ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| ассистент | | Б. К. Акопян |
|---|---|---|
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

# ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

## Изучить методики вейвлет-анализа сигналов

по курсу: методы и устройства цифровой обработки сигналов

Вариант 2

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ ГР | 4711 | | Хасанов Б.Р. |
|---|---|---|---|
| | | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2020

## 1 Цель работы

Изучить методику вейвлет-анализа сигналов с использованием построения масштабно-временного вейвлет-спектра коэффициентов.

## 2 Краткие теоретические сведения

Вейвлет-анализ сигналов является альтернативой традиционному гармоническому анализу сигналов на основе преобразования Фурье (ПФ). Гармонический анализ, как известно, позволяет получить информацию о распределении амплитуд и фаз гармонических колебаний, по предположению составляющих в сумме исходный анализируемый процесс. Однако, несмотря на математическую безупречность теоретических положений Фурье-анализа, у него имеется два недостатка: во-первых, наличие даже сравнительно небольших по уровню помех искажает фазовый спектр процесса настолько, что становится невозможно проводить анализ временных соотношений между гармоническими компонентами, во-вторых, базис Фурье плохо приспособлен для обработки нестационарных сигналов, что проявляется, например, при реализации метода фильтрации сигналов на основе аппроксимации. Использование частотно-временного преобразования Фурье частично снимает эти проблемы, однако, по-прежнему, из анализа “выпадают” кратковременно существующие феномены в обрабатываемом процессе.

Вейвлет анализ сигналов основан на использовании специфической системы базисных функций – вейвлетов, – которые представляют собой множество функций φ(t,a,b), образующихся при масштабировании (коэффициент a) и временном сдвиге (b) функции материнского вейвлета Ф(t):

$$\varphi(t,a,b)=\frac{1}{\sqrt{a}}\Phi\left(\frac{t-b}{a}\right) \quad , \tag{1}$$

Непрерывное вейвлет-преобразованием входного сигнала x(t) длительности T имеет вид:

$$W(a,b)=\int_0^T x(t)\,\varphi(t,a,b)\,dt \quad , \tag{2}$$

получающаяся функция двух аргументов W(a,b) называется масштабно-временным вейвлет-спектром коэффициентов и содержит информацию о свойствах обрабатываемого сигнала, как и спектр преобразования Фурье, хотя и в менее очевидной для наглядной интерпретации форме.

Вейвлет-преобразование выборки дискретизированного процесса x n , n=0,1...N−1 может быть представлено следующим образом:

$$W_{i,k}=\sum_{n=0}^{N-1} x_n \frac{1}{\sqrt{a_i}} \Phi\left(\frac{\frac{n}{N-1}-b_k}{a_i}\right) \quad , \qquad (3)$$

где параметр времени нормирован по длительности выборки: t=n/(N−1), а параметры масштаба и сдвига задаются последовательностями $\{a_i\}$ и $\{b_k\}$. Результатом преобразования (3) будет двумерный массив W, размерности которого (I и K) могут быт заданы исследователем, исходя из требуемой детальности изображения картины вейвлет-спектра. Для параметра сдвига, который в формуле (3) также нормирован можно всегда задавать формулу для последовательных значений в виде $b_k = k/(K-1)$; диапазон изменения масштаба задается исходя из свойств функции материнского вейвлета: вейвлет-функции должны быть нетривиальными для a min и a max при n=0,1...N−1. Для нахождения величин $a_{min}$ и $a_{max}$ следует построить функции $\frac{1}{\sqrt{a_{min}}}\Phi\left(\frac{\frac{n}{N-1}-0}{a_{min}}\right)$ и $\frac{1}{\sqrt{a_{max}}}\Phi\left(\frac{\frac{n}{N-1}-\frac{1}{2}}{a_{max}}\right)$, соответсвующие самой короткой и самой протяженной по длительности вейвлет-функциям и убедиться по графикам, что они соответствуют невырожденным случаям. После определения $a_{min}$ и $a_{max}$, последовательность $\{a_i\}$ задается выражением $a_i=a_{min}+\frac{i}{I-1}(a_{max}-a_{min})$ .

Следует также подчеркнуть, что с целью сохранения энергетических соотношений в вейвлет-спектре для материнского вейвлета предварительно следует осуществить нормировку, так чтобы выполнялось равенство $\int_{-\infty}^{\infty}\Phi^2(t)dt=1$

Полученный спектр коэффициентов принято представлять в виде черно-белого или цветного изображения, каждый элемент (пиксель) которого имеет характеристику (яркость либо цвет), однозначно связанную с величиной соответствующего элемента массива W i,k . При построении черно-белого изображения следует осуществить нормировку массива W: минимальному по значению элементу (с учетом знака, т. е. самому большому по модулю, с отрицательным знаком) ставится в соответствие значение 0, максимальному – десятичное число 255.

