Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С. П. Королева»
(Самарский университет)
Министерство науки и высшего образования Российской Федерации

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

Оптоинформационные технологии и системы

Численная реализация оптического преобразования Фурье на основе быстрого преобразования Фурье

«Отчет по лабораторной работе №2»

Вариант № 9

Выполнил студент: Белоусов А. А.
Группа: 6409-010302D
Преподаватель: Кириленко М.С.

Самара 2019

ЗАДАНИЕ

1. Реализовать одномерное финитное преобразование Фурье с помощью применения алгоритма БПФ.

2. Построить график гауссова пучка e^{-x^2}. Здесь и далее для каждого графика следует строить отдельно графики амплитуды и фазы. Амплитуда находится как модуль каждого значения функции, фаза – как аргумент (или с помощью функции atan2).

3. Убедиться в правильности реализации преобразования, подав на вход гауссов пучок e^{-x^2} – собственную функцию преобразования Фурье. На выходе тоже должен получиться гауссов пучок (построить график на правильной области определения $[-\tilde{b}, \tilde{b}]$). Рекомендуемая входная область: $[-a, a] = [-5, 5]$.

4. Реализовать финитное преобразование Фурье стандартным методом численного интегрировани (например, методом прямоугольников). Важно: необходимо вычислить интеграл для каждого дискретного значения u, чтобы получить результат в виде вектора. На вход преобразования вновь следует подавать гауссов пучок.

5. Построить результаты двух разных реализаций преобразования на одном изображении (одно для амплитуды, одно для фазы) и убедиться, что они совпадают.

6. Используя первую реализацию преобразования, подать на вход световое поле, отличное от гауссова пучка, в соответствии со своим вариантом. Построить графики самого пучка и результата преобразования.

7. Рассчитать аналитически результат преобразования своего варианта поля и построить график на одной системе координат с результатом, полученным в предыдущем пункте.

АЛГОРИТМ РЕАЛИЗАЦИИ ОПТИЧЕСКОГО ФИНИТНОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ ЧЕРЕЗ ИСПОЛЬЗОВАНИЕ БПФ

1. Провести дискретизацию входной функции $f(x)$ в вектор f с размерностью N.

2. Дополнить вектор f и слева, и справа необходимым числом нулей до размерности M.

3. Разбить вектор f на две половины и поменять их местами.

4. Выполнить БПФ от f и умножить результат на шаг h_x , получив вектор F.

5. Разбить вектор F на две половины и поменять их местами.

6. «Вырезать» центральную часть вектора F, оставив N элементов.

СОДЕРЖАНИЕ

1 Реализация одномерного финитного преобразования Фурье

Одномерное финитное преобразование Фурье записывается следующим образом:

$$F_a(u) = \Phi_a\left[f(x)\right](u) = \int\limits_{-a}^{a} f(x) e^{-2\pi i x u}\, dx, \tag{1}$$

где $f(x)$ – финитная функция, $F_a(u)$ – спектр, Φ_a – оператор финитного преобразования Фурье.

Здесь и далее в работе используется язык программирования Python.

Данное преобразование реализуем с использованием функций из пакета fft библиотеки numpy:

```
def fourier_builitn(f, step: float, xs: List[float]) -> List[float]:
    fs = list(map(lambda x: f(x), xs))
    fs = np.fft.fftshift(fs)
    fourier = list(map(lambda x: x * step, np.fft.fft(fs)))
    return np.fft.fftshift(fourier)
```

2 Построение графика гауссова пучка

Гауссов пучок определяется следующим образом:

$$f(x) = e^{-x^2} \tag{2}$$

Напишем функцию, для отображения графиков функции на заданном промежутке:

```
def plot_function(f, a, b, step_count, fig):
    step = (b - a) / step_count
    xs = np.arange(a, b, step)
    ys = f(xs)
    plt.figure(f"Function {fig}")
    plt.grid()
    plt.plot(xs, ys)
```

На рисунках ниже представлены результаты работы программы.

