Цель работы

Изучение методов Фурье-анализа дискретных и цифровых сигналов.

1 Ход работы

1.1 Написать программу вычисления прямого и обратного дискретного преобразования Фурье в матричной форме.

Дискретное преобразование Фурье является быстрым способом Фурье-анализа, которое можно применить к цифровым сигналам.

Прямое дискретное преобразование Фурье для функции вычисляется по формуле:

$$\dot{U}_k = \frac{1}{N}\sum_{n=0}^{N-1} u_n e^{-j\frac{2\pi n}{N}k} \quad (1)$$

При добавлении нормирующего коэффициента $\frac{1}{N}$ перед знаком суммы, как правило, применяется при вычислении обратного ДПФ вместо прямого.

Обратное дискретное преобразование Фурье вычисляется по формуле:

$$u_n = \sum_{n=0}^{N-1} \dot{U}_k e^{j\frac{2\pi n}{N}k} \quad (2)$$

Поскольку прямое и обратное дискретное преобразование Фурье можно интерпретировать в терминах операций над векторами, поэтому формулы (1) и (2) можно представлять в матричной форме:

$$\vec{U} \; = \; \vec{u}\, F^H \quad (3)$$

$$\vec{u} = \vec{U}F \quad (4)$$

В формулах F – матрица N×N, заполненная комплексными экспонентами $e^{j\frac{2\pi n}{N}k}$; F^H – эрмитово сопряженная с F матрицей.

Результат работы программы для функции $u(t) = \; sin(2\pi f t)$ при $f\; = 2$ Гц:

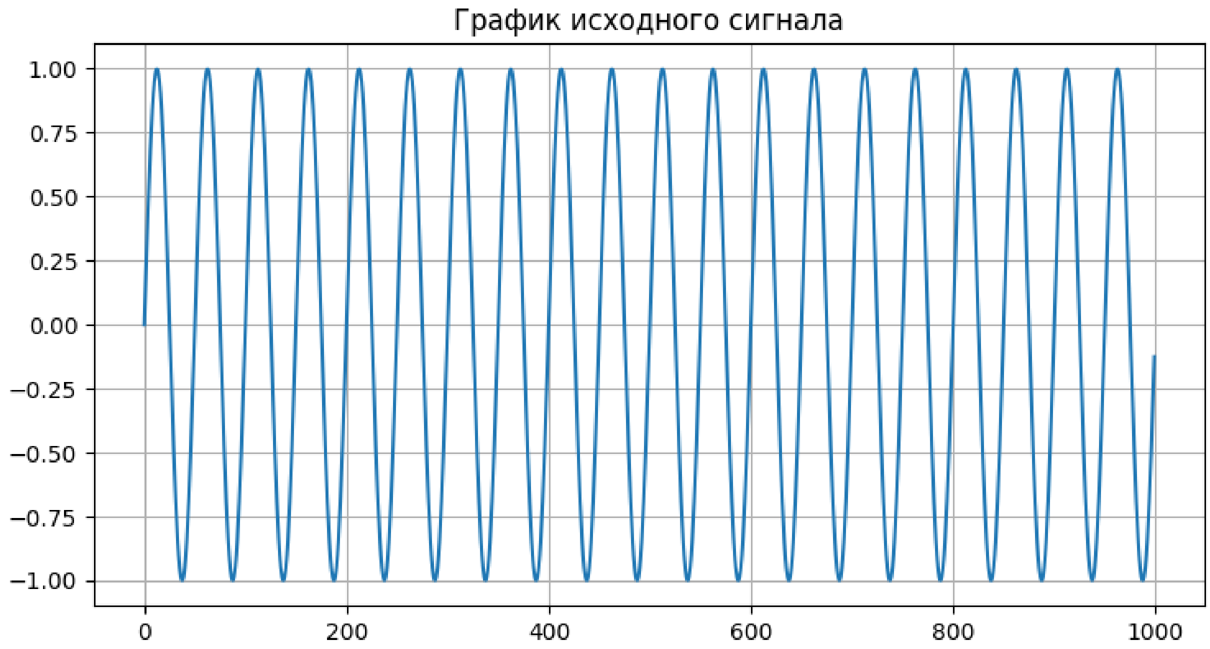

Рисунок 1 - График исходного сигнала

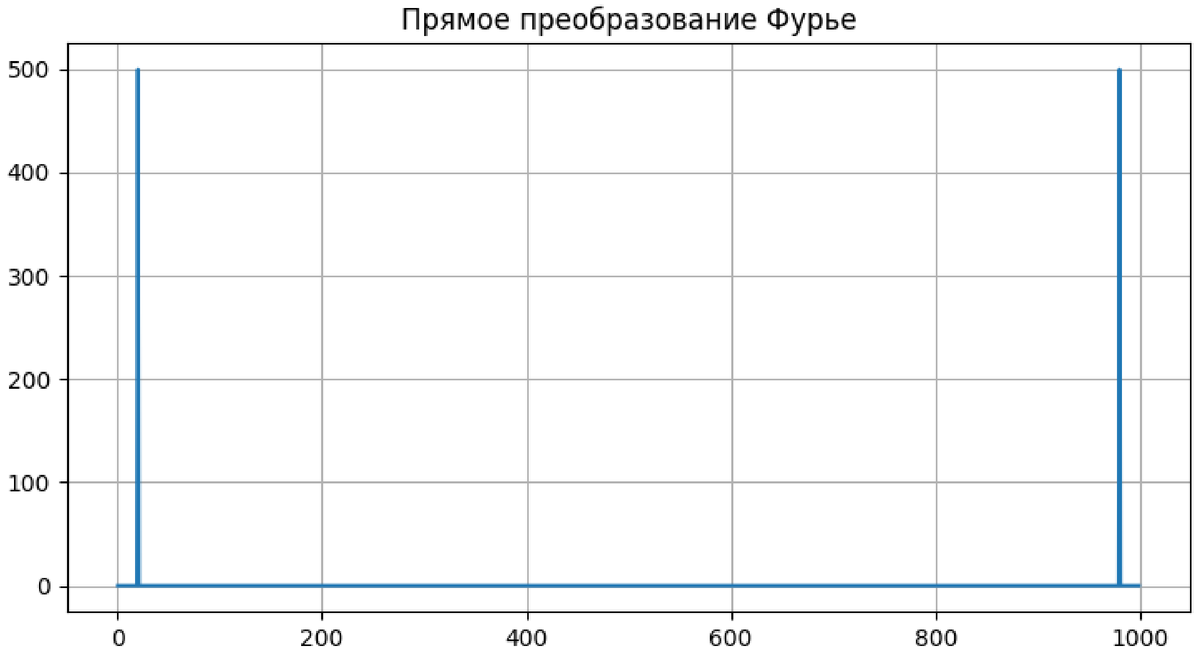

Рисунок 2 - Амплитудный спектр

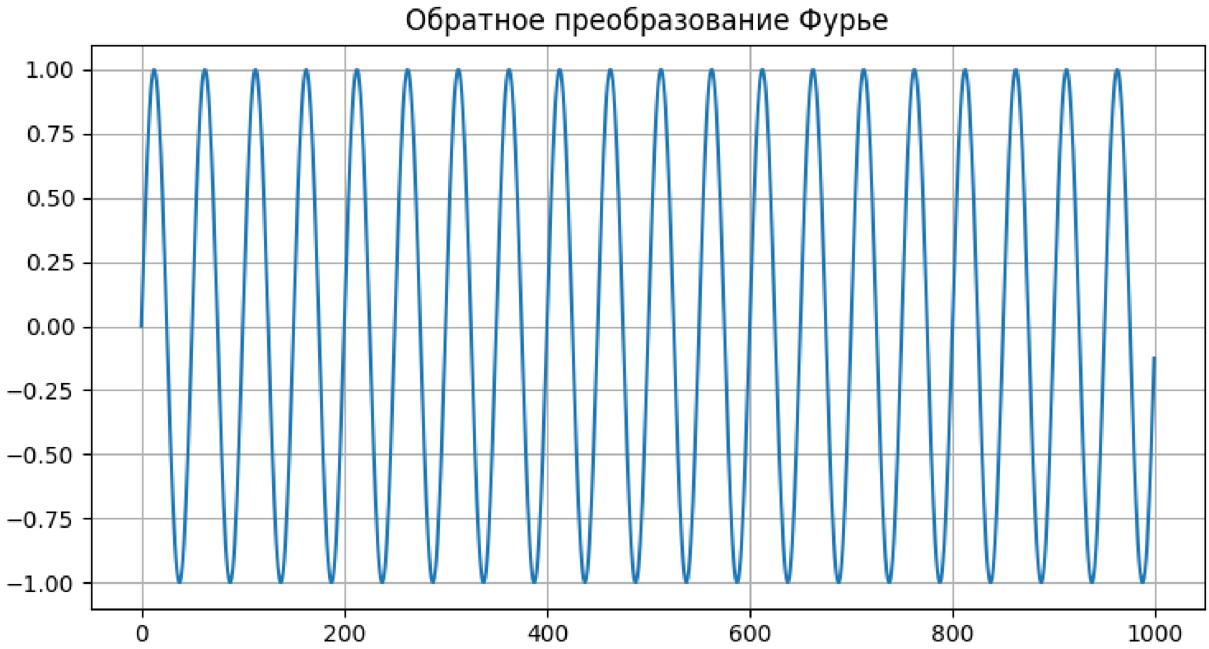

Рисунок 3 - График восстановленного сигнала

В данном пункте было изучено дискретное преобразование Фурье. Используя матричное представление формул (1) и (2), была написана программа для проведения прямого и обратного дискретного преобразования Фурье. Исходя из результатов, представленных на рисунке 1, можно сделать вывод, что программа работает корректно.

1.2 Продемонстрировать с помощью написанной программы свойства линейности, сдвига сигнала во времени и равенство Парсеваля.

С помощью программы, написанной в первом пункте, необходимо проверить свойства дискретного преобразования Фурье:

Линейность:

$$\alpha_1 u_k + \alpha_2 u_k \leftrightarrow \alpha_1 \dot{U}_n + \alpha_2 \dot{U}_n \quad (5)$$

Сдвиг сигнала во времени:

$$u(t-\tau) \leftrightarrow \dot{U}_n e^{-j\frac{2\pi n\tau}{N}} \quad (6)$$

Равенство Парсеваля:

$$\sum_{k=0}^{N-1} (u_k)^2 = \sum_{n=0}^{N-1} \left|\dot{U}_n\right|^2 \quad (7)$$

Для проверки свойства линейности были рассмотрены две функции: $u_1(t) = sin(2\pi f_1 t)$ и $u_2(t) = sin(2\pi f_2 t)$. Частоты функций: $f_1 = 2$ Гц и $f_2 = 3$ Гц, тогда коэффициенты $\alpha_1 = 3$ и $\alpha_2 = 6$.