**3 Ход работы**

Программа написана на языке программирования python 3

Стандартно в начале главной функции импортируются нужные библиотеки

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

```
from numpy import pi, cos, sin
from scipy.integrate import quad
```

Задаём общие характеристики сигналов

```
signal_length = 3000
signal_discretization = 1/1000
signal_range = np.linspace(0, signal_length - 1, signal_length)
signal_time = signal_range * signal_discretization
```

Формируем простой гармонический сигнал (результат на рисунке 3.1, верхний график), сумму двух гармонических сигналов(результат на рисунке 3.2, верхний график), гармонический сигнал со скачкообразным изменением частоты(результат на рисунке 3.3, верхний график), функция, соответствующая материнскому вейвлету по заданию(результат на рисунке 3.4, верхний график

```
simple_signal_freq = 10
simple_harmonic_signal = cos(simple_signal_freq * 2 * signal_time * pi/4)
sum_of_harm_signals = (cos(7 * pi * signal_time)
                       + sin(2 * pi * signal_time))
first_abrupt_freq = 3
second_abrupt_freq = 10
abrupt_change_signal = []
for number in range(0, signal_length):
    if number > signal_length/3:
        abrupt_change_signal.append(2 * sin(2 * pi * signal_time[number]
                                    * first_abrupt_freq))
    else:
        abrupt_change_signal.append(cos(2 * pi * signal_time[number]
                                    * second_abrupt_freq))
abrupt_change_signal = np.array(abrupt_change_signal)
mother_wavelet = mother_wavelet_func(signal_time)
```

