ϕ - ФЧХ;

j – мнимая единица.

Введём переменную частоты в отдельную переменную FMAX равную 10. После чего нужно задать две функции, первая будет сигналом, вторая пример отфильтрованной функции, и выведем эти две функции на графике.

```
1 def y(x: np.ndarray) -> np.ndarray:
2     return 3 * np.sin(FMAX * 2 * np.pi * x) + 2 * np.sin((FMAX / 4) * 2 * np.pi * x)
```

Рис. 8.7. Функция для фильтрации

```
1 def y_filtered_reference(x: np.ndarray) -> np.ndarray:
2     return 2 * np.sin((FMAX / 4) * 2 * np.pi * x)
```

Рис. 8.8. Примерный отфильтрованный сигнал

```
1 N = 30
2 T = 1
3 dx = T / N
4 fd = 1 / dx
5 x = np.arange(start=1, stop=N)
6 sig = y(x * dx)
7 filtered_reference = y_filtered_reference(x * dx)
8 plot(sig, filtered_reference)
```

Рис. 8.9. Создание массива сигнала

Желтый сигнал – примерный, отфильтрованный должен стремится к его виду, синий – это сигнал, который требуется очистить от высокочастотной помехи.

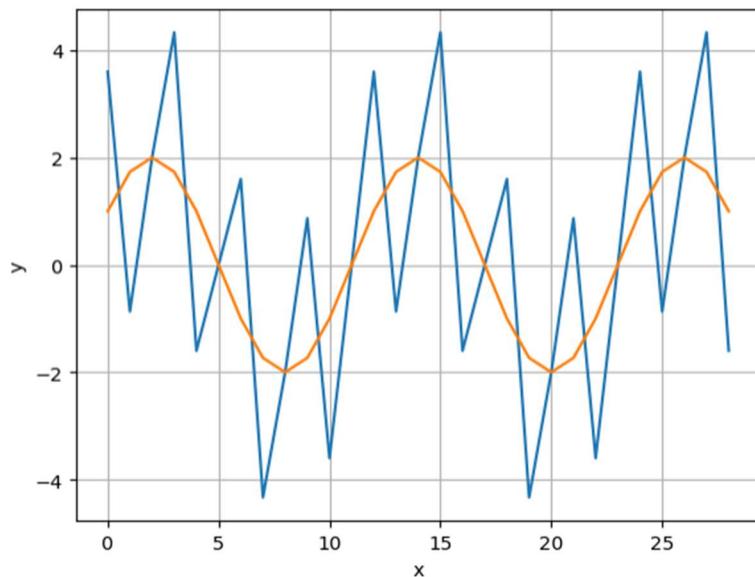


Рис. 8.10. График сигналов

Зададим спектр сигнала `sig`, позволяющих получить график спектра от 0 до $f_d/2$ частот (f_d – частота дискретизации), поскольку зеркальная часть модуля преобразования Фурье физического смысла не несет.

```
1 yf = fft(sig)
2 xf = fftfreq(N, dx)[:N // 2]
3 plot(xf, y=2.0 / N * np.abs(yf[0:N // 2]))
```

Рис. 8.11. Вычисление спектра и его отображение

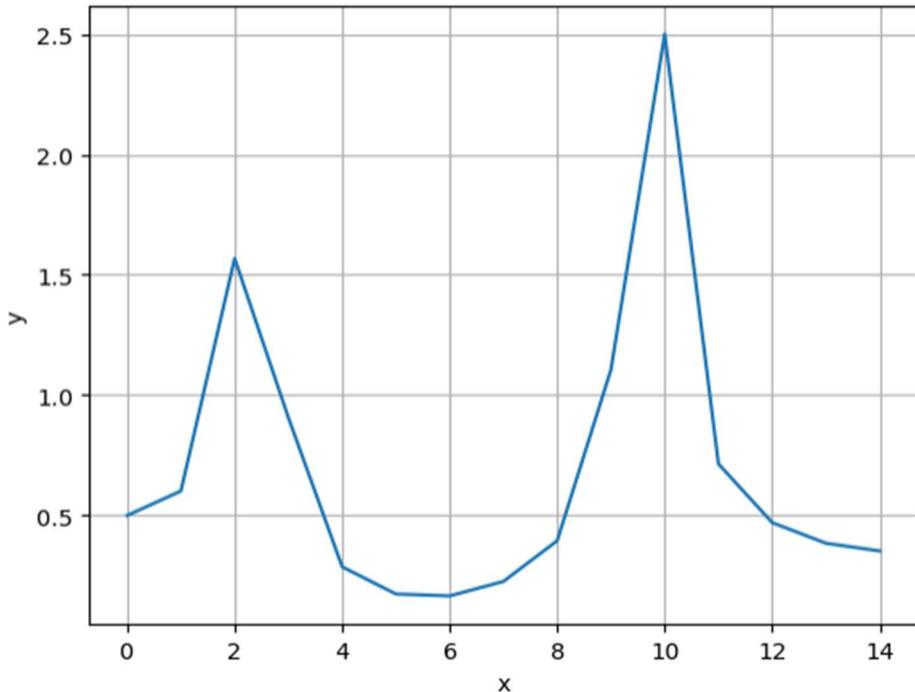


Рис. 8.12. График спектра sig

Сперва зададим функцию, которая формировала бы массив, состоящий из взятых по модулю элементов исходного массива, принимаемого в качестве параметра (это нужно, чтобы найти модуль ДПФ). В дальнейших примерах в рамках данной работы эта функция будет названа `absa(ar)`.

Спроектируем частотную характеристику, соответствующую требованиям. Следует сразу отметить, что при проектировании характеристики необходимо сохранить зеркальную часть модуля ДПФ, поскольку после умножения на экспоненту в степени ФЧХ результат подвергнется ОДПФ, и если отбросить область, соответствующую отрицательным частотам, то ОДПФ даст неадекватный результат.

```

1 Ha = np.array([1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1])
2 plot(Ha)

```

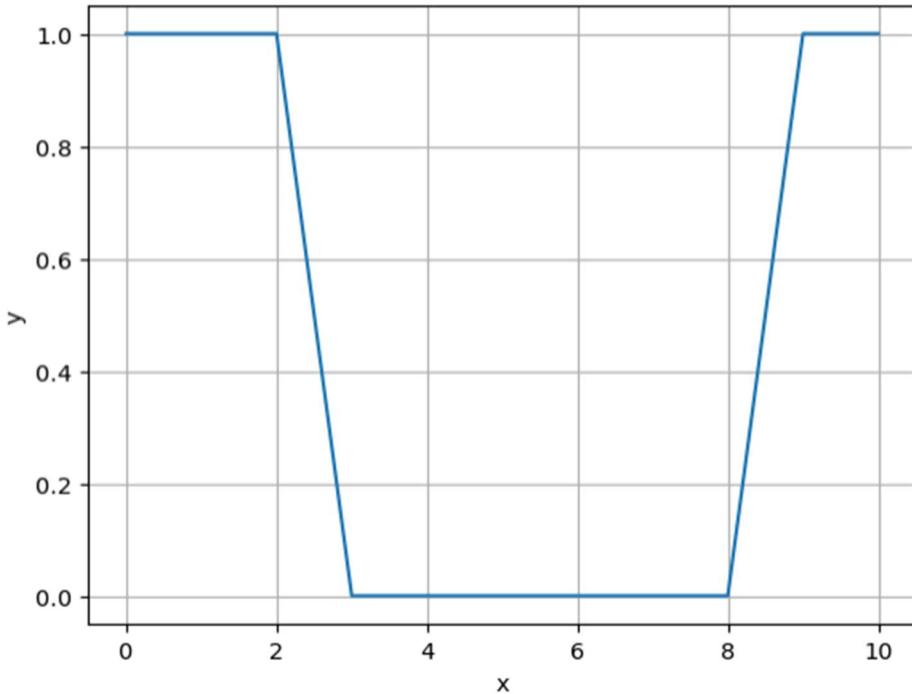


Рис. 8.13. Частотная характеристика фильтра

Еще раз напомним, что приведенный выше график (как и массив, его задающий), характеризует как фильтр будет реагировать на те или иные частоты. Последнему элементу массива (11-му) соответствует fd , первому — 0. Так как в данном примере $fd=30$ Гц, получим, что расстояние между соседними отсчетами по оси x имеет размер $\sim 2,7$ Гц. Число точек АЧХ фильтра равно числу точек его импульсной характеристики и выбирается пользователем (чем больше точек, тем дольше обработка, но тем точнее можно задать частоты на АЧХ). В данном случае 11 точек вполне хватает.

Если возникают проблемы при построении этой характеристики, можно посмотреть подробнее, как был получен данный график. При известной fd и подавляемой/пропускаемой частоте можно построить АЧХ идеального ФНЧ, который бы устранил 10 Гц, сохраняя 2 Гц. Построим данную теоретическую АЧХ.

```
1 Ha = np.array([1, 0.9, 0.7, 0, 0, 0, 0, 0, 0, 0, 0])
2 plot(Ha)
```

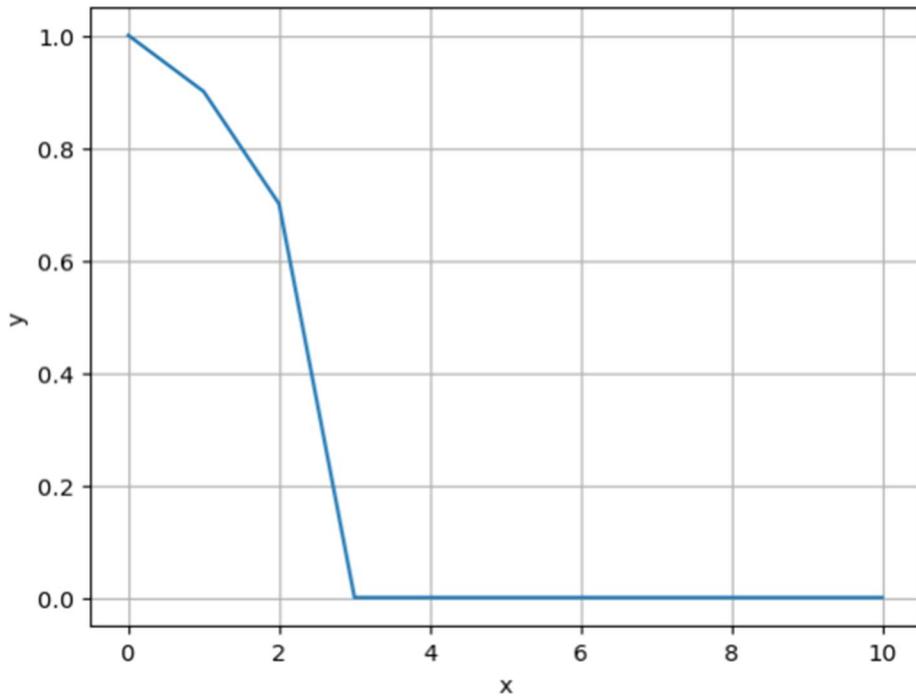


Рис. 8.14. Теоретическая АЧХ

Характеристика обращается в нуль на четвертом отсчете, который соответствует $2,7 \cdot 3 = \sim 8$ Гц, причем левее – плавно спадающая область пропускания, правее – ровная область подавления.