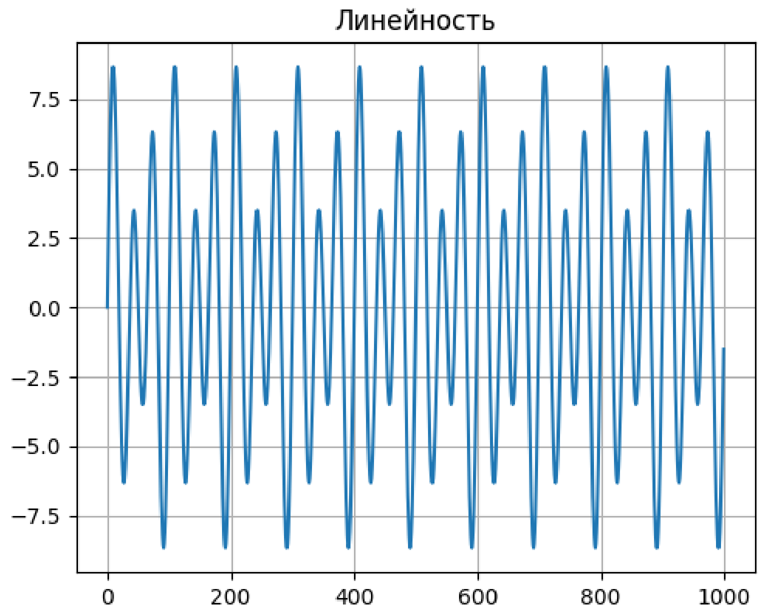

Рисунок 4 – Исходный сигнал

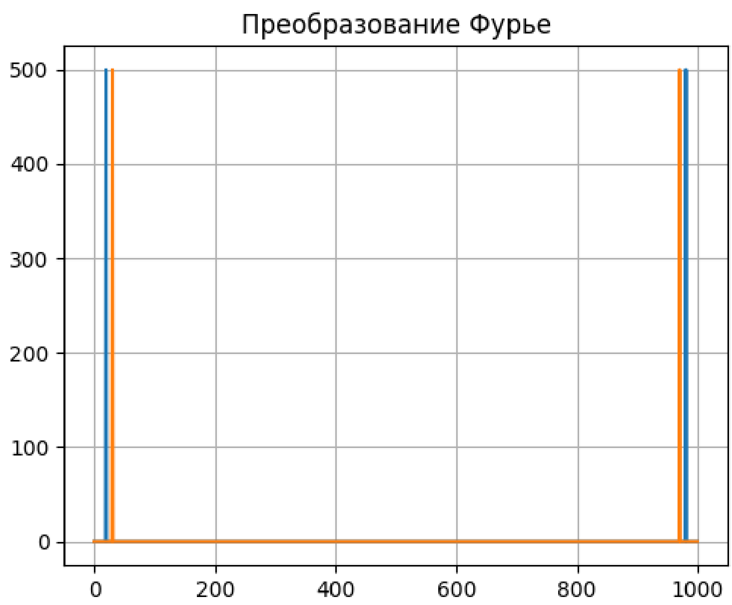

Рисунок 5 – Линейная комбинация спектров сигналов (для доказательства свойства линейности)

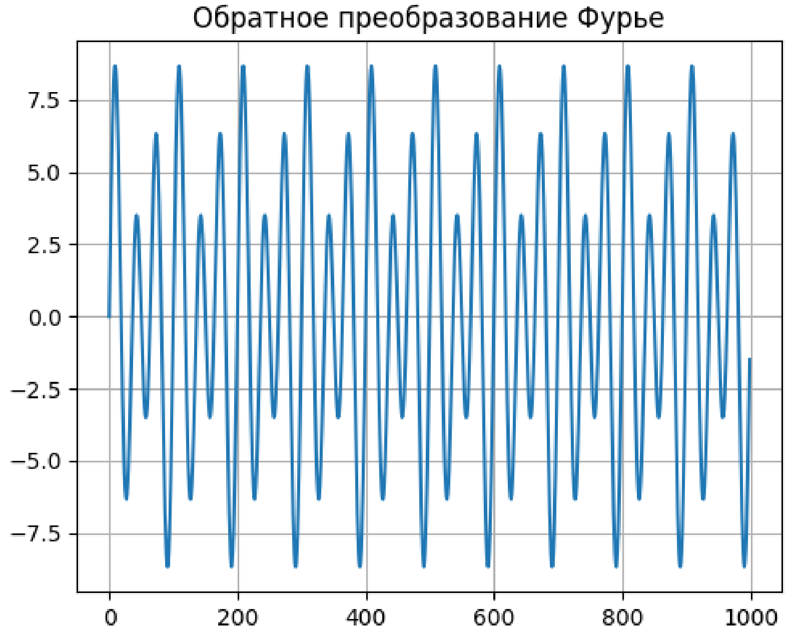

Рисунок 6 – Восстановленный сигнал

Доказательство свойства сдвига:

Для проверки свойства линейности были рассмотрена функция: $u(\mathrm{t}) = \sin(2\pi f \mathrm{t})$. Частоты функций: $f = 2$ Гц

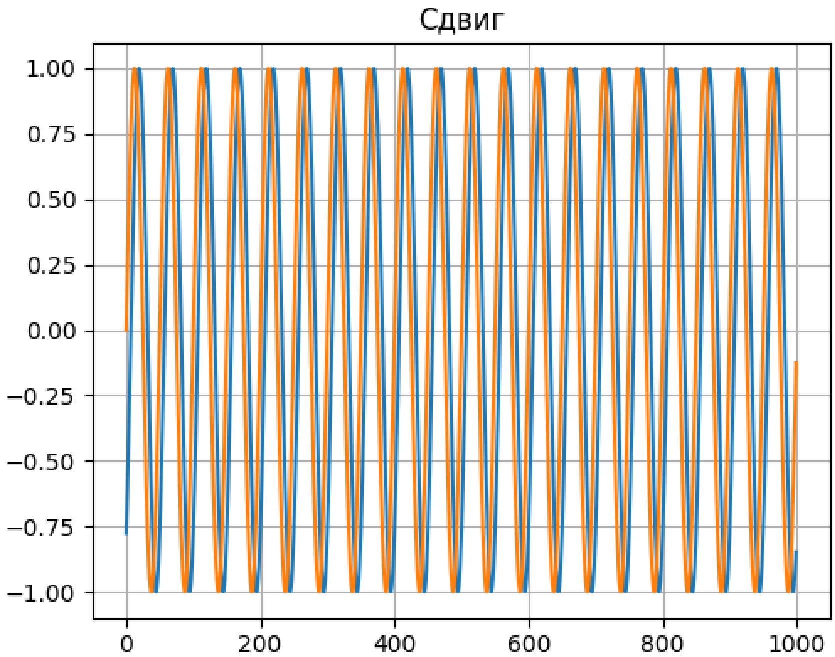

Рисунок 7 – Исходные сигналы

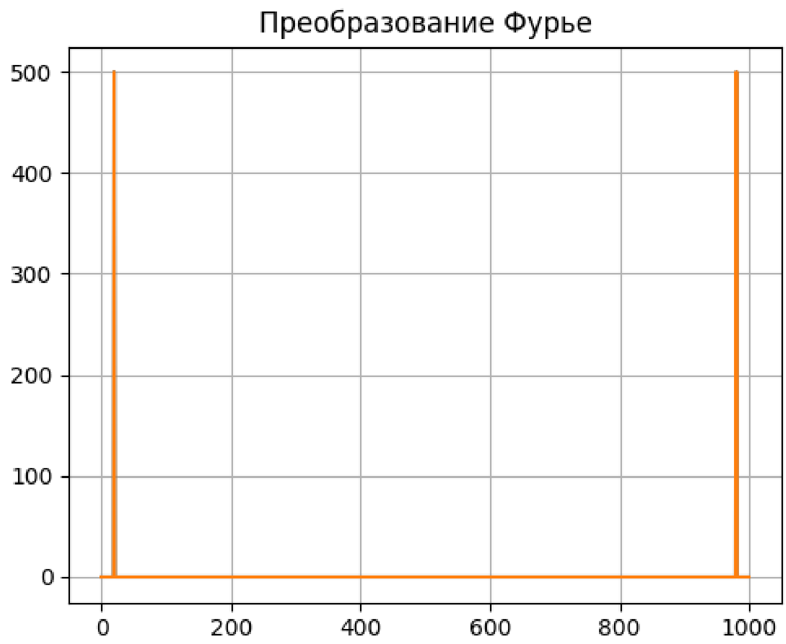

Рисунок 8 – Амплитудные спектры (для доказательства свойства сдвига сигнала во времени)