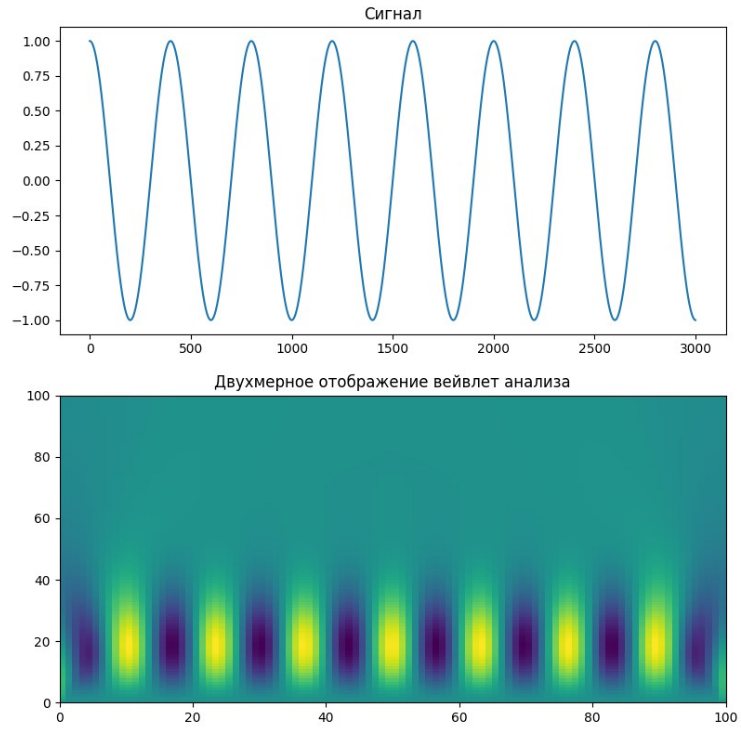


Рисунок 3.1 – Вейвлет анализ простого гармонического сигнала

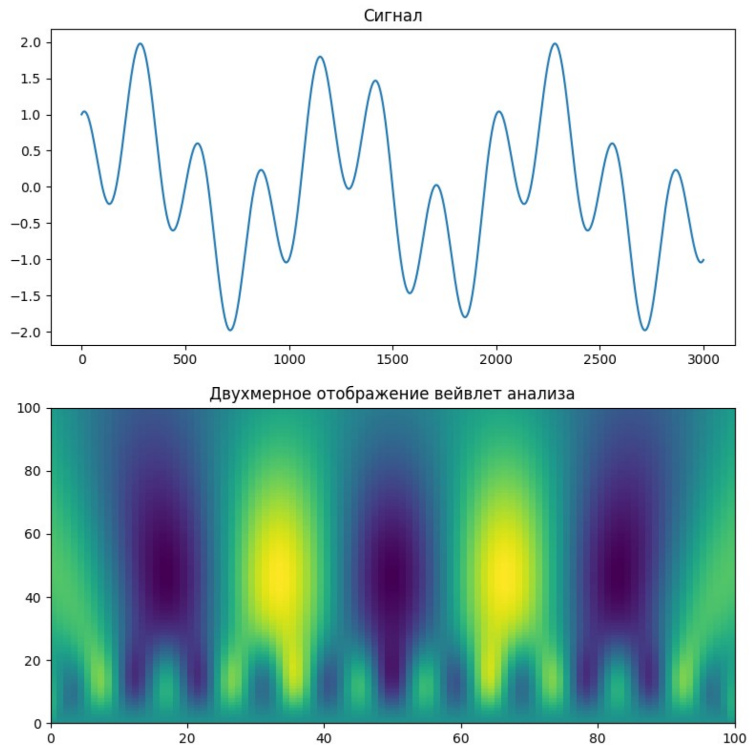


Рисунок 3.2 – Вейвлет анализ суммы двух гармонических сигналов

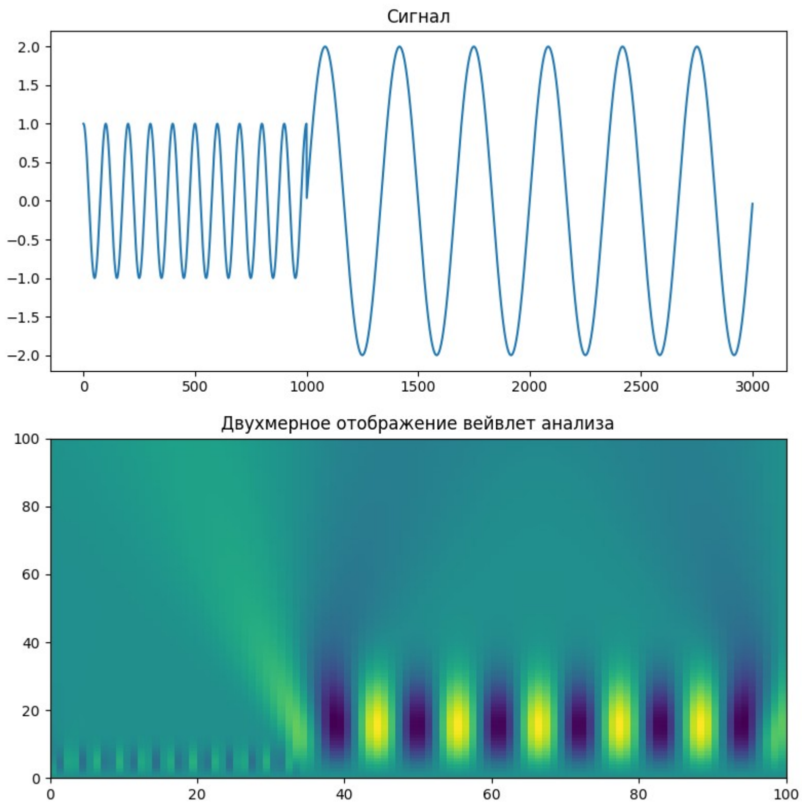


Рисунок 3.3 – Вейвлет анализ  гармонического сигнала со скачкообразным изменением частоты