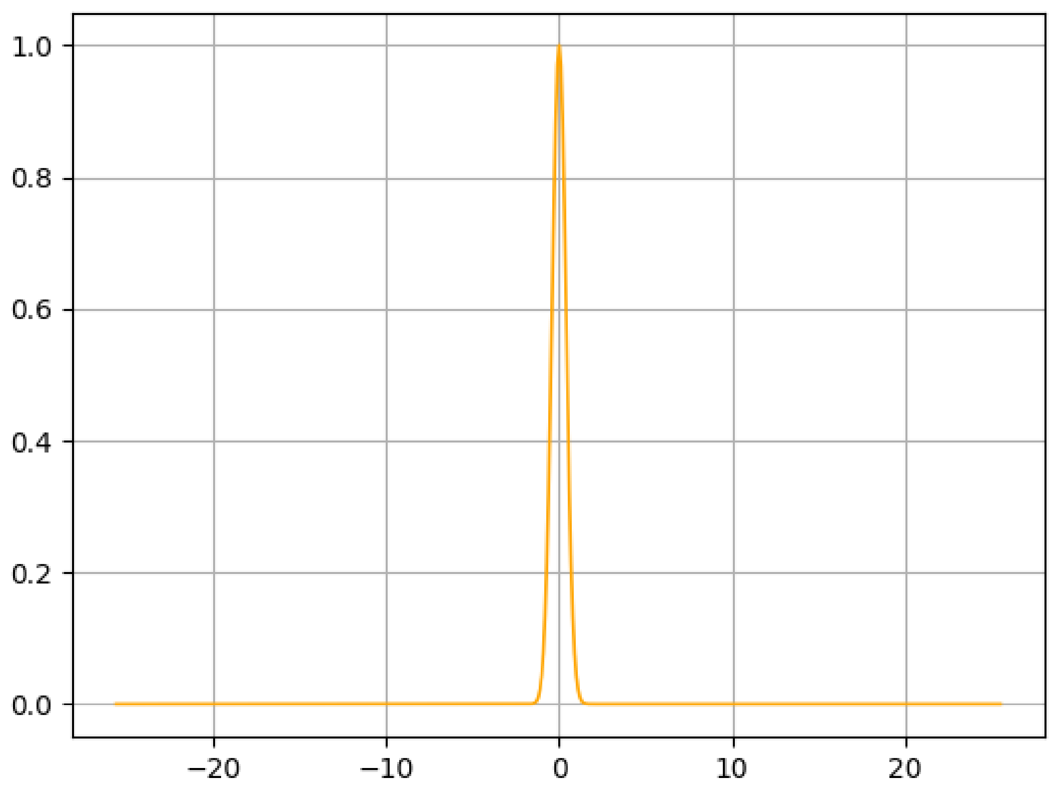

Рисунок 1 – График функции гауссова пучка

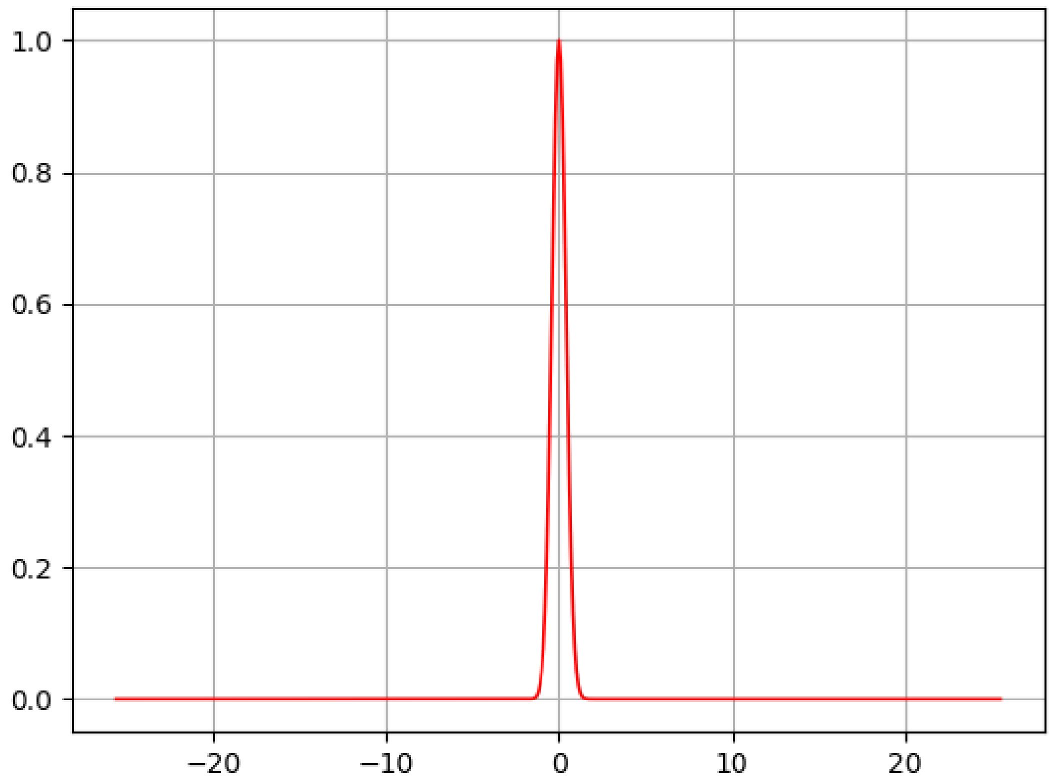

Рисунок 2 – Амплитуда функции гауссова пучка

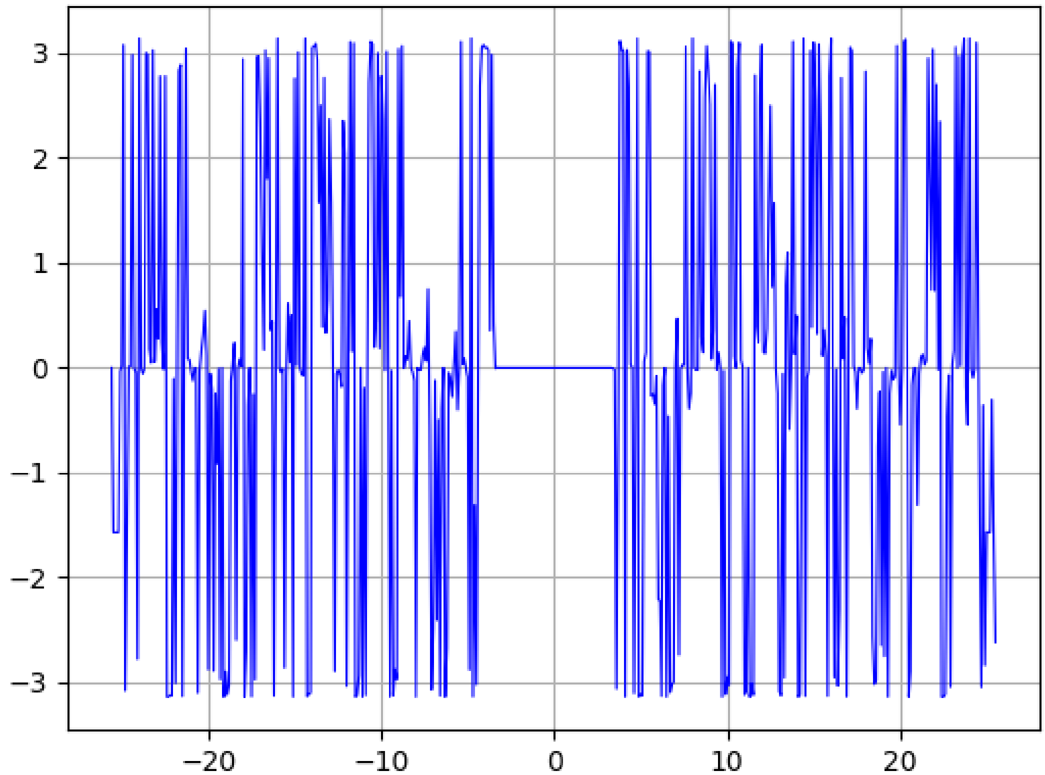

Рисунок 3 – Фаза функции гауссова пучка

Для проверки корректности работы программы наложим график исходной функции на график. Амплитуда Гауссова пучка после преобразования должна совпадать с исходной функцией.

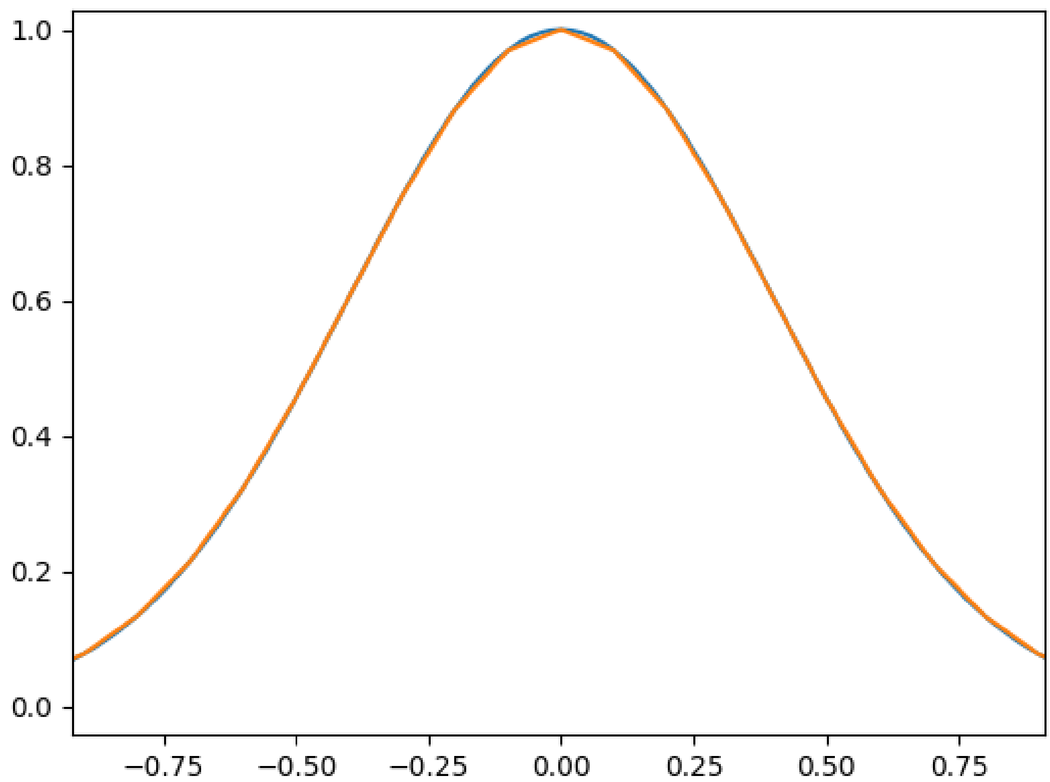

Рисунок 4 – Объединенный график исходной функции и её амплитуды

Как видно из рисунка 4, графики совпадают с незначительным отклонениями, следовательно преобразование реализовано верно.