Достроим правую часть характеристики, отразив все значения характеристики, кроме первого. Отраженные элементы будут располагаться в следующем порядке – второй станет последним, третий – предпоследним и так далее. Первый элемент не учитывается потому, что он характеризует постоянную составляющую (соответствует нулевой частоте). В результате получим следующую характеристику, идентичную приведенной ранее.

```
1 Ha = np.array([1, 0.9, 0.7, 0, 0, 0, 0, 0, 0, 0.7, 0.9])
2 plot(Ha)
```

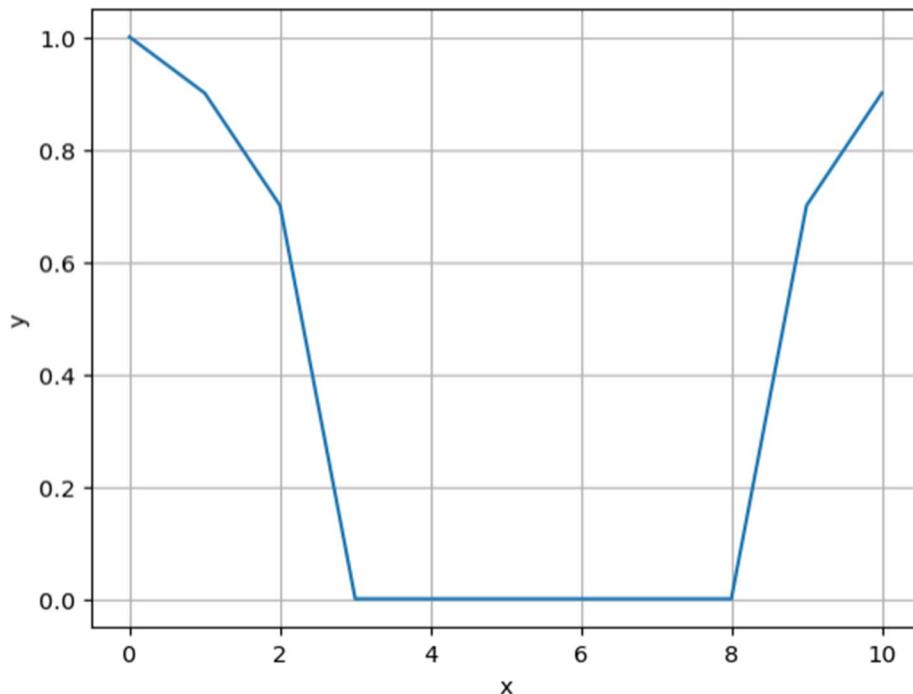


Рис.8.15. Дополненная АЧХ

Теперь зададим ФЧХ. Так как по заданию требований к ней нет, но без нее алгоритм реализован быть не может, поступаем наиболее простым способом – задаем линейную ФЧХ.

```
1 Fa = np.array([0, -0.2, -0.4, -0.6, -0.8, -1, 1, 0.8, 0.6, 0.4, 0.2])
2 plot(Fa)
```

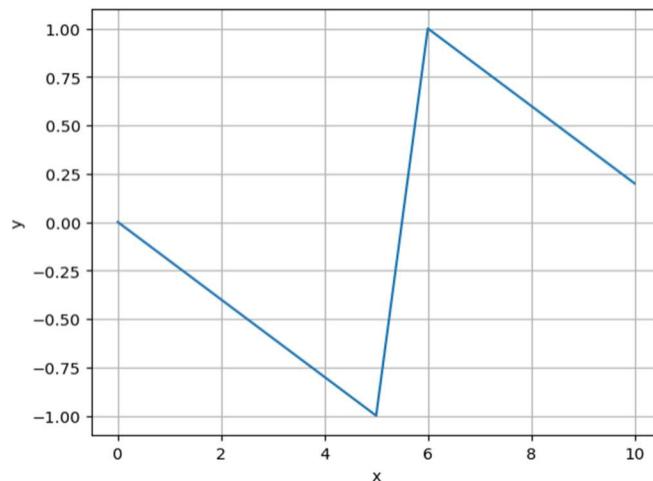


Рис.8.16. Теоретическая ФЧХ

График ФЧХ является не просто зеркальным, по аналогии с АЧХ, но и инвертированным в правой части. Такую ФЧХ можно использовать и при заданном варианте.

Имеем массивы ФЧХ и АЧХ, можем начать восстанавливать импульсную характеристику. Для этого, сперва, сформируем массив, элементы которого равны комплексной экспоненте в степени ФЧХ.

```
1 def arexp(array: np.ndarray) -> np.ndarray:  
2     result = np.zeros(array.size)  
3     for idx in range(result.size):  
4         result[idx] = np.exp(array[idx] * 1j)  
5     return result
```

```
1 Fe = arexp(Fa)
```

Рис.8.17. Массив экспоненты

Рассчитаем и отобразим отфильтрованный массив с изначальным и примерным. В данном случае синий сигнал – это фильтруемый массив, желтый – отфильтрованный, зелёный – примерный отфильтрованный

```
1 IR = np.fft.ifft(Ha * Fe)  
2 plot(sig, filtr(sig, IR), filtered_reference)
```

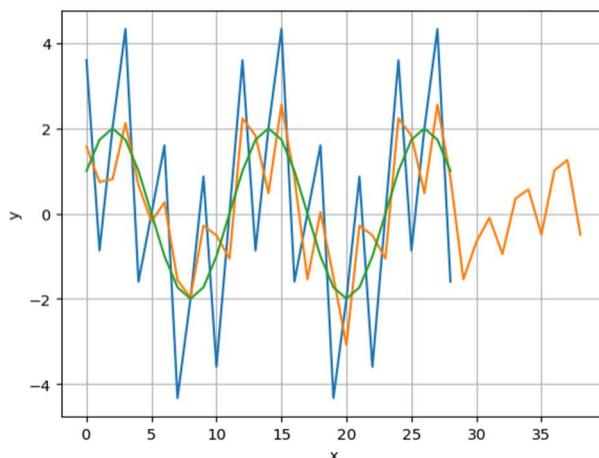


Рис.8.18. Фильтрация массива

Видим, что помеха подавлена, но далеко не полностью. Проверяем АЧХ фильтра.

```
1 plot(np.abs(np.fft.fft(IR)))
```

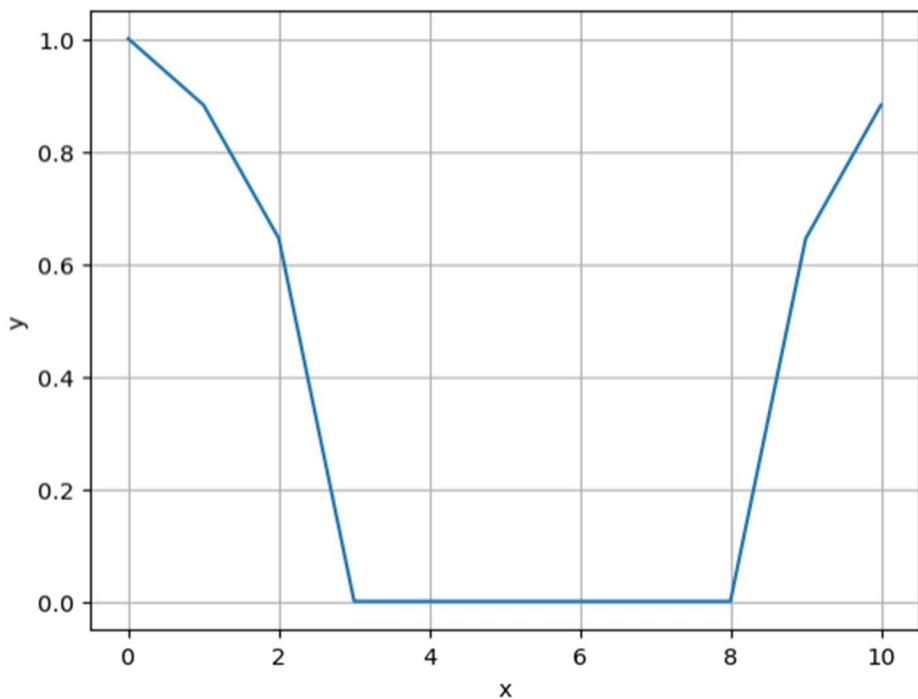


Рис. 8.19. АЧХ фильтра

Она совпадает заданной, значит все выполнено правильно.

Суть данной проблемы (подавления с меньшей амплитудой) в том, что проявляется так называемый эффект Гиббса. АЧХ рассчитанного фильтра (жёлтый график), точно проходит через узлы дискретизации, однако между точками дискретизации сильно отличается от идеальной характеристики (синий график). При этом наблюдается сильная неравномерность АЧХ в полосе пропускания фильтра, и высокий уровень боковых лепестков в полосе заграждения.