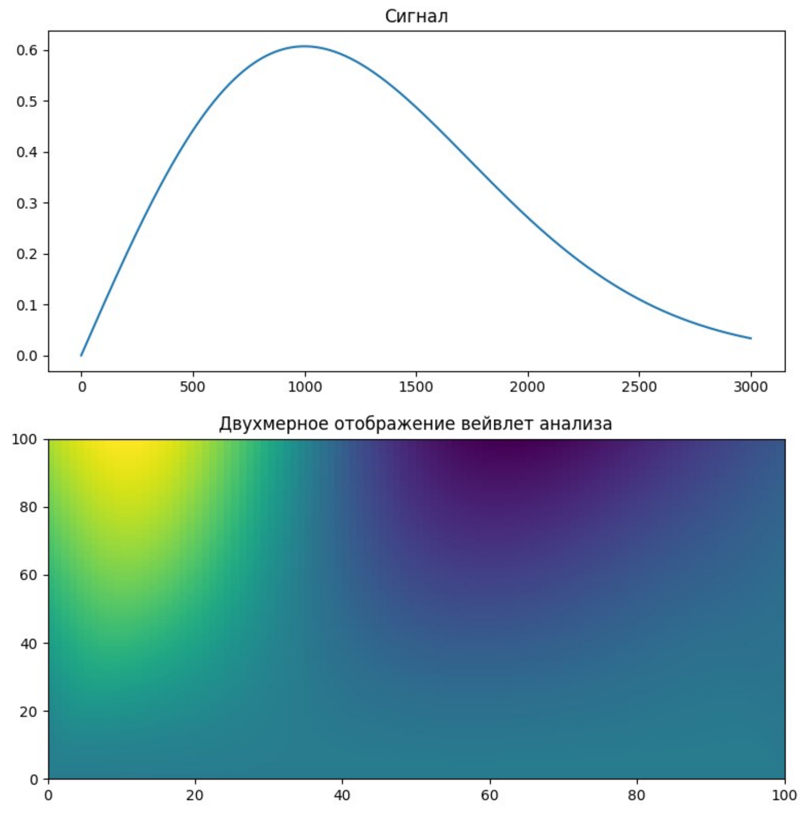


Рисунок 3.4 – Вейвлет анализ функции, соответствующей материнскому вейвлету

Добавляем гауссовский шум к сгенерированным ранее сигналам. Результаты на рисунках с 3.5 до 3.8, верхние графики

```
white_noise = np.random.normal(0, 1, signal_length)
simple_harmonic_signal_noised = simple_harmonic_signal + white_noise
sum_of_harm_signals_noised = sum_of_harm_signals + white_noise
abrupt_change_signal_noised = abrupt_change_signal + white_noise
mother_wavelet_noised = mother_wavelet + white_noise
```