3 Численное финитное преобразование Фурье

Для реализации численного финитного преобразования воспользуемся методом прямоугольников при численном вычислении определенного интеграла. Метод прямоугольников определяется следующим образом:

$$\int\limits_a^b f(x)dx = \sum_{k=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}(x_i - x_{i-1})\right) \tag{3}$$

Реализуем численное финитное преобразование Фурье используя метод прямоугольников. Конечная программа представлена ниже.

```
def calculus_exp(u: float, x: float) -> float:
    """calculus_exp
    Returns e^{-2πiux} from fourier transform.
    :param u: u value
    :type u: float
    :param x: x value
    :type x: float
    :rtype: float
    """
    return np.exp(-2 * np.pi * 1j * u * x)


def calculus_fourier(f, step: float, xs: [float],
                     us: List[float]) -> List[float]:
    fourier = []
    for u in us:
        newx = 0
        for x in xs:
            newx += f(x) * calculus_exp(u, x)
        newx *= step
        fourier.append(newx)
    return fourier
```

Сравним написанную реализацию численного финитного преобразования Фурье с финитным преобразованием Фурье через БПФ. На рисунках 5 и 6 показано сравнения реализованных методов.

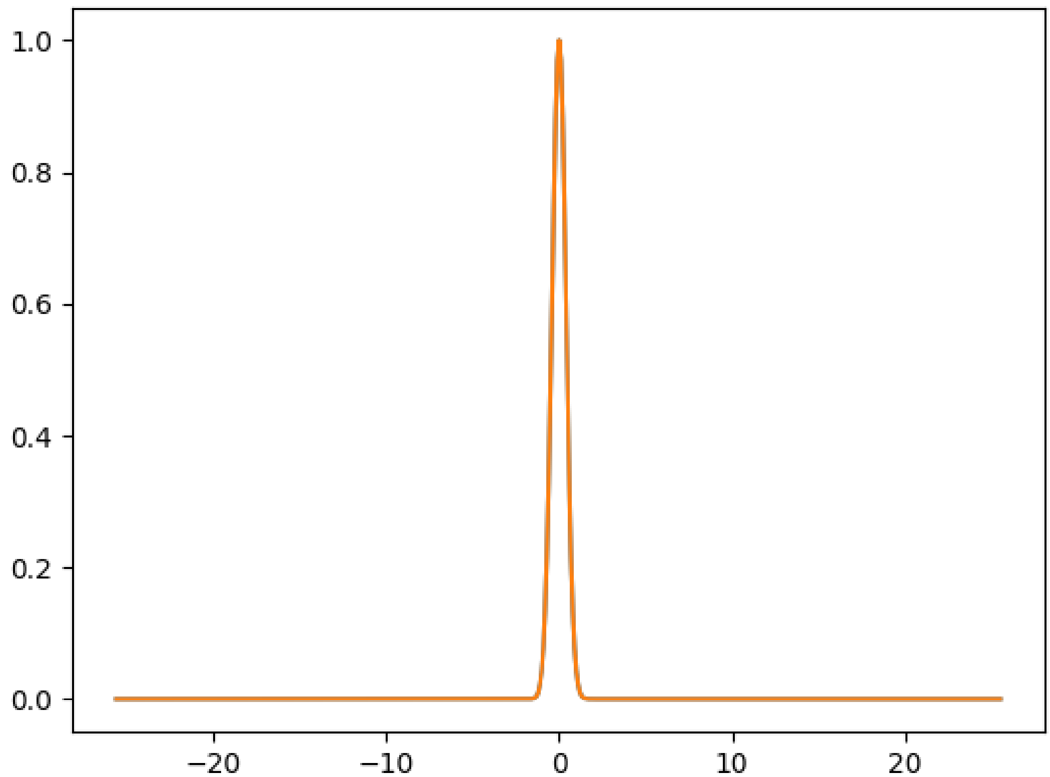

Рисунок 5 – Сравнение численного преобразование Фурье и Фурье через БПФ

При достаточном приближении видно, что значения преобразований совпадают, но есть небольшая погрешность вычисления численным методом. Однако для нашей задачи это не сыграет роли.

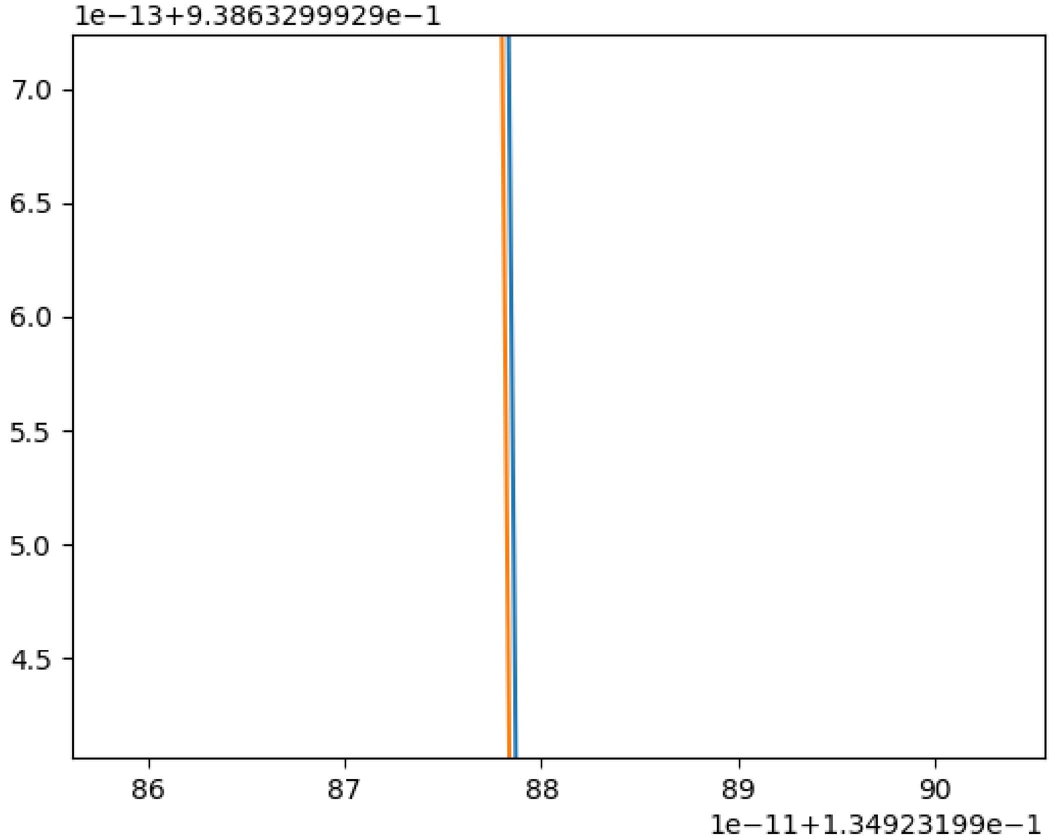

Рисунок 6 – Сравнение численного преобразование Фурье и Фурье через БПФ в приближении

4 Построение функции по варианту

Дана функция

$$f(x) = e^{2\pi i x} + e^{-5\pi i x} \tag{4}$$

Построим для функции (4) графики преобразования Фурье через БПФ. На рисунках ниже показаны амплитуда и фаза исходной и преобразованной функции.