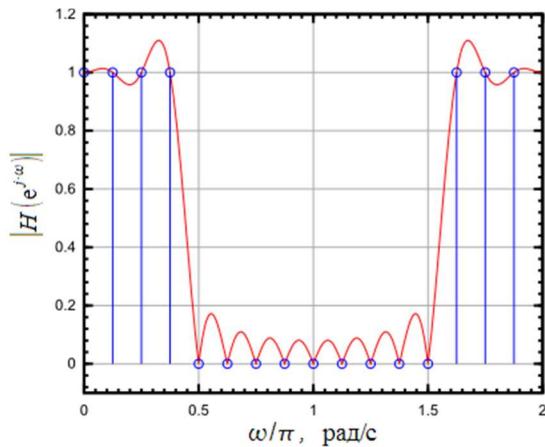


Рис.8.20. Эффект Гиббса

Поскольку 10 Гц не попадает точно в точку, где коэффициент усиления равен нулю, помеха подавляется не полностью. Эта проблема решается либо проектированием фильтра с более плавной АЧХ в местах разрывов, либо увеличением порядка фильтра (тогда пульсации остаются, но их амплитуда во много раз меньше), либо проектированием такой АЧХ, чтобы подавляемая частота точно попадала в одно из значений.

Так как последний способ достаточно просто реализуется, попробуем прибегнуть к нему. Поскольку фильтр состоит из 11 точек, а в той форме, в которой сигнал был задан выше, частота дискретизации равна числу точек в сигнале, чтобы точно попасть на АЧХ в некоторую частоту, следует сделать размер сигнала кратным числу точек фильтра (на практике, когда фильтр и сигнал имеют большее число значений, это не обязательно). Меняем N с 30 на 33.

Чтобы узнать, какой точке АЧХ соответствует та или иная частота, зададим итератор l и выражение, для формирования массива частот, соответствующих точкам импульсной характеристики.

```

1 fir = np.ceil(np.arange(IR.size + 1) * (fd / IR.size))
2 plot(np.abs(np.fft.fft(IR)), stem=True)
3 fir

```

Рис. 8.21. Массив частот

Теперь построим АЧХ фильтра и вызовем рядом значения массива частот, чтобы сопоставить данные.

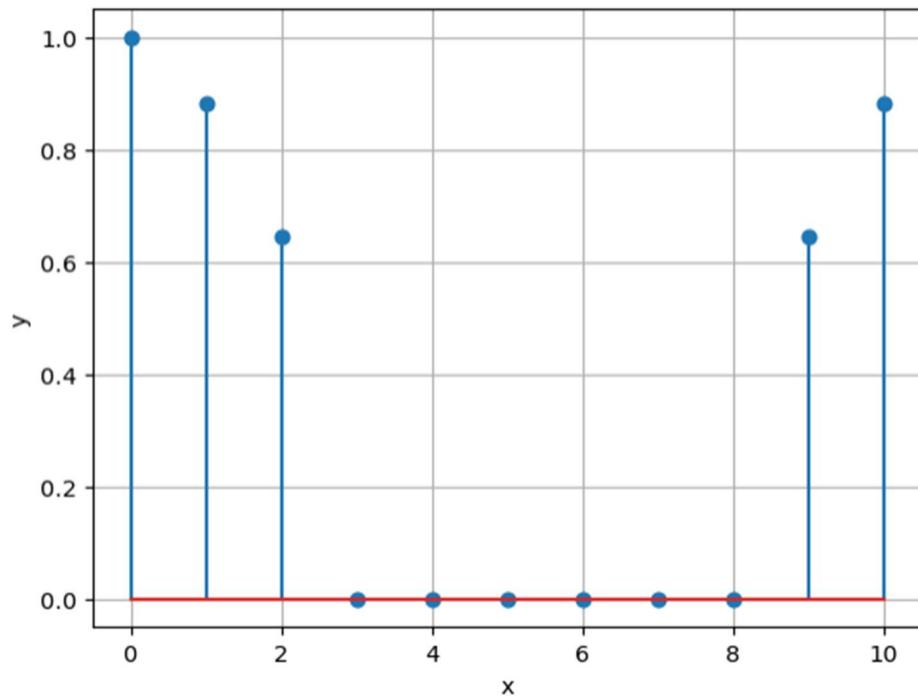


Рис. 8.22. АЧХ фильтра с частотами

По графику видно, что подавление начинается с 4 точки АЧХ (первое значение в нуле). В массиве частот видим, что полностью подавить фильтр с такой АЧХ может частоты 9 Гц, 12 Гц, 15 Гц (на частотах выше располагается зеркальная часть ДПФ).

Проверим работу фильтра. Изменим частоту помехи fmax на 25 и проверим результат.

```
1 N = 33
2 FMAX = 25
3 x = np.arange(start=1, stop=N)
4 sig = y(x * dx)
5 filtered_reference = y_filtered_reference(x * dx)
6 plot(sig, filtr(sig, IR), filtered_reference)
```

Рис. 8.23. Фильтрация данных

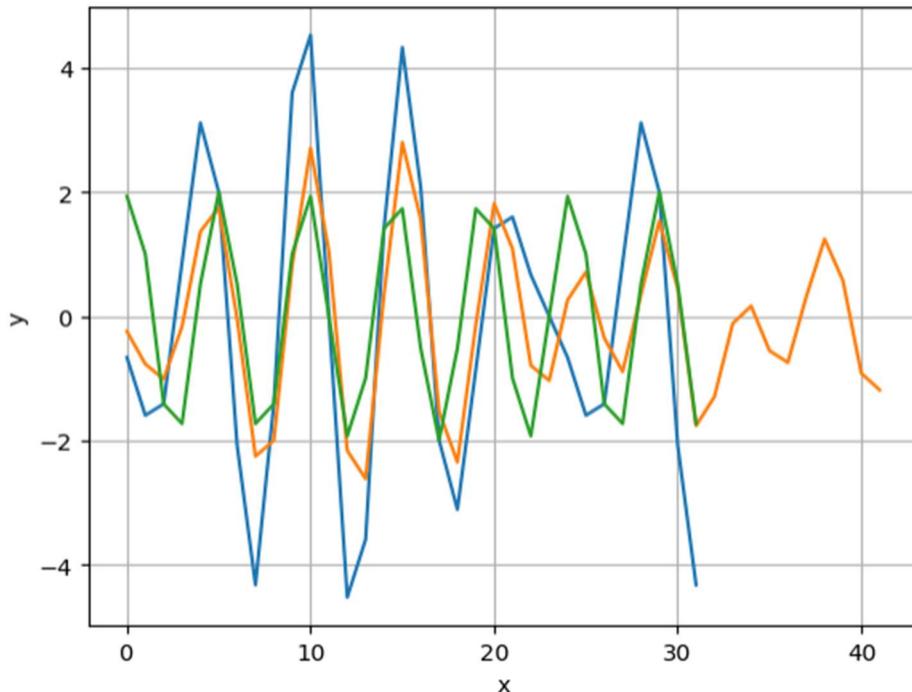


Рис. 8.24. График данных

Видно, что в средней части, где задействованы все элементы импульсной характеристики фильтра, помехи подавляется полностью, так как наш примерный зелёный сигнал и жёлтый отфильтрованный совпадают. Что говорит о том, что правильный подбор числа точек в фильтре на практике позволяет создавать простые и довольно точные алгоритмы устранения помехи.

Задание

1. Повторить приведенный выше пример.
2. Изменить указанные в таблице ниже выражения в соответствии с заданными значениями;
3. Спроектировать фильтр для подавления слагаемого на частоте f_{\max} в сигнале без подавления второго слагаемого.

Таблица 11.1

Варианты заданий

Вариант	Выражения
1,6,11, 16,21,26	$f_{max} := 12$ $y(x) := 10 \cdot \sin(f_{max} \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(9 \cdot 2 \cdot \pi \cdot x)$
2,7,12, 17,22,27	$f_{max} := 15$ $y(x) := 10 \cdot \sin(f_{max} \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(12 \cdot 2 \cdot \pi \cdot x)$
3,8,13, 18,23,28	$f_{max} := 9$ $y(x) := 10 \cdot \sin(f_{max} \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(6 \cdot 2 \cdot \pi \cdot x)$
4,9,14, 19,24,29	$f_{max} := 6$ $y(x) := 10 \cdot \sin(f_{max} \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(3 \cdot 2 \cdot \pi \cdot x)$
5,10,15, 20,25,30	$f_{max} := 9$ $y(x) := 10 \cdot \sin(f_{max} \cdot 2 \cdot \pi \cdot x) + 2 \cdot \sin(3 \cdot 2 \cdot \pi \cdot x)$

РАБОТА №9. НЕПРЕРЫВНОЕ ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ ДИСКРЕТНЫХ СИГНАЛОВ

Цель работы: изучение возможностей применения вейвлет-преобразования при обработке сигналов.

Планируемая продолжительность: от 2 до 4 академических часов.

Тип работы:  с использованием компьютерных средств.

Теоретические основы

Непрерывное вейвлет преобразование (НВП) – еще один инструмент анализа сигнала, а также его сжатия. Для сигнала $f(t)$ НВП определяется следующим образом:

$$W(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \cdot \psi\left(\frac{t-b}{a}\right) dt, \quad (13.1)$$

где $\psi\left(\frac{t-b}{a}\right)$ - вейвлет, используемый для анализа;

a – параметр масштаба вейвлета;

b – параметр сдвига вейвлета.

То есть, вычислив интеграл (13.1) для какого-либо сигнала получим двумерную поверхность с величинами a и b в качестве параметров. Принято обозначать по оси абсцисс параметр a , по оси ординат – b , а $W(a,b)$, который называется вейвлет-коэффициентом при данных a,b . Соответственно результат вычисления (13.1) – набор вейвлет-коэффициентов при всех допустимых a и b .