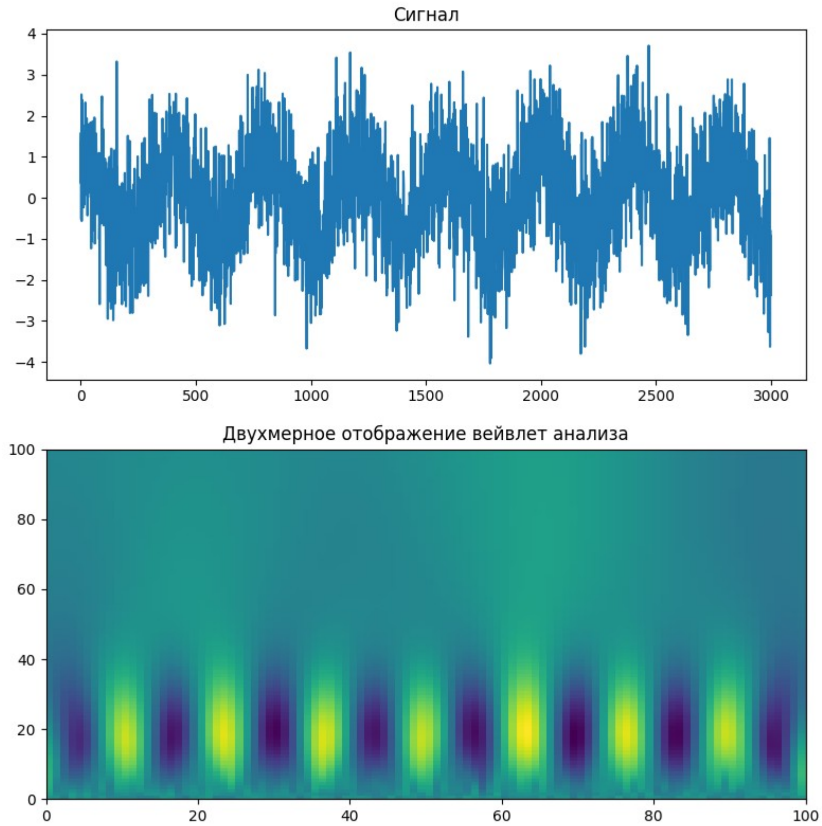


Рисунок 3.5 – Вейвлет анализ зашумлённого простого гармонического сигнала

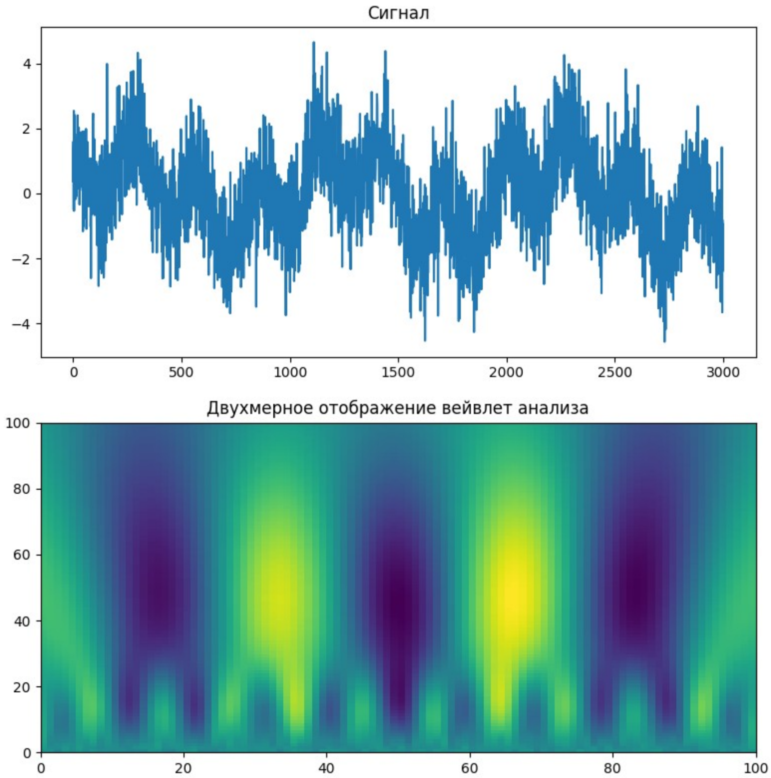


Рисунок 3.6 – Вейвлет анализ суммы двух гармонических сигналов с шумом

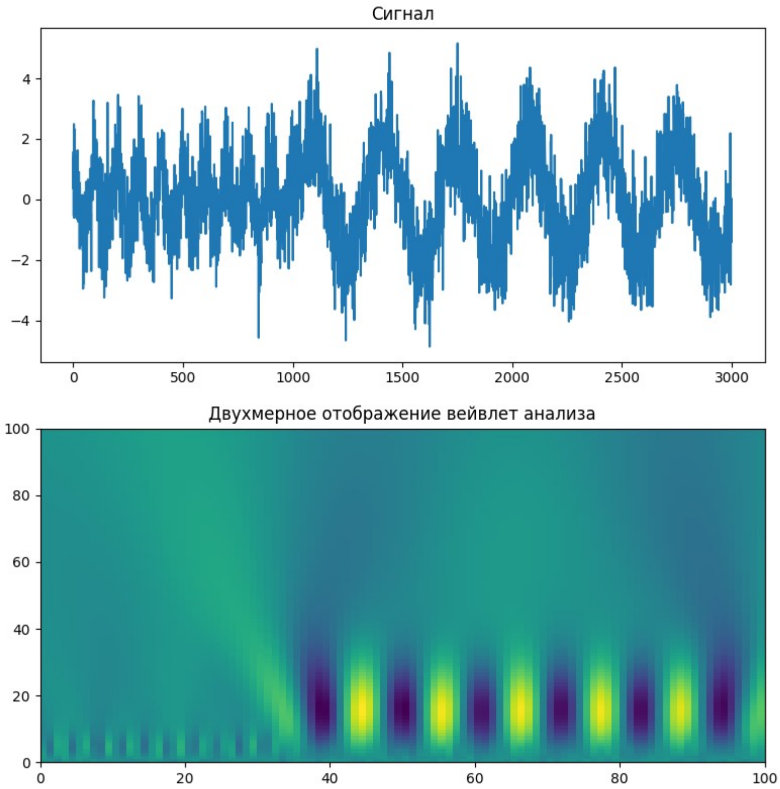


Рисунок 3.7 – Вейвлет анализ зашумлённого гармонического сигнала со скачкообразным изменением частоты