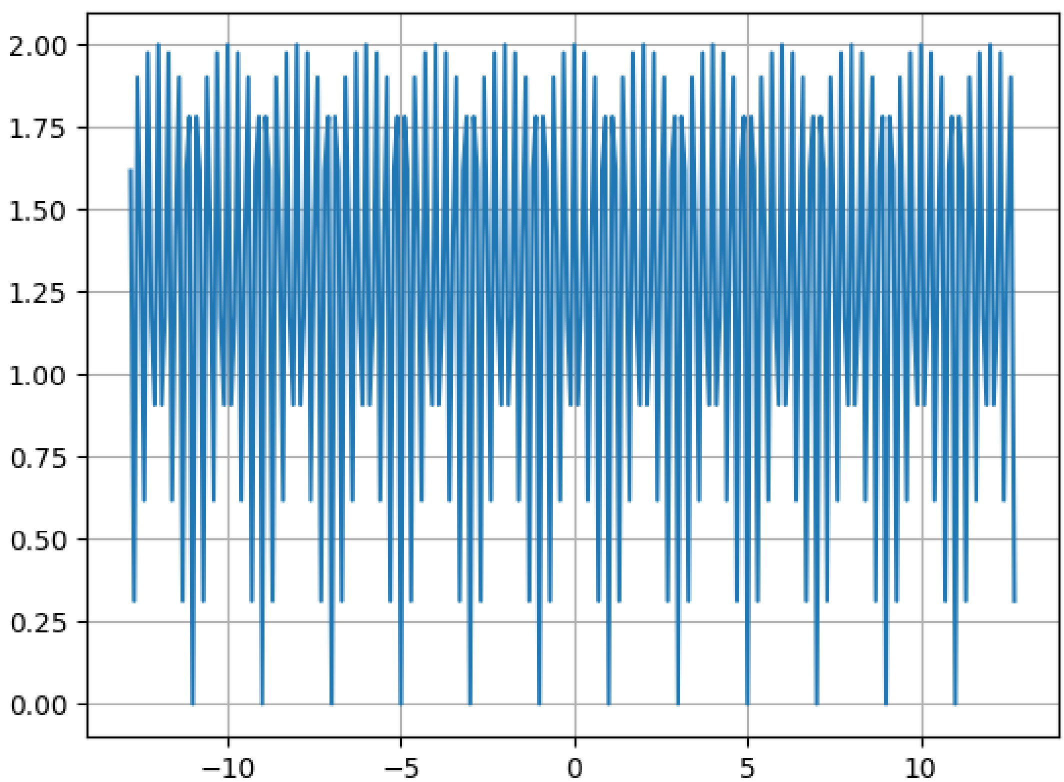

Рисунок 7 – Амплитуда исходной функции

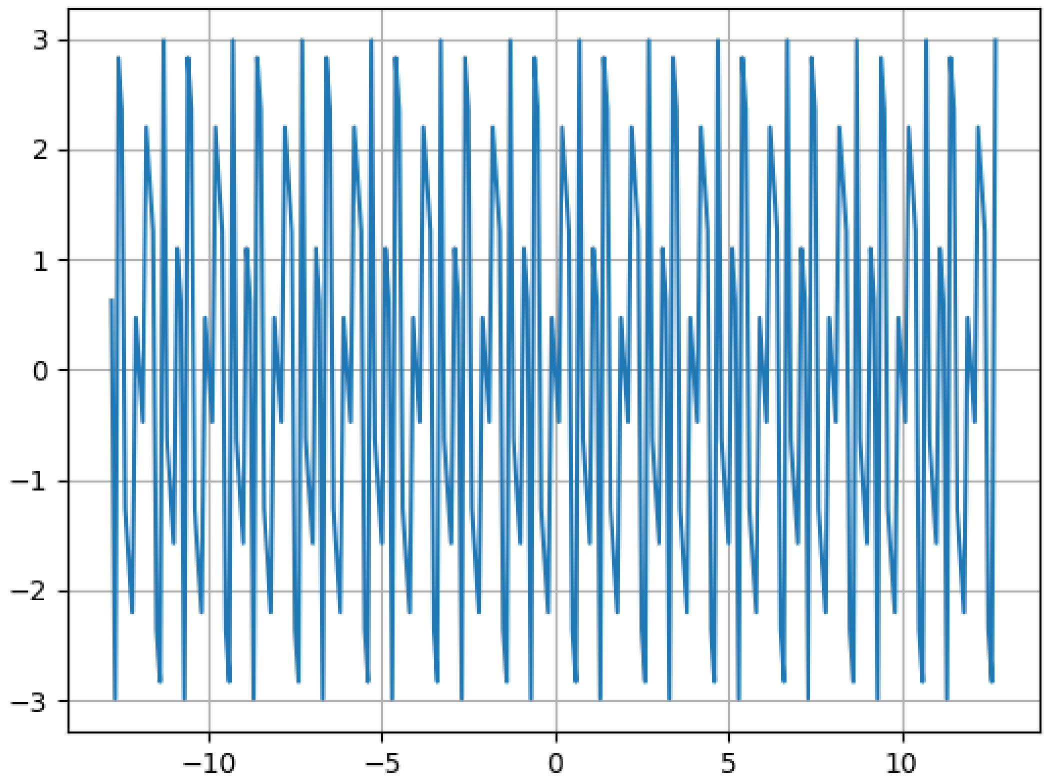

Рисунок 8 – Фаза исходной функции

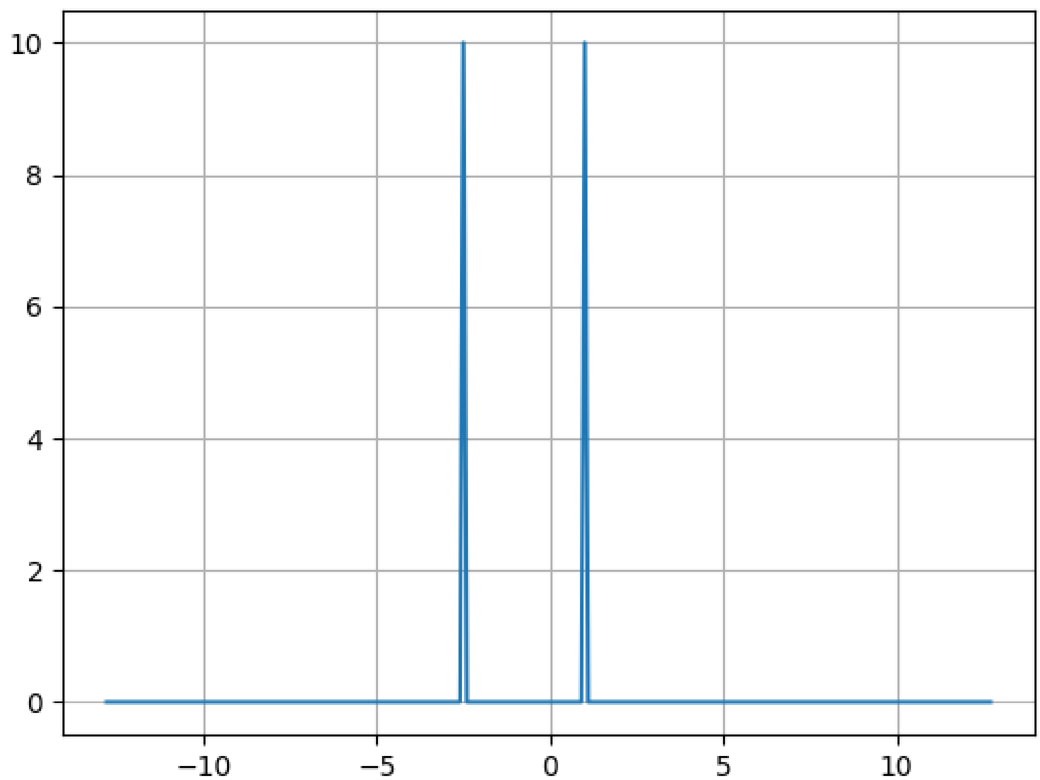

Рисунок 9 – Амплитуда преобразованной функции

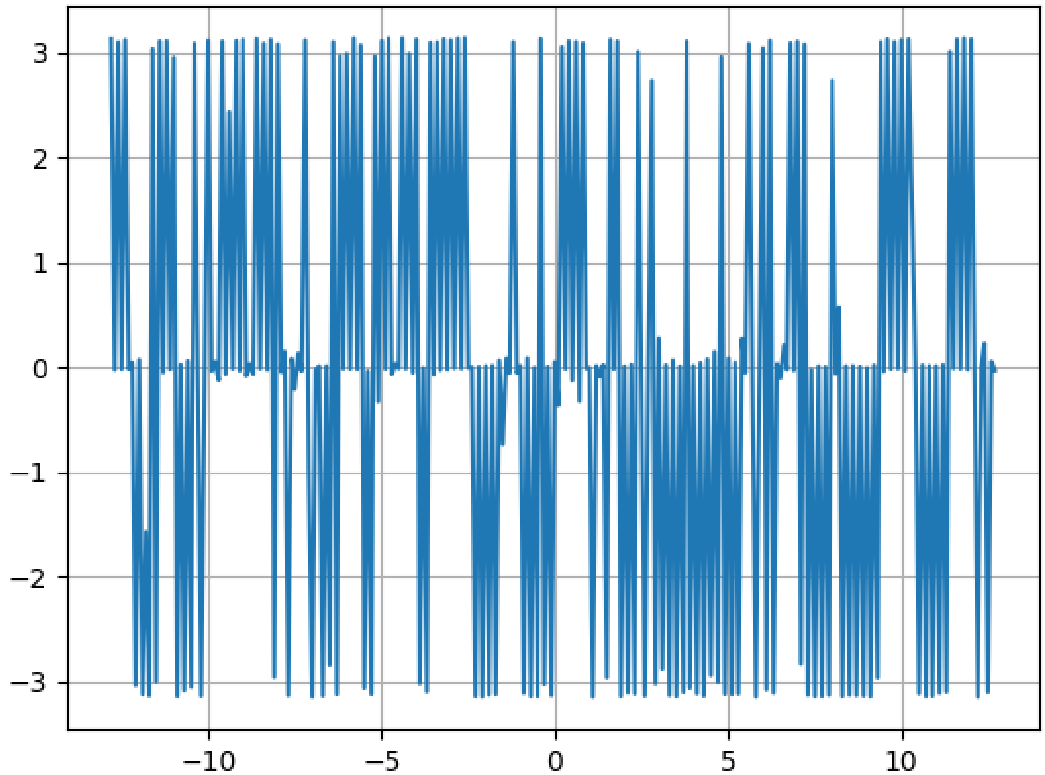

Рисунок 10 – Фаза преобразованной функции

5 Построение аналитической функции по варианту

Для функции (4) найдем аналитическое преобразование Фурье $(\mathfrak{F}(f(x)))$.

$$e^{2\pi ix}+e^{-5\pi ix}\xrightarrow{\mathfrak{F}}\int\limits_a^b\left(e^{2\pi ix}+e^{-5\pi ix}\right)e^{-2\pi ix\xi}dx=\int\limits_a^b e^{2\pi ix}e^{-2\pi ix\xi}dx+\int\limits_a^b e^{-5\pi ix}e^{-2\pi ix\xi}dx=$$

$$=\int\limits_a^b e^{-2\pi ix(\xi-1)}dx+\int\limits_a^b e^{-2\pi ix\left(\xi+\frac{5}{2}\right)}dx=\left.\frac{e^{-2\pi ix(\xi-1)}}{-2\pi i(\xi-1)}\right|_a^b+\left.\frac{e^{-2\pi ix\left(\xi+\frac{5}{2}\right)}}{-2\pi i\left(\xi+\frac{5}{2}\right)}\right|_a^b=$$