С увеличением a вейвлет сжимается вдоль оси абсцисс (и наоборот), с увеличением b – сдвигается вдоль сигнала. При сжатии вейвлета, в сущности, увеличивается частота присутствующих в нем составляющих (так как при сжатии уменьшается период), поэтому можно сказать, что **ось a на графике вейвлет-коэффициентов сопоставима с осью частот, а ось b – с осью времени.** То есть, НВП относится к классу частотно-временных преобразований, позволяющих анализировать изменение частотного состава сигнала с течением времени.

Можно обратить внимание, что при конкретных значениях a (в

данном примере $a=1$) выражение (13.1) принимает знакомый вид:

$$W(1, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \cdot \psi(t - b) dt,$$

что соответствует ВКФ сигнала и вейвлета. Следовательно, НВП является множеством функций взаимной корреляции сигнала с вейвлетом при различных масштабах a последнего.

Для дискретных сигналов, за счет замены интеграла на сумму, (13.1) примет вид:

$$W(a, b) = \frac{\Delta t}{\sqrt{a}} \sum_{j=1}^N S_j \cdot \psi\left(\frac{j \cdot \Delta t - b}{a}\right), \quad (13.2)$$

где Δt - интервал дискретизации;

S - дискретный сигнал.

В качестве вейвлета может быть использован целый ряд функций, допускается даже использование собственной функции в качестве вейвлета. Чтобы быть вейвлетом, функция должна удовлетворять некоторым требованиям, которые здесь описываются не будут. На практике чаще всего используют МНАТ-вейвлет (Mexican Hat, мексиканская шляпа):

$$\psi(t) = (1 - t^2)e^{-\frac{t^2}{2}}. \quad (13.3)$$

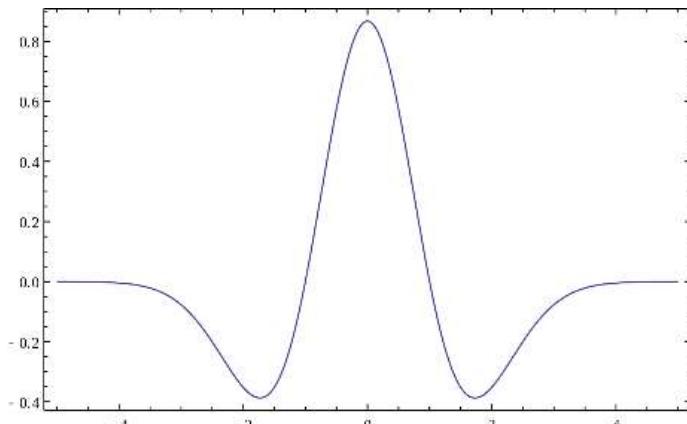


Рис. 9.1. МНАТ-вейвлет

Для визуализации НВП принято использовать так называемую скалограмму, значения для которой вычисляются на основе вейвлеткоэффициентов:

$$Sk(a, b) = |W(a, b)|^2. \quad (13.4)$$

Ниже представлен пример сигнала и его скалограммы.

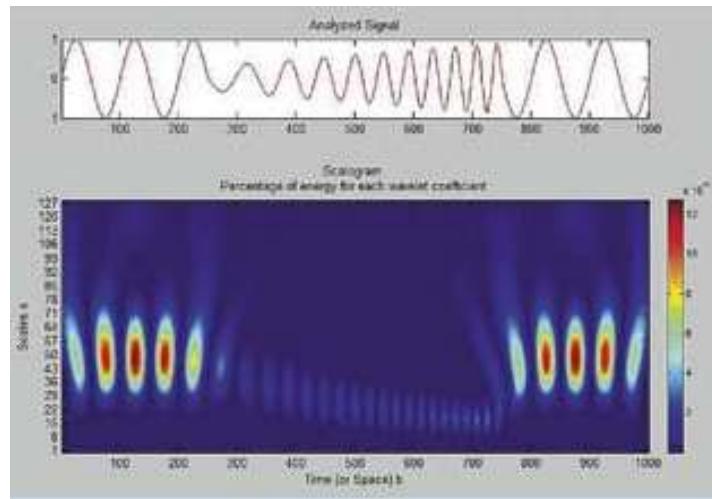


Рис. 9.2. Сигнал и его скалограмма

Непрерывное вейвлет-преобразование в Python

Как и до этого импортируем зависимости, переносим `plot`, и добавляем ещё одну функцию отрисовки графиков `plot_scalogram`, как следует из её названия при помощи неё будет рисовать график вейвлет коэффициентов.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from typing import Callable
4 from scipy.fft import fft, fftfreq
5 %matplotlib widget
6 # from google.colab import output
7 # output.enable_custom_widget_manager()

```

Рис. 9.3. Импорт зависимостей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

```

1 def plot(*args, y=None, stem=False) -> None:
2     ax = plt.figure()
3     if y is None:
4         for y in args:
5             x = np.arange(y.size)
6             if stem:
7                 plt.stem(x, y)
8             else:
9                 plt.plot(x, y)
10    else:
11        if stem:
12            plt.stem(args[0], y)
13        else:
14            plt.plot(args[0], y)
15    plt.grid(True)
16    plt.ylabel('y')
17    plt.xlabel('x')
18    plt.show()

```

Рис. 9.4. Функция построения графиков

```

1 def plot_scalogram(wavelet_coeficients: np.ndarray) -> None:
2     xlist = ylist = np.arange(0, wavelet_coeficients.shape[0])
3     X, Y = np.meshgrid(xlist, ylist)
4     Z = wavelet_coeficients
5     fig, ax = plt.subplots(1, 1)
6     cp = ax.contourf(X, Y, Z)
7     fig.colorbar(cp, label='Значения вейвлет-коэффициентов')
8     ax.set_title('Скалограмма сигнала')
9     ax.set_ylabel('Ряды')
10    ax.set_xlabel('Отсчёты')
11    plt.show()

```

Рис. 9.5. Функция построения графика скалограммы

Для реализации НВП необходимо сперва задать вейвлет. В данной работе ограничимся МНАТ-вейвлетом.

```

1 def mexican_hat(t: np.ndarray):
2     return np.exp((-1) * np.power(t, 2) / 2) * (1 - t ** 2)

```

```

1 t = np.linspace(start=-6, stop=6)
2 plot(mexican_hat(t))

```

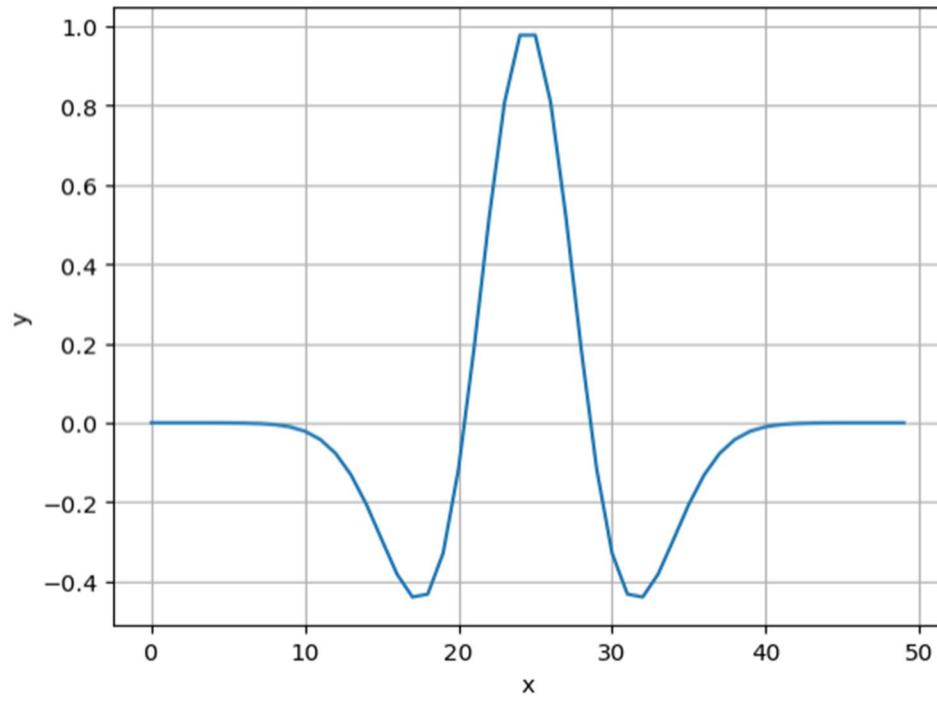


Рис. 9.6. МНАТ-вейвлет в Smath

Зададим анализируемый сигнал. Чтобы увидеть основное отличие ДПФ от НВП, в сигнале предусмотрено резкое повышение частоты.