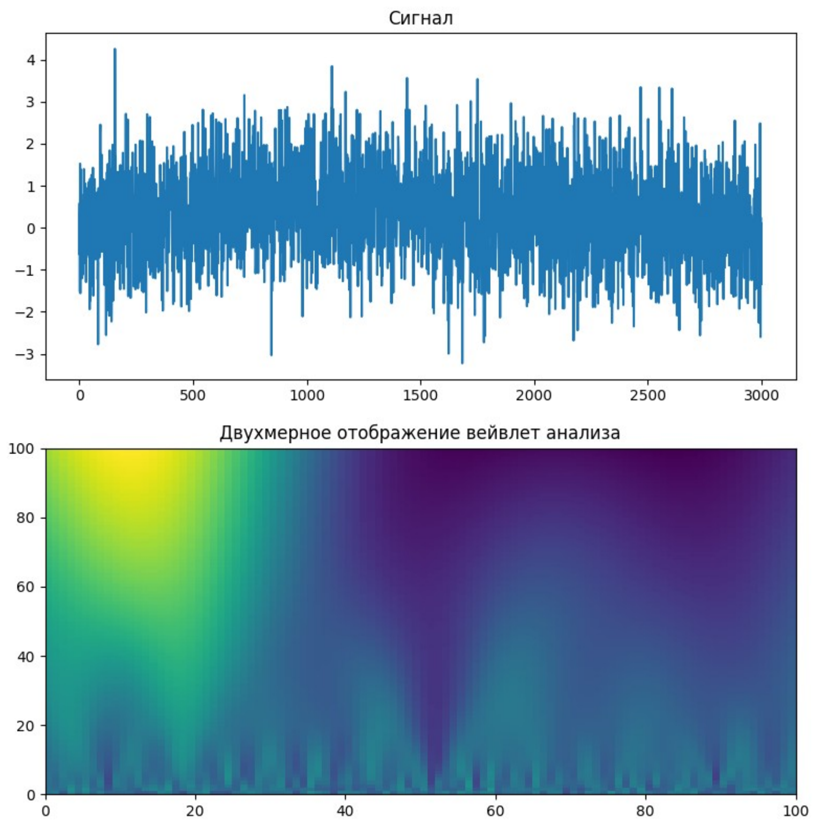


Рисунок 3.8 – Вейвлет анализ функции, соответствующей материнскому вейвлету с шумом

Задаём разрешение вейвлета и нормализуем материнский вейвлет

```
wavelet_column = 100
wavelet_row = 100
wavelet_length = 1000
normalization_check = quad(lambda x: mother_wavelet_func(x)**2, -np.inf,
                    np.inf)[0]
if normalization_check < 0.99 or 1.01 < normalization_check:
    normalization_coeff = sum((mother_wavelet_func(
        np.linspace(0, wavelet_length, wavelet_length + 1)))**2)
    normalizated_mother_wavelet = (
        lambda time: ((mother_wavelet_func(time))
                / np.sqrt(normalization_coeff)))
else:
    normalizated_mother_wavelet = mother_wavelet_func()
```

Задаём границы диапазона варьирования вейвлет функции и выводим график вейвлет-функции при заданных параметрах минимума и максимума. Результаты на рисунке 3.9

```
scale_min = 1/signal_length
scale_max = 0.14
scale_min_graph = (1/np.sqrt(scale_min)
            * normalizated_mother_wavelet(
                ((signal_range / signal_length) - 0.5)/scale_min))
```

```
scale_max_graph = (1/np.sqrt(scale_max)
                   * normalizated_mother_wavelet(
                       ((signal_range / signal_length) - 0.5)/scale_max))
wavelet_scale = (scale_min + (np.linspace(0, wavelet_column - 1,
                                           wavelet_column)
                              / (wavelet_column - 1))
                 * (scale_max - scale_min))
wavelet_shift = np.linspace(0, wavelet_row - 1, wavelet_row)/(wavelet_row
                                                               - 1)
```

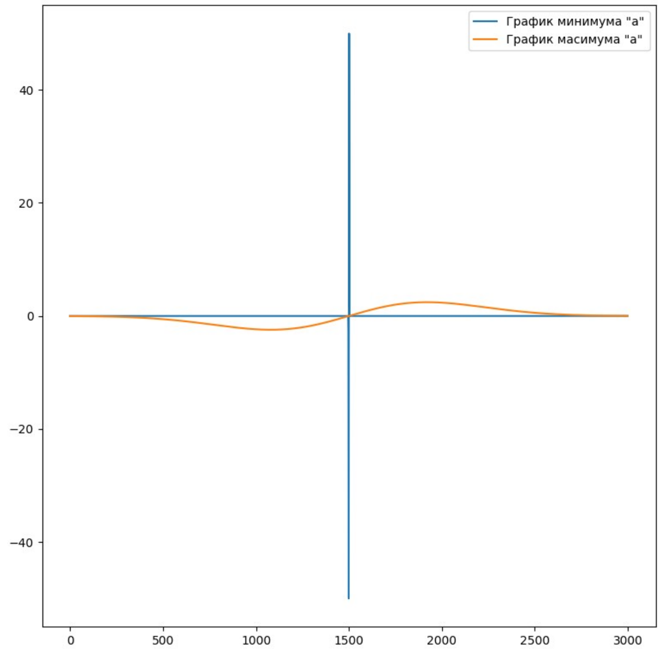


Рисунок 3.9 – График максимума и минимума масштаба вейвлет функции

Формируем вейвлет функцию

```
wavelet = (
    lambda row_number, column_number: (
        (1/np.sqrt(wavelet_scale[column_number]))
        * normalizated_mother_wavelet(
            (signal_range / (signal_length - 1)
             - wavelet_shift[row_number])
            / wavelet_scale[column_number])))
```