$$=\frac{e^{-2\pi ib(\xi-1)}-e^{-2\pi ia(\xi-1)}}{-2\pi i(\xi-1)}+\frac{e^{-2\pi ib\left(\xi+\frac{5}{2}\right)}-e^{-2\pi ia\left(\xi+\frac{5}{2}\right)}}{-2\pi i\left(\xi+\frac{5}{2}\right)}$$

Ниже представлена реализация данной аналитической функции на языке программирования Python.

```
def analit_func(a, b, u):
        """analit_func
```

$$\frac{e^{-2\pi ib(u-1)}-e^{-2\pi ia(u-1)}}{-2\pi i(u-1)}+\frac{((e^{-2\pi ib(u+2.5)})-e^{(-2\pi ia(u+2.5))})}{(-2\pi i(u+2.5))}$$

```
        """
        return (np.exp(-2 * np.pi * 1j * b * (u - 1)) - np.exp(
            -2 * np.pi * 1j * a * (u - 1))) / (-2 * np.pi * 1j * (u - 1)) + (
            (np.exp(-2 * np.pi * 1j * b *
                    (u + 2.5)) - np.exp(-2 * np.pi * 1j * a * (u + 2.5))) /
            (-2 * np.pi * 1j * (u + 2.5)))
```

На следующих рисунках представлен результат работы аналитического решения и его сравнение с численным решением.