```

1 def y(x: np.ndarray) -> np.ndarray:
2     result = np.zeros(x.size)
3     for idx in range(result.size):
4         if x[idx] < np.pi:
5             result[idx] = np.sin(0.5 * 2 * np.pi * x[idx])
6         else:
7             result[idx] = np.sin(0.8 * 2 * np.pi * x[idx])
8     return result

```

```

1 N = 30
2 x = np.arange(start=1, stop=30)
3 signal = y(x * (2 * np.pi / N))
4 plot(signal)

```

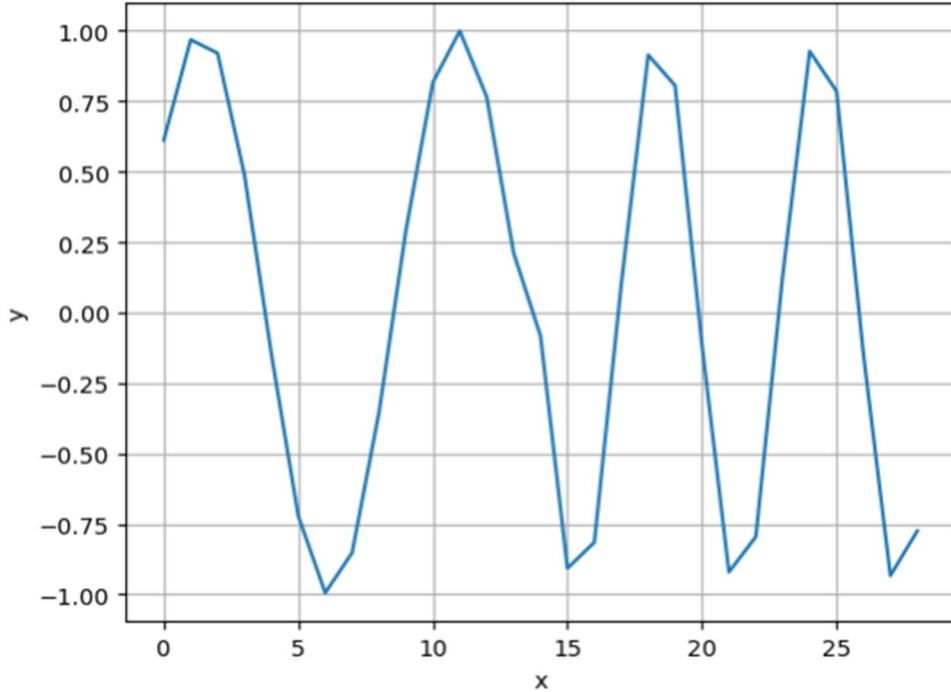


Рис. 9.7. Анализируемый сигнал

Теперь можно сформировать выражение для самого НВП. Как было подмечено выше, НВП при конкретных значениях масштаба является ВКФ вейвлета и сигнала. Воспользуемся этими данными и на основе спроектированного в предыдущих работах выражения для ВКФ зададим в Smath выражение (13.2), как это представлено на рисунке ниже.

```

1 def wavelet_transform(signal: np.ndarray,
2                     psi_func: Callable):
3     dt = 2 * np.pi / signal.size
4
5     def wavelet_transform_array(a: float):
6
7         def wavelet_transform_element(b: float):
8             result_sum = np.zeros(signal.size)
9             for idx in range(result_sum.size):
10                 result_sum[idx] = signal[idx] * psi_func((idx + 1) * dt - b) / a
11             return np.sum(result_sum)
12
13     result_array = np.zeros(signal.size)
14     for idx in range(result_array.size):
15         result_array[idx] = wavelet_transform_element((idx + 1) * dt)
16     return result_array
17
18     result = []
19     for k in range(signal.size):
20         result.append(wavelet_transform_array((k + 1) * dt))
21     return np.array(result)

```

Рис. 9.8. Задание НВП

Результат построения скалограммы для описанного выше примера на рисунке.

```
1 plot_scalogram(wavelet_transform(signal, mexican_hat))
```

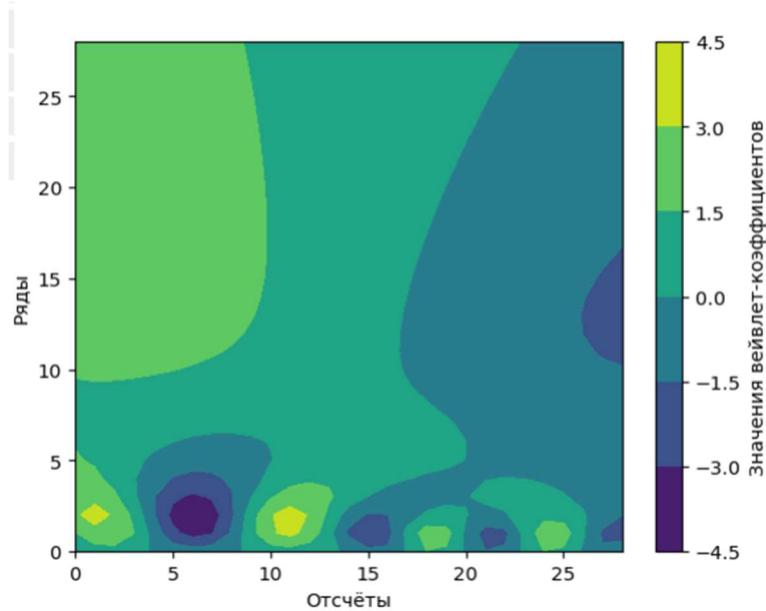


Рис. 9.9. Коэффициенты НВП

Из графика коэффициентов видно, что при малых значениях a , что соответствует большому масштабу вейвлета, мелких деталей выявлять не удается, но четко видна граница перехода сигнала из одной частоты в другую. Причем, при масштабе, которому соответствует приблизительно 9й ряд, эта граница по временному положению совпадает с ее положением в самом сигнале. Идентифицировать данный участок как границу двух частот легко – при масштабе большем граница смещается в сторону, а при меньшем начинают проявляться более мелкие детали, соответствующие отдельным полупериодам синусоиды (то есть в подобных случаях следует искать такой масштаб, при котором за счет совсем уж мелких деталей, таких как отдельные вершины, граница частот не уводится в сторону, и при котором детализация не столь мала, что граница «упливает» или размывается). Подобный случай рассматривается на рисунке ниже (в сущности, совпадает с нашим примером, но здесь коэффициентов больше, сигнал большей длительности, за счет чего результат

выглядит плавнее, присутствуют полутона).

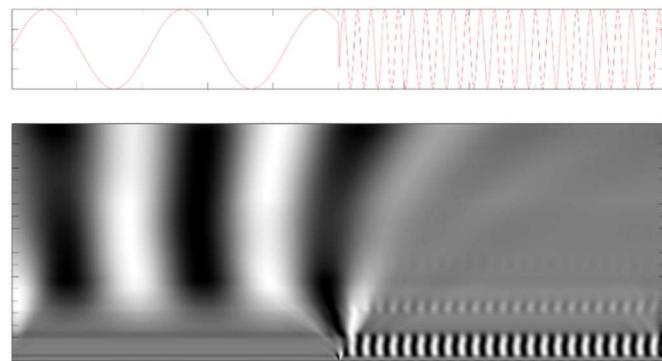


Рис. 9.10. Пример резкого перехода частот

Теперь повторно построим скалограмму, но только в этот раз каждое значения возьмём абсолютно, прибавим 50, так как значения вейвлет коэффициентов будут меньше нуля, следовательно при возведении в степень большая часть данных будет сглажена. И соответственно после суммирования возведём во 2 степень полученные данные

```
1 plot_scalogram(np.power(np.abs(wavelet_transform(signal, mexican_hat)) + 50, 2))
```

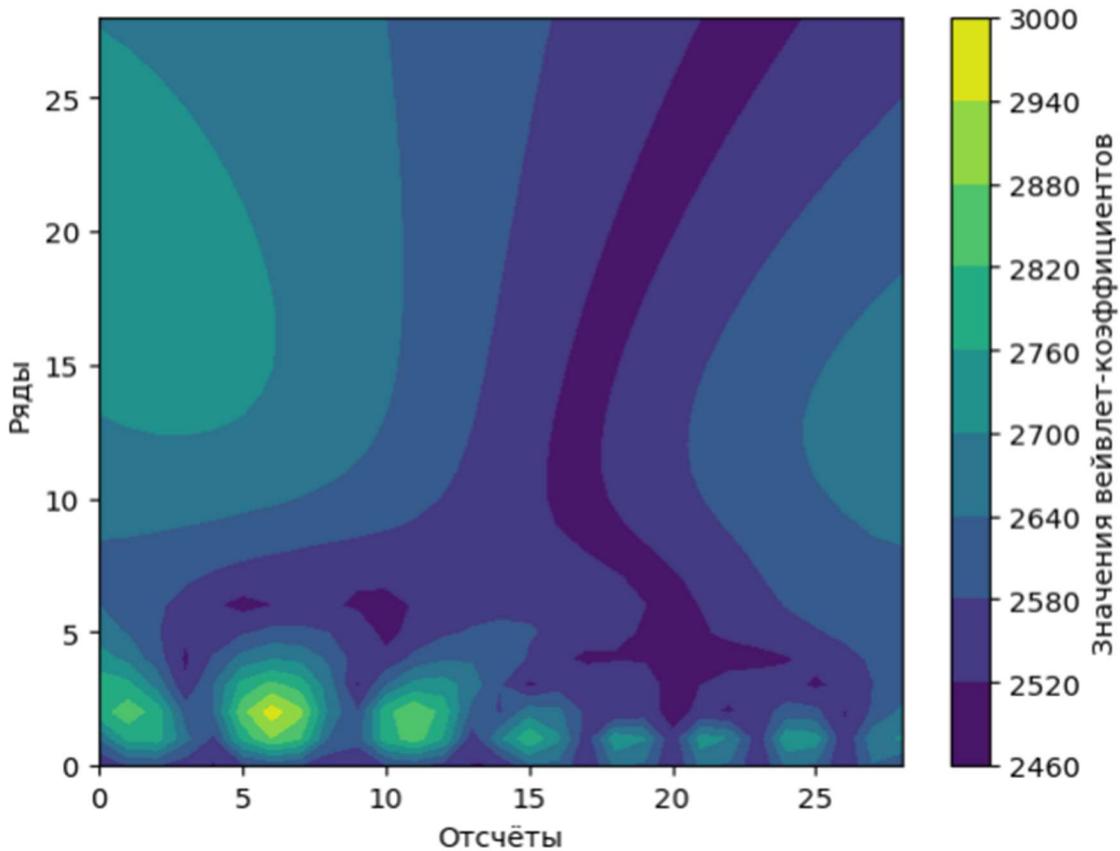


Рис. 9.11. Скалограмма

Граница частот видна еще контрастнее из-за модуля и квадратирования.

Сравним скалограмму с частотным спектром. Для этого зададим выражения для его реализации, рассмотренные в прошлых работах.

```
1 N = 30
2 T = (2 * np.pi / N)
3 x = np.linspace(0.0, N * T, N, endpoint=False)
4 yf = fft(y(x))
5 xf = fftfreq(N, T)[:N // 2]
```

Рис. 9.12. Выражения для реализации спектра

И построим соответствующий график.