Наконец, применяем вейвлет анализ на сигналах. Результат на рисунках с 3.1 до 3.8, нижние графики

```
for signal in signal_list:
    wavelet_transform_to_plot = wavelet_analisis(
        wavelet, signal, wavelet_scale, wavelet_shift)
    wavelet_plot(signal, wavelet_transform_to_plot)
```

Функция wavelet_plot выглядит следующим образом. Последние два действия перед возвращением переменной нормируют массив таким образом, чтобы любой его элемент находился в диапазоне от 0 до 255:

```
def wavelet_analisis(wavelet_function, signal_to_analisis, wavelet_scale_arr,
                     wavelet_shift_arr):
    """Вейвлет анализ над переданным сигналом с помощью переданной вейвлет
    функции """
    import numpy as np

    wavelet_transform = []
    for column in range(0, len(wavelet_scale_arr)):
        wavelet_transform.append([])
        for row in range(0, len(wavelet_shift_arr)):
            wavelet_transform[column].append(
                sum(signal_to_analisis * wavelet_function(row, column)))
    wavelet_transform = np.array(wavelet_transform)

    wavelet_transform_intermediate = (wavelet_transform
                                      - np.min(wavelet_transform))
    wavelet_transform_normalizated = (wavelet_transform_intermediate
                                      / wavelet_transform_intermediate.max()
                                      * 255)

    return wavelet_transform_normalizated
```

## 4 Выводы

В рамках данной работы мной была построена нормированная вейвлет-функция. На практике убедился, что вейвлет анализ позволяет узнать не только частоту сигнала, но и увидеть его положение во временной области, что можно наблюдать на всех рисунках с изображением вейвлет анализа (рисунки с 3.1 по 3.8). Так же убедился, что вейвлет анализ позволяет эффективно отделять шум от сигналов даже при дисперсии  в несколько раз превышающую амплитуду чистого сигнала. На нижних частях графиков вейвлет преобразований можно рассмотреть шум, в отличие от полезных сигналов он непереодичен.

**Список источников**

1. Цифровая обработка сигналов: учебное пособие / В.А. Сериков, В.Р. Луцив; С.-Петерб. гос. ун-т аэрокосм. приборостроения. - СПб: Изд-во ГУАП, 2014. – 110 с. [библиотечный шифр 621.391 С32]

2. Лекция 5: Вейвлеты в цифровой обработке сигналов О.О. Жаринов, ГУАП, 14 октября 2020г. [Электронный ресурс]. – Режим доступа: https://lms.guap.ru/new/mod/bigbluebuttonbn/view.php?id=26256# – Загл. с экрана. (Дата обращения 6.12.2020г.).

3. Лекция 8. Вейвлет-анализ. [Электронный ресурс]. – Режим доступа: http://matematika.phys.msu.ru/files/stud_spec/270/MM_lec8.pdf – Загл. с экрана. (Дата обращения 6.12.2020г.).