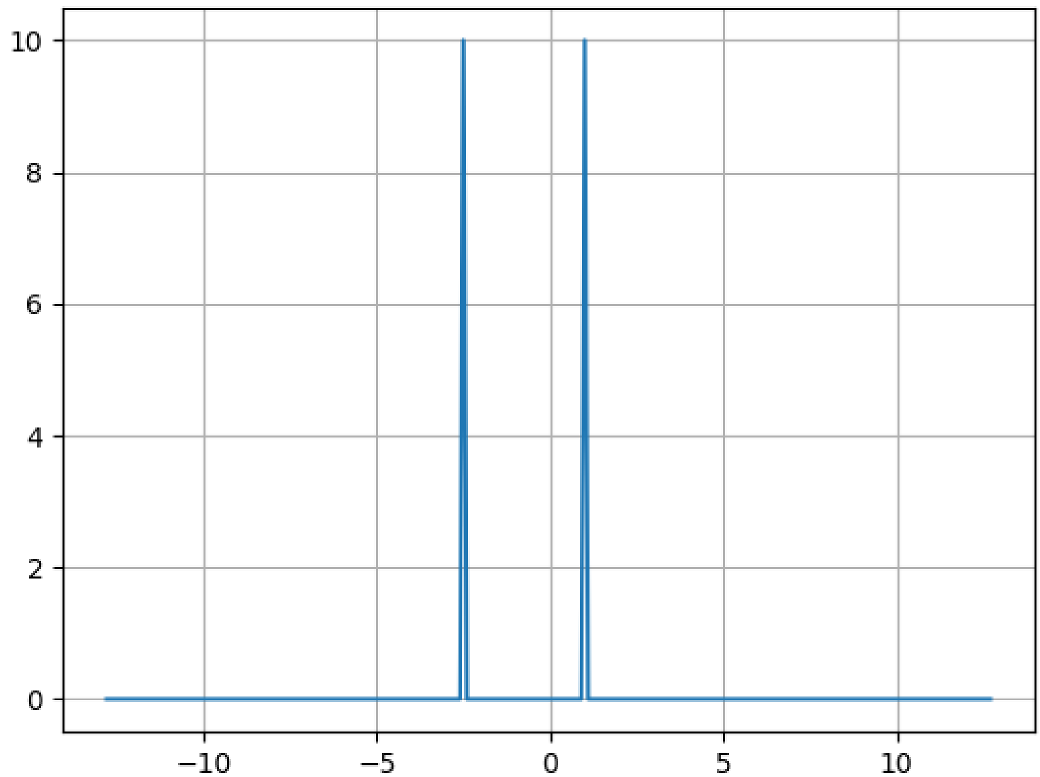

Рисунок 11 – Амплитуда преобразованной функции

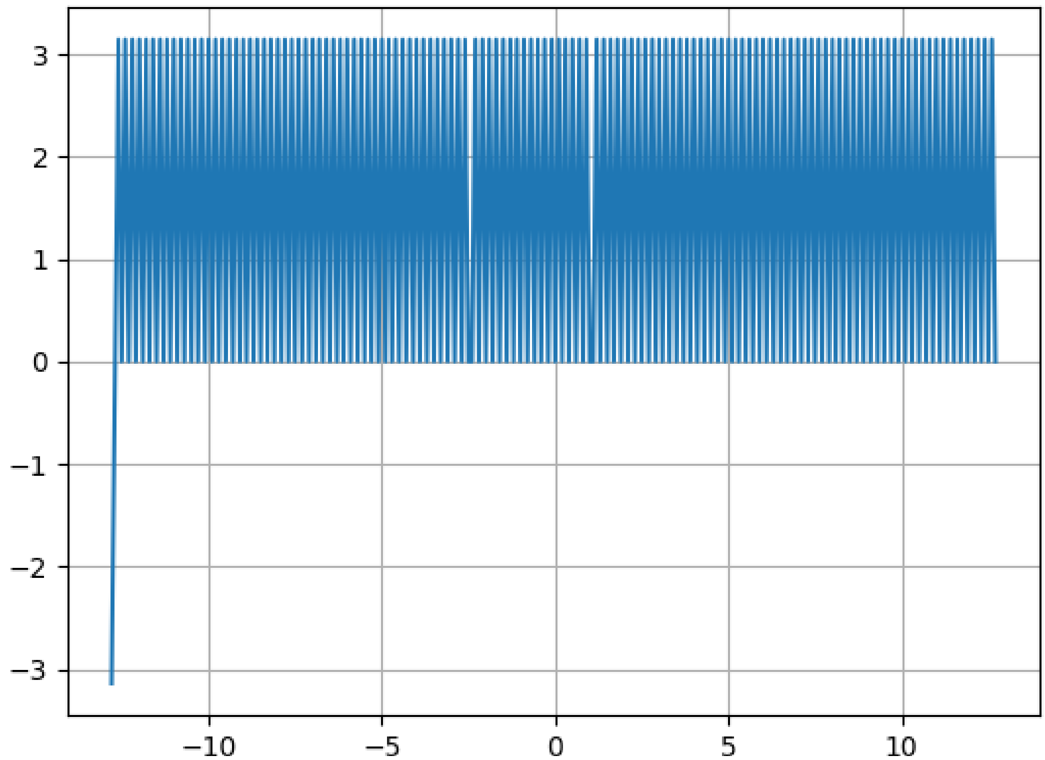

Рисунок 12 – Амплитуда преобразованной функции

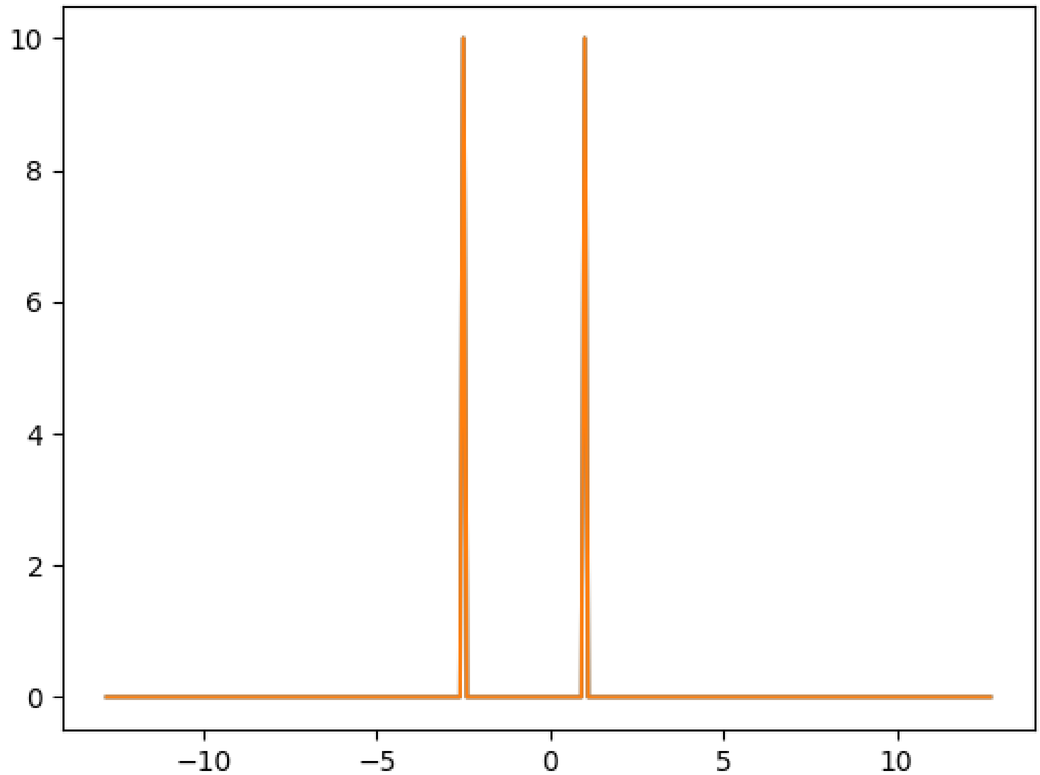

Рисунок 13 – Сравнение амплитуды

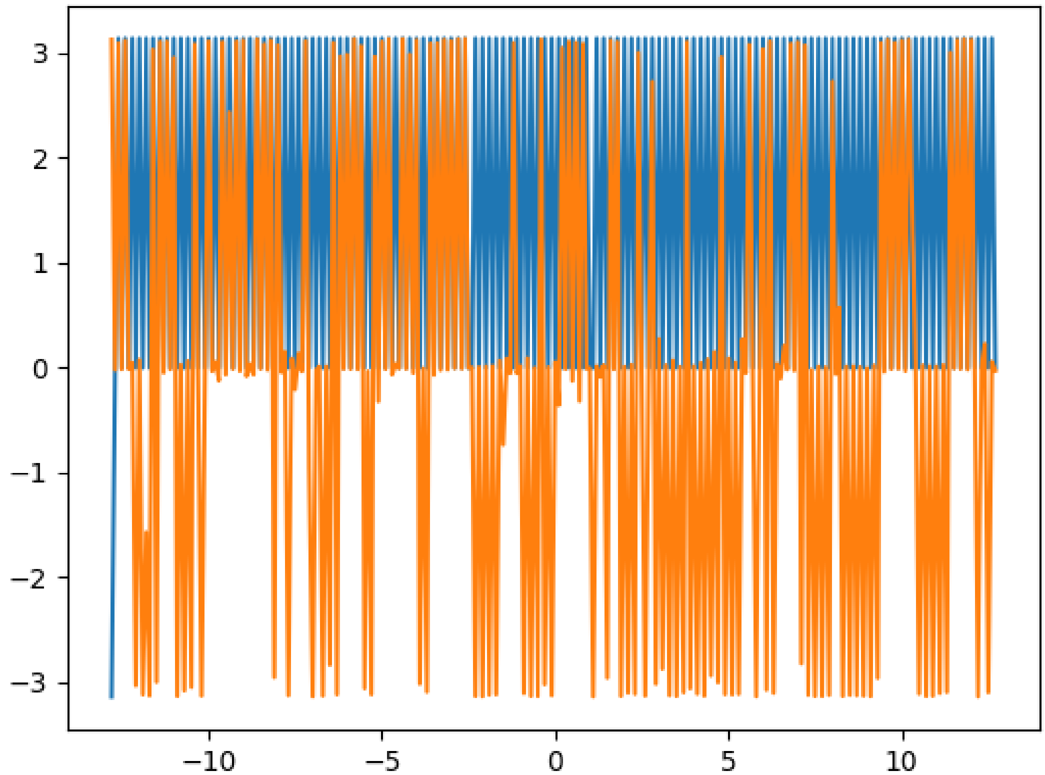

Рисунок 14 – Сравнение фазы

Ниже представлено сравнение графиков в нулевых точках.