```
1 plot(xf, y=2.0 / N * np.abs(yf[0:N // 2]))
```

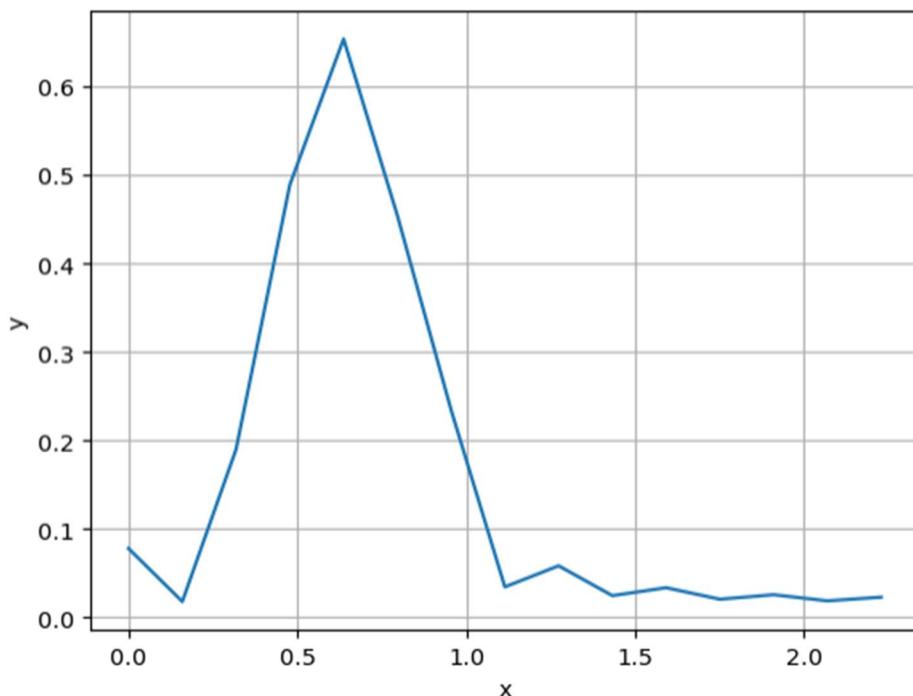


Рис. 9.13. График спектра

Можно видеть, что из-за близости частот на спектре они объединяются. При этом данный график не представляет никакой информации об изменении во времени спектра сигнала, в отличие от скалограммы (или самих вейвлет-коэффициентов).

Задание 1

1. Задать сигнал в соответствии с вариантом, построить его график
2. Задать МНАТ-вейвлет, построить его график
3. Задать выражения для реализации НВП
4. Сформировать массив вейвлет-коэффициентов заданного сигнала
5. Построить в нем как график коэффициентов, так и скалограмму.
6. Построить спектр сигнала по ДПФ.
7. Записать вывод об отличиях и схожести получения спектра по ДПФ и скалограммы вейвлет-преобразования

Таблица 13.1.

Варианты заданий

Варианты	Сигнал
1,6,11, 16,21,26	$y(x) := \begin{cases} \sin(0,5 \cdot 2 \cdot \pi \cdot x) + \sin(0,8 \cdot 2 \cdot \pi \cdot x) \\ \text{else} \\ \sin(0,8 \cdot 2 \cdot \pi \cdot x) \end{cases}$
2,7,12, 17,22,27	$y(x) := \begin{cases} \sin(0,5 \cdot 2 \cdot \pi \cdot x) \\ \text{else} \\ \sin(0,8 \cdot 2 \cdot \pi \cdot x) + \sin(0,5 \cdot 2 \cdot \pi \cdot x) \end{cases}$
3,8,13, 18,23,28	$y(x) := \begin{cases} \cos(0,5 \cdot 2 \cdot \pi \cdot x) \\ \text{else} \\ \cos(0,8 \cdot 2 \cdot \pi \cdot x) \end{cases}$
4,9,14, 19,24,29	$y(x) := \begin{cases} \cos(0,5 \cdot 2 \cdot \pi \cdot x) + \cos(0,8 \cdot 2 \cdot \pi \cdot x) \\ \text{else} \\ \cos(0,8 \cdot 2 \cdot \pi \cdot x) \end{cases}$

**5,10,15,
20,25,30**

```
 $y(x) := \begin{cases} \cos(0, 5 \cdot 2 \cdot \pi \cdot x) & \text{if } x < \pi \\ \cos(0, 8 \cdot 2 \cdot \pi \cdot x) + \cos(0, 5 \cdot 2 \cdot \pi \cdot x) & \text{else} \end{cases}$ 
```

РАБОТА №10. ФИЛЬТРАЦИЯ СИГНАЛОВ С ПРИМЕНЕНИЕМ НЕПРЕРЫВНОГО ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ

Цель работы: изучение возможностей применения непрерывного вейвлет-преобразования для фильтрации сигналов.

Планируемая продолжительность: от 2 до 4 академических часов.

Тип работы:  с использованием компьютерных средств.

Фильтрация сигнала с применением непрерывного вейвлет-преобразования

В данном практическом занятии, как и на предыдущих, рассматривается НВП дискретных сигналов. Результатом такого НВП является массив коэффициентов (двумерный массив), в котором от столбца к столбцу меняется параметр b , связанный с временем, а от строки к строке – параметр a , связанный с частотой. Таким образом, массив вейвлет-коэффициентов можно охарактеризовать как поверхность, содержащая сведения о изменении сигнала во времени (по оси x) и по частоте (по оси y).

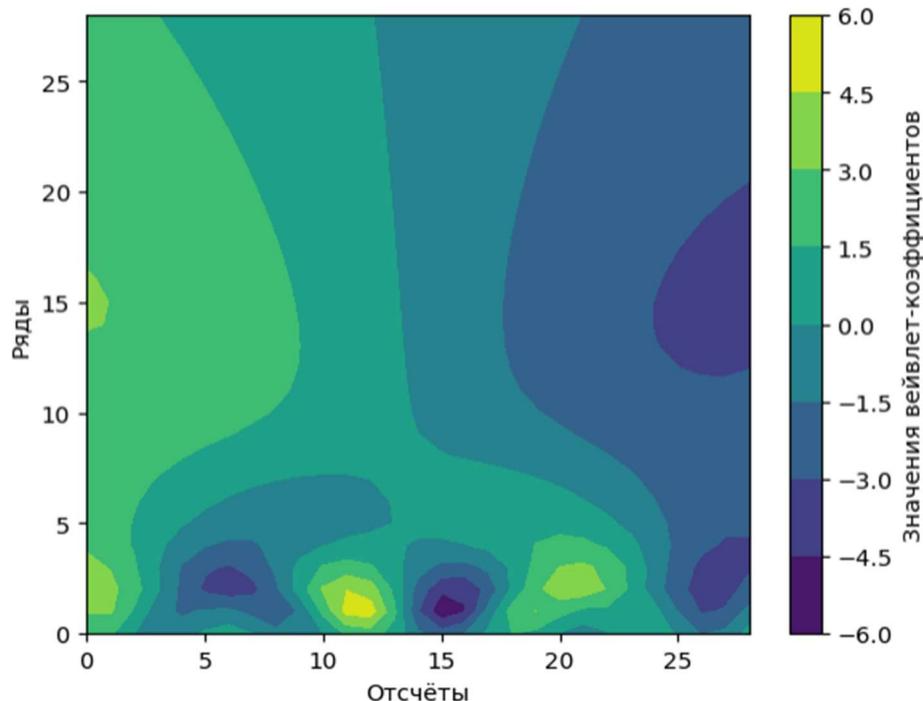


Рис. 10.1. Вейвлет-коэффициенты

Помимо этого известно, что имеется обратное НВП (ОНВП), позволяющее из массива вейвлет-коэффициентов восстановить исходный сигнал. Следовательно, **если обнулить часть вейвлет-коэффициентов в массиве и затем восстановить из него сигнал с помощью ОНВП, можно очистить сигнал от соответствующих частотных компонентов в соответствующем временном участке сигнала.** Именно таким образом и реализуется фильтрация на базе НВП.

Обратное непрерывное вейвлет преобразование

Известно выражение для прямого НВП:

$$W(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \cdot \psi\left(\frac{t-b}{a}\right) dt,$$

где $\psi\left(\frac{t-b}{a}\right)$ - вейвлет, используемый для анализа;

a – параметр масштаба вейвлета;

b – параметр сдвига вейвлета.

Для дискретных сигналов данное выражение принимает вид:

$$W(a, b) = \frac{\Delta t}{\sqrt{a}} \sum_{j=1}^N S_j \cdot \psi\left(\frac{j \cdot \Delta t - b}{a}\right), \quad (14.1)$$

где Δt - интервал дискретизации;

S - дискретный сигнал.

При этом a, b также изменяют дискретно с некоторым шагом. В результате и формируется массив вейвлет-коэффициентов.

Для восстановления из массива вейвлет-коэффициентов исходного сигнала используют выражение:

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{\sqrt{a} \cdot a^2} W(a, b) \cdot \psi\left(\frac{t-b}{a}\right) da db,$$

где C_ψ - постоянная допустимости используемого вейвлета, **для используемого в работе вейвлета МНАТ** $C_\psi = \pi$.

Для дискретных сигналов с учетом $C_\psi = \pi$ данное выражение принимает вид:

$$S_j = \frac{1}{\pi} \sum_{k=1}^L \sum_{i=1}^M \frac{1}{\sqrt{i \cdot \Delta a} \cdot (i \cdot \Delta a)^2} W_{i,k} \cdot \psi\left(\frac{j \cdot \Delta t - k \cdot \Delta b}{i \cdot \Delta a}\right), \quad (14.2)$$

где Δt - интервал дискретизации;
 Δa - шаг изменения масштаба (выбирается при прямом НВП);
 Δb - шаг изменения сдвига (выбирается при прямом НВП);
 i – текущая строка;
 k – текущий столбец;
 $W_{i,k}$ - текущий элемент в массиве вейвлет-коэффициентов;
 j – текущий номер элемента в дискретном сигнале S .

Реализация алгоритма фильтрации с помощью НВП Python

Для начала импортируем следующие зависимости.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from typing import Callable
4 %matplotlib widget
5 # from google.colab import output
6 # output.enable_custom_widget_manager()

```

Рис. 10.2. Импорт зависимостей

Если в работе применяется Google Colab в качестве редактора, то для работы интерактивных графиков нужно раскомментировать последние две строчки из ячейки импортов.