Приложение А – Программа

```
def mother_wavelet_func(time):
    """returns vivlet function"""
    from numpy import exp

    return time * exp(- (time**2)/2)


def wavelet_plot(analisated_signal, wavelet_analisis_result):
    """Plot 2d and 3d graph of wavelet analisis result"""
    import matplotlib.pyplot as plt
    # import numpy as np

    fig = plt.figure()
    signal_ax = fig.add_subplot(211)
    signal_ax.plot(analisated_signal)
    signal_ax.set_title("Сигнал")
    two_d_wavelet_ax = fig.add_subplot(212)
    two_d_wavelet_ax.pcolormesh(wavelet_analisis_result)
    two_d_wavelet_ax.set_title("Двухмерное отображение вейвлет анализа")
    # fig_3d = plt.figure()
    # three_d_wavelet_ax = fig_3d.add_subplot(111, projection='3d')
    # x, y = np.meshgrid(np.arange(wavelet_analisis_result.shape[0]),
    #                    np.arange(wavelet_analisis_result.shape[1]))
    # three_d_wavelet_ax.set_title("Трёхмерное отображение вейвлет анализа")
    # three_d_wavelet_ax.plot_surface(x, y, wavelet_analisis_result,
    #                                 cmap='viridis', rcount=20, ccount=20)


def wavelet_analisis(wavelet_function, signal_to_analisis, wavelet_scale_arr,
                     wavelet_shift_arr):
    """Вейвлет анализ над переданным сигналом с помощью переданной вейвлет
    функции """
    import numpy as np

    wavelet_transform = []
    for column in range(0, len(wavelet_scale_arr)):
        wavelet_transform.append([])
        for row in range(0, len(wavelet_shift_arr)):
            wavelet_transform[column].append(
                sum(signal_to_analisis * wavelet_function(row, column)))
    wavelet_transform = np.array(wavelet_transform)

    wavelet_transform_intermediate = (wavelet_transform
                                      - np.min(wavelet_transform))
    wavelet_transform_normalizated = (wavelet_transform_intermediate
                                      / wavelet_transform_intermediate.max()
                                      * 255)

    return wavelet_transform_normalizated
```

```
def main():
    import numpy as np
    import matplotlib.pyplot as plt
    from numpy import pi, cos, sin
    from scipy.integrate import quad

    # Задаём общие характеристики сигналов
    signal_length = 3000
    signal_discretization = 1/1000
    signal_range = np.linspace(0, signal_length - 1, signal_length)
    signal_time = signal_range * signal_discretization

    # Формируем сигналы
    simple_signal_freq = 10
    simple_harmonic_signal = cos(simple_signal_freq * 2 * signal_time * pi/4)
    sum_of_harm_signals = (cos(7 * pi * signal_time)
                           + sin(2 * pi * signal_time))
    first_abrupt_freq = 3
    second_abrupt_freq = 10
    abrupt_change_signal = []
    for number in range(0, signal_length):
        if number > signal_length/3:
            abrupt_change_signal.append(2 * sin(2 * pi * signal_time[number]
                                        * first_abrupt_freq))
        else:
            abrupt_change_signal.append(cos(2 * pi * signal_time[number]
                                        * second_abrupt_freq))
    abrupt_change_signal = np.array(abrupt_change_signal)
    mother_wavelet = mother_wavelet_func(signal_time)

    # Добовляем помеху к сигналу
    white_noise = np.random.normal(0, 1, signal_length)
    simple_harmonic_signal_noised = simple_harmonic_signal + white_noise
    sum_of_harm_signals_noised = sum_of_harm_signals + white_noise
    abrupt_change_signal_noised = abrupt_change_signal + white_noise
    mother_wavelet_noised = mother_wavelet + white_noise

    signal_list = [simple_harmonic_signal, sum_of_harm_signals,
                   abrupt_change_signal, mother_wavelet,
                   simple_harmonic_signal_noised,
                   sum_of_harm_signals_noised, abrupt_change_signal_noised,
                   mother_wavelet_noised]

    # Проводим нормализацию для вейвлет функции
    wavelet_column = 100
    wavelet_row = 100
    wavelet_length = 1000
    normalization_check = quad(lambda x: mother_wavelet_func(x)**2, -np.inf,
                               np.inf)[0]
    if normalization_check < 0.99 or 1.01 < normalization_check:
        normalization_coeff = sum((mother_wavelet_func(
            np.linspace(0, wavelet_length, wavelet_length + 1)))**2)
```

```
        normalizated_mother_wavelet = (
            lambda time: ((mother_wavelet_func(time))
                          / np.sqrt(normalization_coeff)))
    else:
        normalizated_mother_wavelet = mother_wavelet_func()

    # Подбираем значения для диапазона изменения масштаба
    scale_min = 1/signal_length
    scale_max = 0.14
    scale_min_graph = (1/np.sqrt(scale_min)
                   * normalizated_mother_wavelet(
                       ((signal_range / signal_length) - 0.5)/scale_min))
    scale_max_graph = (1/np.sqrt(scale_max)
                   * normalizated_mother_wavelet(
                       ((signal_range / signal_length) - 0.5)/scale_max))
    wavelet_scale = (scale_min + (np.linspace(0, wavelet_column - 1,
                                   wavelet_column)
                              / (wavelet_column - 1))
                  * (scale_max - scale_min))
    wavelet_shift = np.linspace(0, wavelet_row - 1, wavelet_row)/(wavelet_row
                                                   - 1)

    # Рисуем гарфики минимума и макисимума диапазона
    plt.figure()
    plt.plot(scale_min_graph, label='График минимума "a"')
    plt.plot(scale_max_graph, label='График масимума "a"')
    plt.legend()

    # Формируем вейвлет функцию для работы
    wavelet = (
        lambda row_number, column_number: (
            (1/np.sqrt(wavelet_scale[column_number]))
            * normalizated_mother_wavelet(
                (signal_range / (signal_length - 1)
                 - wavelet_shift[row_number])
                / wavelet_scale[column_number])))

    # Вейвлет анализ всех сгенерированных нами сигналов
    for signal in signal_list:
        wavelet_transform_to_plot = wavelet_analisis(
            wavelet, signal, wavelet_scale, wavelet_shift)
        wavelet_plot(signal, wavelet_transform_to_plot)

    plt.show()


if __name__ == "__main__":
    main()
```