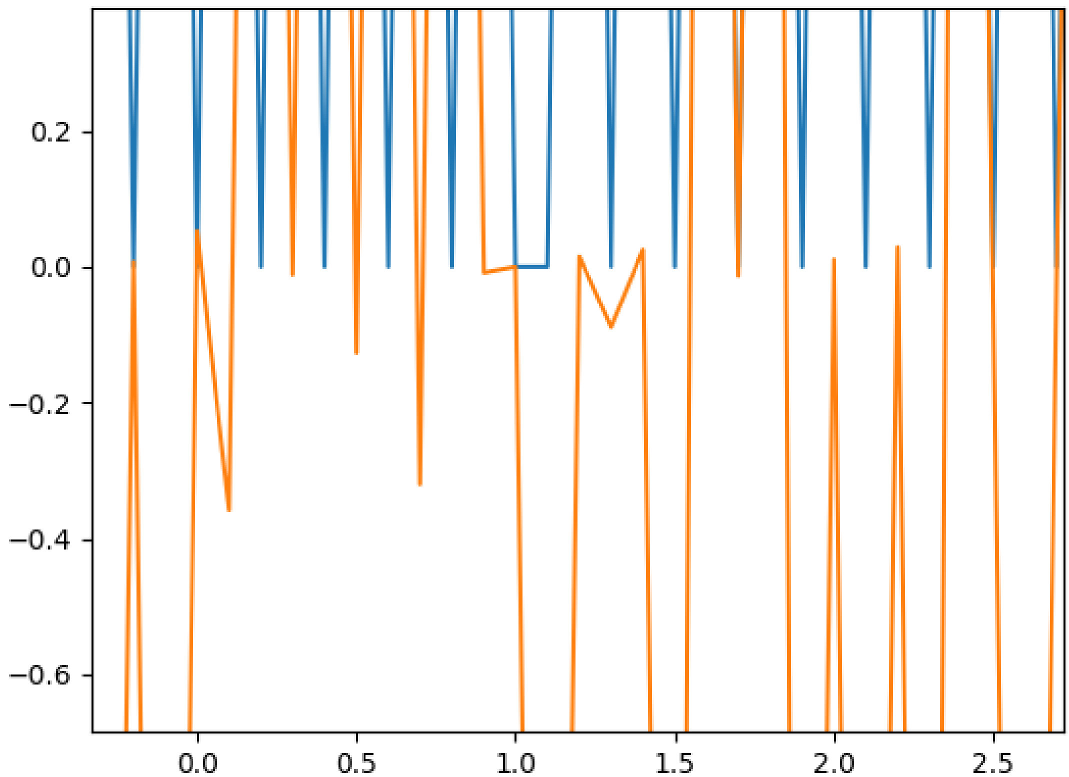

Рисунок 15 – Сравнение фазы в точке x=1

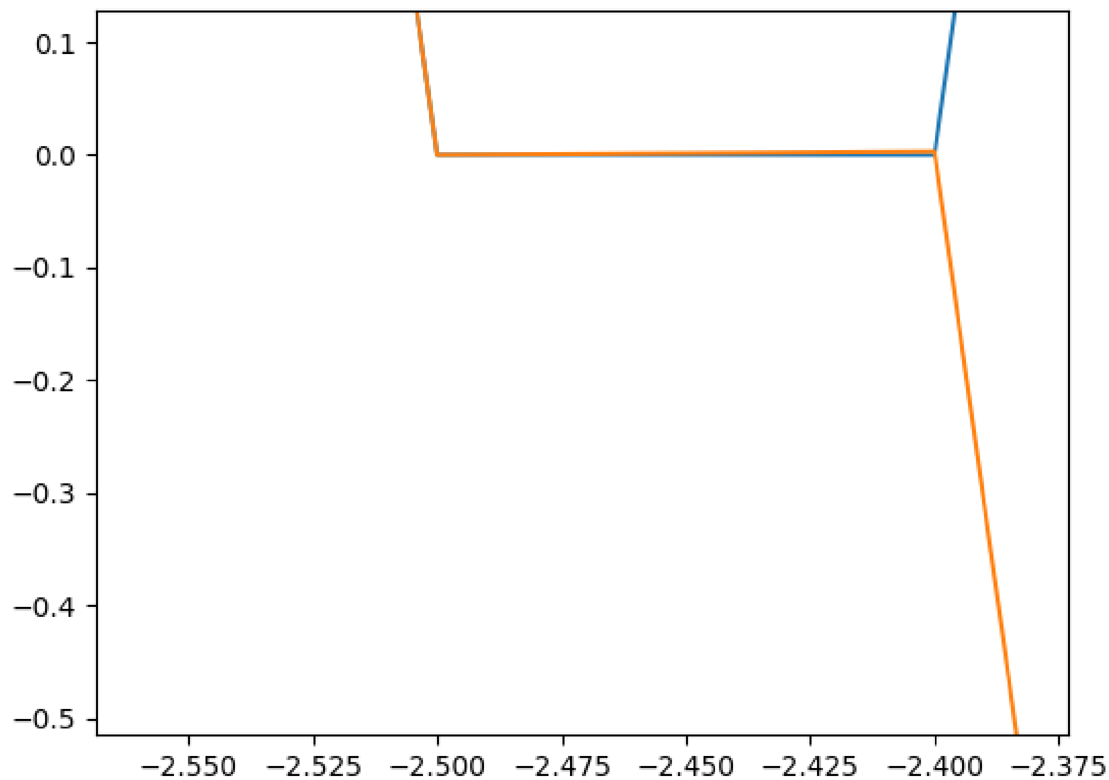

Рисунок 16 – Сравнение фазы в точке x=-$\frac{5}{2}$

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе было реализовано одномерное финитное преобразование Фурье с помощью метода БПФ, а так же с помощью метода численного интегрирования. Был рассчитан теоритический результат преобразования Фурье для функции $e^{2\pi ix} + e^{-5\pi ix}$. Построены графики для сравнения результатов различных преобразований. Были найдены результаты аналитического преобразования Фурье и преобразования Фурье через быстрое преобразование Фурье .

Приложение А

Код программы

```
from typing import List
import numpy as np
import matplotlib.pyplot as plt



def gauss_beam(x):
    return np.exp(-np.pi * (x**2))



def bokunofunction(x):
    """bokunofunction
    Function that calculates:
    f(x) = e^{2πix} + e^{-5πix}
    """
    return np.exp(2 * np.pi * 1j * x) + np.exp(-5 * np.pi * 1j * x)



def finit_fourier(f, step: float, xs: List[float]) -> List[float]:
    """finit_fourier
    Returns finite Fourier transform using fft.
    :param f: function to process.
    :param step: step.
    :type step: float
    :param xs: List of xs.
    :type xs: List[float]
    """
    fs = list(map(lambda x: f(x), xs))
    fs = np.fft.fftshift(fs)
    fourier = list(map(lambda x: x * step, np.fft.fft(fs)))
    return np.fft.fftshift(fourier)



def calculus_exp(u: float, x: float) -> float:
    """calculus_exp
```

```
    Returns e^{-2πiux} from fourier transform.
    :param u: u value
    :type u: float
    :param x: x value
    :type x: float
    :rtype: float
    """
    return np.exp(-2 * np.pi * 1j * u * x)


def analit_func(a, b, u):
    """analit_func
        (e^{-2πib(u-1)} - e^{-2πia(u-1)}) / (-2πi(u-1)) + ((e^{-2πib(u+2.5)}) - e^{(-2πia(u+2.5))}) / (-2πi(u+2.5))
    """
    return (np.exp(-2 * np.pi * 1j * b * (u - 1)) - np.exp(
        -2 * np.pi * 1j * a * (u - 1))) / (-2 * np.pi * 1j * (u - 1)) + (
            (np.exp(-2 * np.pi * 1j * b *
                    (u + 2.5)) - np.exp(-2 * np.pi * 1j * a * (u + 2.5))) /
            (-2 * np.pi * 1j * (u + 2.5)))


def calculus_fourier(f, step: float, xs: [float],
                     us: List[float]) -> List[float]:
    fourier = []
    for u in us:
        newx = 0
        for x in xs:
            newx += f(x) * calculus_exp(u, x)
        newx *= step
        fourier.append(newx)
    return fourier


def trapezium(f, x, h):
    return (f(x) + f(x + h)) / 2.0
```

$$\frac{e^{-2\pi ib(u-1)} - e^{-2\pi ia(u-1)}}{-2\pi i(u-1)} + \frac{((e^{-2\pi ib(u+2.5)}) - e^{(-2\pi ia(u+2.5))})}{(-2\pi i(u+2.5))}$$

```python