А также из предыдущей работы нужно взять функции `plot`, `plot_scalogram`, `mexican_hat`, `wavelet_transform`

После чего – сам фильтруемый сигнал.

```

1 def y(x: np.ndarray) -> np.ndarray:
2     result = np.zeros(x.size)
3     for idx in range(result.size):
4         result[idx] = np.sin(0.5 * 2 * np.pi * x[idx]) + \
5             np.sin(0.8 * 2 * np.pi * x[idx] + (np.pi / 6))
6     return result

```

Рис. 10.3. Тестовый сигнал

```

1 N = 30
2 x = np.arange(start=1, stop=30)
3 signal = y(x * (2 * np.pi / N))
4 plot(signal)

```

Рис. 10.4. Создание тестового сигнала

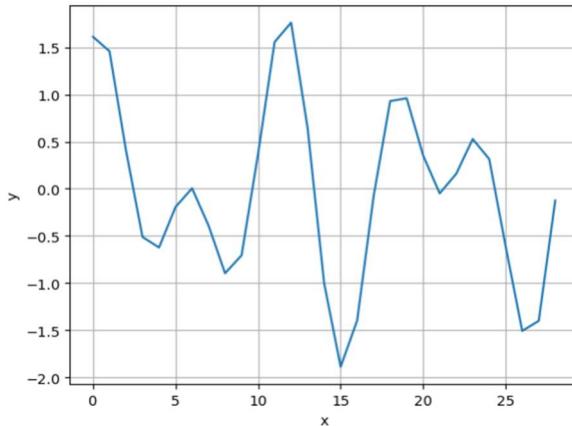


Рис. 10.5. График тестового сигнала

И обратное НВП (в нашем случае $\Delta t = \Delta a = \Delta b$).

```

1 def inverse_wavelet_transform(
2     array: np.ndarray,
3     psi_func: np.ndarray
4 ) -> np.ndarray:
5     dt = 2 * np.pi / signal.size
6     """
7         Так как на вход передается массив, где
8         элементы массива являются срезам по рядам, то
9         нужно транспонировать матрицу, чтобы получить
10        срезы по отсчетам
11    """
12    array = array.T
13
14    def inverse_wavelet_element_transform(t: float) -> np.ndarray:
15        result_element = np.zeros(shape=(array.shape[0], array.shape[0]))
16        for y in range(1, result_element.shape[0] + 1):
17            for x in range(1, result_element.shape[1] + 1):
18                result_element[y - 1, x - 1] = array[y - 1, x - 1] * \
19                    psi_func((t - y * dt) / (x * dt)) * \
20                    (1 / (((x * dt) ** 2) * np.sqrt(x * dt)))
21        return ((dt ** 2) / np.pi) * np.sum(result_element)
22
23    result = []
24    for t in range(array.shape[0]):
25        result.append(inverse_wavelet_element_transform((t + 1) * dt))
26    return np.array(result)

```

Рис. 10.6. Обратное НВП

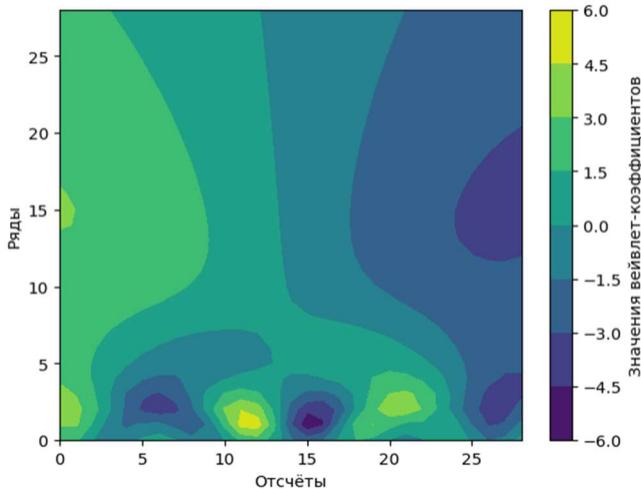


Рис. 10.7. График вейвлет-коэффициентов

Попробуем отфильтровать в первых 10 отсчётах высокие частоты, которые, согласно графику коэффициентов, сосредоточены в рядах 0-7 (вторая граница берётся не включительно поэтому индексы на единицу больше).

```
1 wt = wavelet_transform(signal, mexican_hat)
2 wt[0:8, 0:11] = 0
3 plot_scalogram(wt)
```

Рис. 10.8. Фильтрация коэффициентов

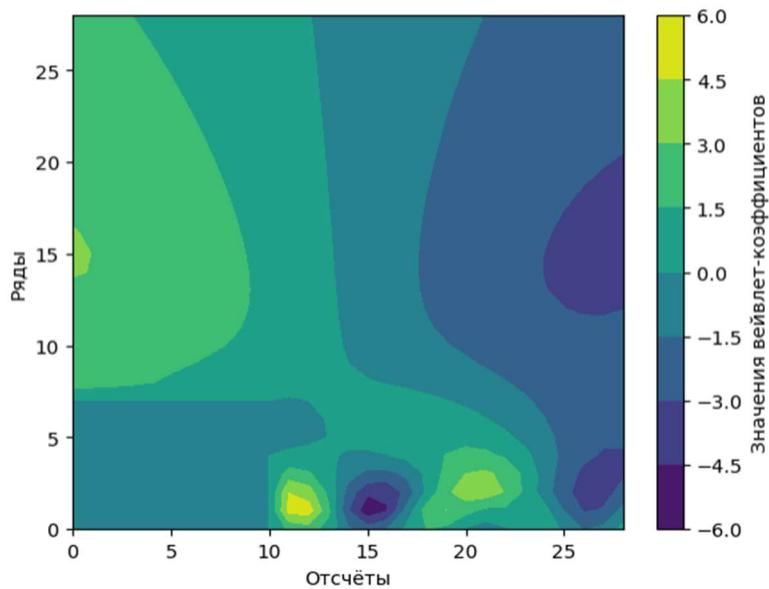


Рис. 10.9. Отфильтрованные коэффициенты

Строим график сигнала, полученного из отфильтрованного массива.

```
1 plot(inverse_wavelet_transform(wt, mexican_hat))
```

Рис. 10.10. Построение графика сигнала

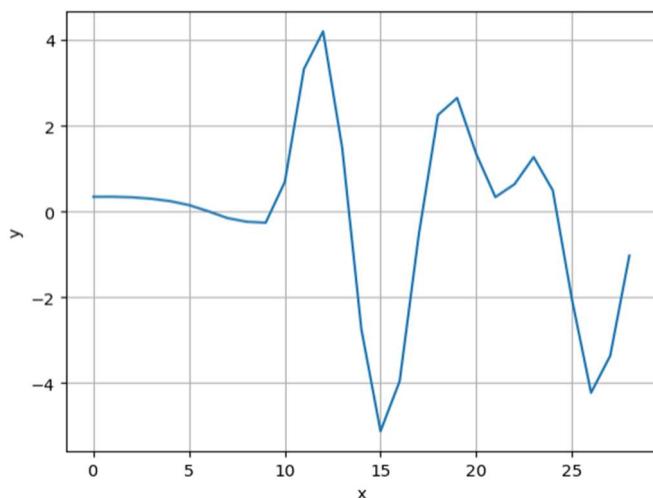


Рис. 10.11. Отфильтрованный сигнал

Как видим, в сигнале отсутствуют выбросы на границе фильтрации, при том, что коэффициенты были обнулены без какого-либо окна.

Попробуем устраниить только высокочастотную составляющую в более широкой временной области сигнала, не подавив низкочастотную.

```
1 wt = wavelet_transform(signal, mexican_hat)
2 wt[0:3, 0:21] = 0
3 plot_scalogram(wt)
```

Рис. 10.12. Фильтрация коэффициентов

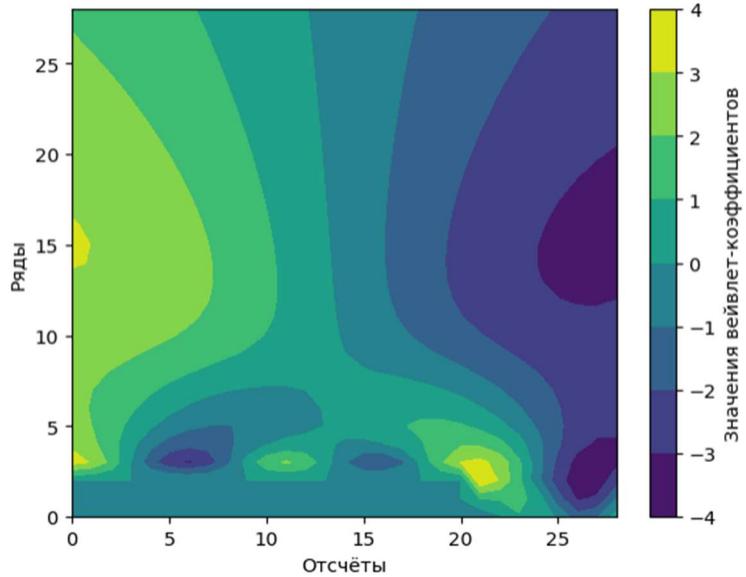


Рис. 10.13. Отфильтрованные коэффициенты

```
1 plot(inverse_wavelet_transform(wt, mexican_hat))
```

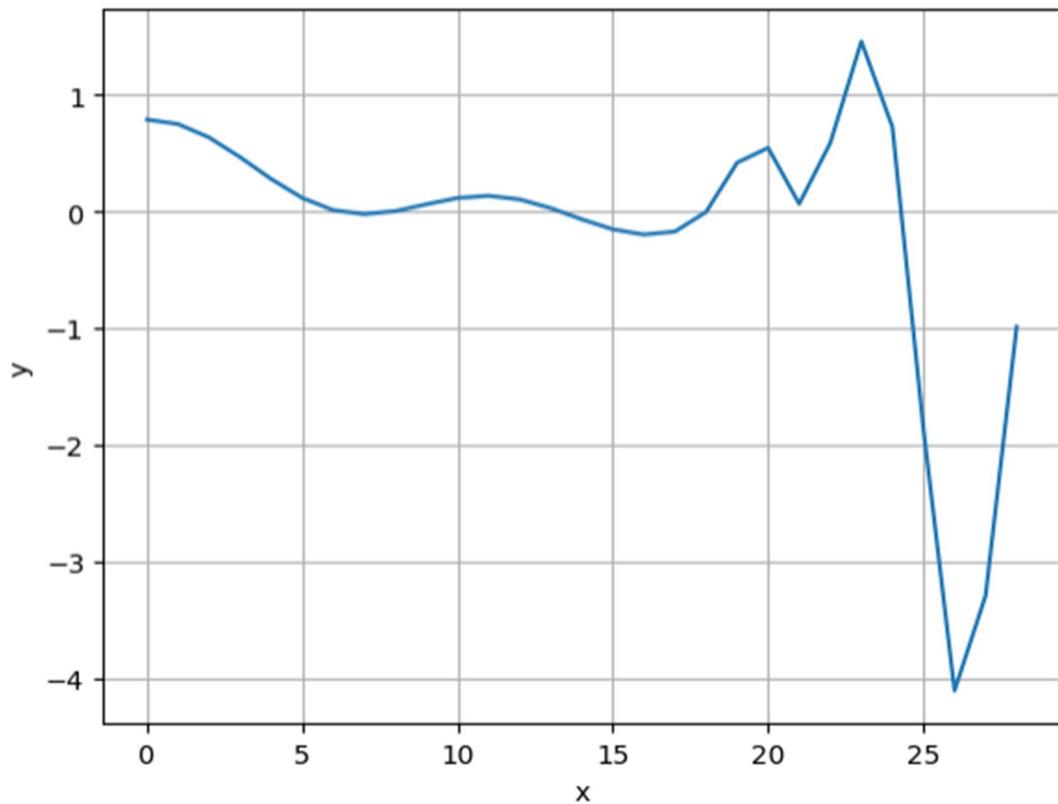


Рис. 10.14. Отфильтрованный сигнал

Задание

Реализовать алгоритм фильтрации из последнего раздела для заданного вариантом сигнала.

Таблица 14.1

Варианты заданий

Вариант	Сигнал
1,6,11, 16,21,26	$y(x) := \sin(0, 4 \cdot 2 \cdot \pi \cdot x) + \sin\left(0, 9 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{6}\right)$
2,7,12, 17,22,27	$y(x) := \cos\left(0, 4 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{12}\right) + \sin\left(0, 9 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{9}\right)$
3,8,13, 18,23,28	$y(x) := \sin\left(0, 4 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{10}\right) + \cos\left(0, 7 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{4}\right)$
4,9,14, 19,24,29	$y(x) := \sin(0, 4 \cdot 2 \cdot \pi \cdot x) + \cos\left(0, 7 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{7}\right)$
5,10,15, 20,25,30	$y(x) := \cos\left(0, 6 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{7}\right) + \cos\left(0, 9 \cdot 2 \cdot \pi \cdot x + \frac{\pi}{11}\right)$

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Градштейн И.С. Таблицы интегралов, сумм, рядов и произведений / И.С. Градштейн, И.М. Рыжик – М.: Физматгиз, 1963, – 1100 с.
2. Баскаков С.И. Радиотехнические цепи и сигналы / С.И. Баскакв. – М.: - Высшая школа, 2000. – 462с.
3. Дубнищев Ю.Н. Теория и преобразование сигналов в оптических системах / Ю.Н. Дубнищев. – Новосибирск: изд-во НГУ, 2004. – 275 с.
4. Стеценко О.А. Радиотехнические цепи и сигналы: учеб. / О.А. Стеценко. – М.: Высш. шк., 2007. – 432 с.
5. Садовский Г.А. Теоретические основы информационно-измерительной техники / Г.А. Садовский. – М.: - Высшая школа, 2008. – 478с.
6. Романенко С.В. Классификация математических моделей аналитических сигналов в форме пиков / С.В. Романенко, А.Г. Стремберг // Журнал аналитической химии. – 2000. – Т.55, N11, – с. 44-48.
7. Стремберг А.Г. Систематическое исследование элементарных моделей аналитических сигналов в виде пиков и волн / А.Г. Стремберг, С.В. Романенко, Э.С. Романенко // Журнал аналитической химии. – 2000. – Т55, N7, – с.687-697.
8. Сайфуллин Р.Т. Вейвлет-анализ в обработке сигналов аналитических приборов / Р.Т. Сайфуллин, А.А. Наумов // Известия Самарского научного центра Российской академии наук. – 2016. – Т. 18. № 1-2. – с. 412-415.

СОДЕРЖАНИЕ

Введение.....	3
Общие положения выполнения работ в рамках занятий	4
Работа №1. Общая теория радиотехнических сигналов	6
Вычисление мощности и энергии сигнала Ошибка! Закладка не определена.	
Задание 1 Ошибка! Закладка не определена.	
Вычисление косинуса угла между двумя сигналами Ошибка!	
Закладка не определена.	
Задание 2 Ошибка! Закладка не определена.	
Работа №2. Автокорреляционная и взаимнокорреляционная	
функция дискретных и непрерывных сигналов.....	10
Теоретические основы	18
ВКФ и АКФ дискретного сигнала Ошибка! Закладка не определена.	
АКФ и ВКФ непрерывных сигналов Ошибка! Закладка не определена.	
Задание	31
Работа №3. Критерии выбора частоты дискретизации..... Ошибка!	
Закладка не определена.	
Теоретические основы	Ошибка! Закладка не определена.
Реализация критериев выбора частоты дискретизации	
в Smath Studio	Ошибка! Закладка не определена.
Задание	Ошибка! Закладка не определена.
Работа №4. Корреляционная функция и преобразование Лорана	6
Теоретические основы	6
Задание 1	8
Задание 2	8
Работа №5. Дискретное преобразование Фурье	32
Основные понятия преобразования Фурье	
для дискретных сигналов.....	32
Реализация преобразования Фурье в Smath Studio	33
Задание	42
Работа №6. Частотно-временные способы анализа сигналов с	
применением преобразования Фурье.....	44
Кортко-временное преобразование Фурье в Smath Studio.....	50
Оконное преобразование Фурье	56
Реализация ОКПФ в Smath Studio.....	57

Задание	61
Работа №7. Разложение сигнала в ряд Фурье по гармоникам	10
Теоретические основы	10
Задание	11
Работа №8. Применение корреляционной функции и преобразования Фурье для исследования сигналов.....	113
Исследование сигнала Ошибка! Закладка не определена.	
Задание	49
Работа №9. Свертка и деконволюция	64
Теоретические основы	64
Свертка в Smath Studio	66
Деконволюция в частотной области.....	69
Реализация деконволюции в Smath Studio.....	70
Свертка сигнала с функцией Гаусса.....	72
Деконволюция сигнала с использованием априорной информации о форме импульсной характеристики и последовательным ручным уточнением ее параметров.....	77
Задание	83
Работа №10. Решение задач с использованием свертки	13
Теоретические основы	13
Задание	16
Работа №11. Построение нерекурсивных цифровых фильтров методом частотной выборки и их применение	85
Теоретические основы	85
Проектирование нерекурсивного фильтра в Smath Studio.....	86
Задание	100
Работа №12. Сглаживающий фильтр по n точкам..Ошибка! Закладка не определена.	
Теоретические основы	Ошибка! Закладка не определена.
Пример фильтрации	Ошибка! Закладка не определена.
Задание	Ошибка! Закладка не определена.
Работа №13. Непрерывное вейвлет-преобразование дискретных сигналов.....	102
Теоретические основы	102
Непрерывное вейвлет-преобразование в Smath Studio.....	104
Задание	111
Работа №14. Фильтрация сигналов с применением непрерывного вейвлет-преобразования.....	113

Фильтрация сигнала с применением непрерывного вейвлет-преобразования.....	113
Обратное непрерывное вейвлет преобразование	114
Импорт массивов из файлов Excel в Smath Studio	Ошибка!
Закладка не определена.	
Реализация алгоритма фильтрации с помощью НВП с использованием Smath Studio и Excel	115
Задание	120
Работа №15. Алгоритм Малла для вычисления быстрого вейвлет-преобразования	Ошибка! Закладка не определена.
Теоретические основы	Ошибка! Закладка не определена.
Алгоритм Малла при N=4.....	Ошибка! Закладка не определена.
Задание	Ошибка! Закладка не определена.
Работа №16. Разделение наложенных сигналов с помощью вейвлет-преобразования.....	Ошибка! Закладка не определена.
Теоретические основы	Ошибка! Закладка не определена.
Разделение наложенных пиков .	Ошибка! Закладка не определена.
Задание	Ошибка! Закладка не определена.
Библиографический список	121

Учебное издание

*БОЧКАРЕВ Андрей Владимирович
САЙФУЛЛИН Раухат Талгатович*

Преобразование измерительных сигналов

Редактор Г.В. Загребина

Выпускающий редактор Н.В. Беганова

Компьютерная верстка Е.Э. Парсаданян

Подп. в печать 25.11.19

Формат 60×84 1/16 . Бумага офсетная

Усл. п. л. 5,87. Уч.-изд. л. 2,35

Тираж 50 экз. Рег. № 551/09

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Самарский государственный технический университет»
443100 г. Самара, Молодогвардейская, 244. Главный корпус

Отпечатано в типографии
Самарского государственного технического университета
443100 г. Самара, ул. Молодогвардейская, 244. Корпус № 8