def simpson(f, x, h):
    return (f(x) + 4 * f(x + h / 2) + f(x + h)) / 6.0


def integrate(f, a, b, steps, meth):
    h = (b - a) / steps
    ival = h * sum(meth(f, a + i * h, h) for i in range(steps))
    return ival


def plot_calculus(f,
                  a: float,
                  b: float,
                  step_count: int,
                  fig: str = "Calculus"):
    step_src = abs(b - a) / step_count
    c = step_count / (2 * abs(b - a))
    step = (2 * abs(c)) / step_count
    us = np.arange(-c, c, step)
    xs = np.arange(a, b, step_src)
    fourier = calculus_fourier(f, step_src, xs, us)
    phase = np.angle(fourier)
    amplitude = np.abs(fourier)
    plt.figure(f"Phase {fig}")
    plt.grid()
    plt.plot(us, phase)
    plt.figure(f"Amplitude {fig}")
    plt.plot(us, amplitude)
    plt.grid()


def plot_fft(f, a: float, b: float, step_count: int, fig: str = "fft"):
    step_src = (b - a) / step_count
    c = step_count / (2 * (abs(a) + abs(b)))
    step = (2 * abs(c)) / step_count
    # step = (abs(a) + abs(b)) / step_count
    xs = np.arange(a, b, step_src)
    nxs = np.arange(-c, c, step)
```

```
    fourier = finit_fourier(f, step_src, xs)
    phase = np.angle(fourier)
    amplitude = np.abs(fourier)
    plt.figure(f"Phase {fig}")
    plt.plot(nxs, phase)
    plt.grid()
    plt.figure(f"Amplitude {fig}")
    plt.plot(nxs, amplitude)
    plt.grid()


def plot_function(f, a, b, step_count, fig):
    """plot_function
    Plot function on the interval from a to b
    with specified step count.
    :param f: f(x)
    :param a: start of the interval.
    :param b: interval end.
    :param step_count: how many points needs to be plotted.
    :param fig: plot name.
    """
    step = (b - a) / step_count
    xs = np.arange(a, b, step)
    ys = f(xs)
    plt.figure(f"Amplitude {fig}")
    plt.plot(xs, ys)
    plt.grid()


def plot_complex(f, a, b, step_count, fig="Source"):
    c = step_count / (2 * (abs(a) + abs(b)))
    step = (2 * abs(c)) / step_count
    nxs = np.arange(-c, c, step)
    ys = f(nxs)
    amplitude = np.abs(ys)
    phase = np.angle(ys)
    plt.figure(f"Phase {fig}")
    plt.plot(nxs, phase)
```

```
    plt.grid()
    plt.figure(f"Amplitude {fig}")
    plt.plot(nxs, amplitude)
    plt.grid()


if __name__ == "__main__":
    # plot_function(gauss_beam, -5, 5, 512, "frf")
    # plot_fft(gauss_beam, -5, 5, 512, "frf")
    # plot_calculus(gauss_beam, -5, 5, 512, "frf")
    # plot_complex(bokunofunction, -5, 5, 256, "bokunosource")
    plot_complex(lambda x: analit_func(-5, 5, x), -5, 5, 256, "test")
    plot_fft(bokunofunction, -5, 5, 256, "test")
    # plot_calculus(bokunofunction, -5, 5, 256, "bokunocalculus")
    plt.show()
```