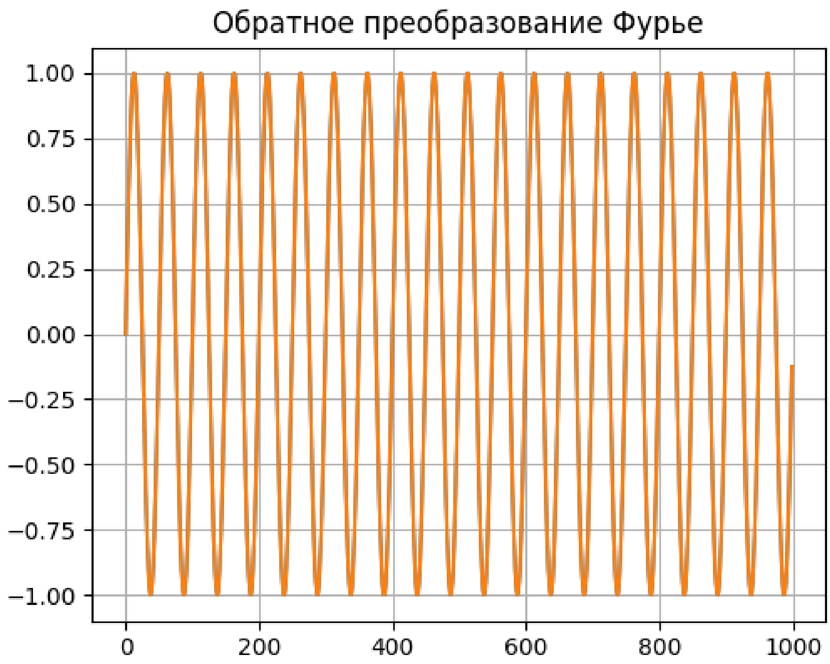

Рисунок 9 – Восстановленные сигналы

Результат работы программы для доказательства равенства Парсеваля: true

В данном пункте графически были доказаны линейность (рис.4-6), свойство сдвига сигнала во времени (рис. 7-8), равенство Парсеваля (рис. 9).

По формуле (6) спектр сдвинутого сигнала равен $\dot{U}_n e^{-j\frac{2\pi n\tau}{N}}$, где $\dot{U}_n$ – спектр исходного сигнала, однако $\left|e^{-j\frac{2\pi n\tau}{N}}\right| = 1$, поэтому, как видно по рисунку 8, амплитудные спектры не отличаются. Фазовый спектр сдвинутого сигнала будет отличаться от исходного, так как происходит умножение спектра исходного сигнала на комплексную экспоненту $e^{-j\frac{2\pi n\tau}{N}}$. На рисунке 8 можно увидеть, что для двух сигналов с разностью фаз τ амплитудные спектры совпадают.

1.3 Необходимо произвести декодирование аудио файла с записью тонального сигнала DTMF с помощью анализа амплитудного спектра его отсчетов.

DTMF – двухтональный многочастотный аналоговый сигнал, используемый для набора телефонного номера.

Исходный сигнал представляется набором отсчетов, где каждый отсчет является сложением двух синусоид с частотами, которые представлены в таблице 1.

Таблица 1 - Таблица частот для кодирования DTMF сигналов

(f_1/f_2)	1209 Гц	1336 Гц	1477 Гц	1633 Гц
697 Гц	1	2	3	A
770 Гц	4	5	6	B
852 Гц	7	8	9	C
941 Гц	*	0	#	D

Для каждого отсчета проводится дискретное преобразование Фурье. По полученному амплитудному спектру отсчета проводится поиск частот, используемых в кодировании сигнала. Поиск происходит по нахождению максимума в верхней и нижней группах частот. Максимальное значение в каждой из групп будут достигаться при искомых частотах. Таким образом, значение первой максимальной частоты будет соответствовать по таблице 1 значению f1, значение второй максимальной частоты значению f2. На пересечении этих частот по таблице 1 можно определить нажатую клавишу сигнала.

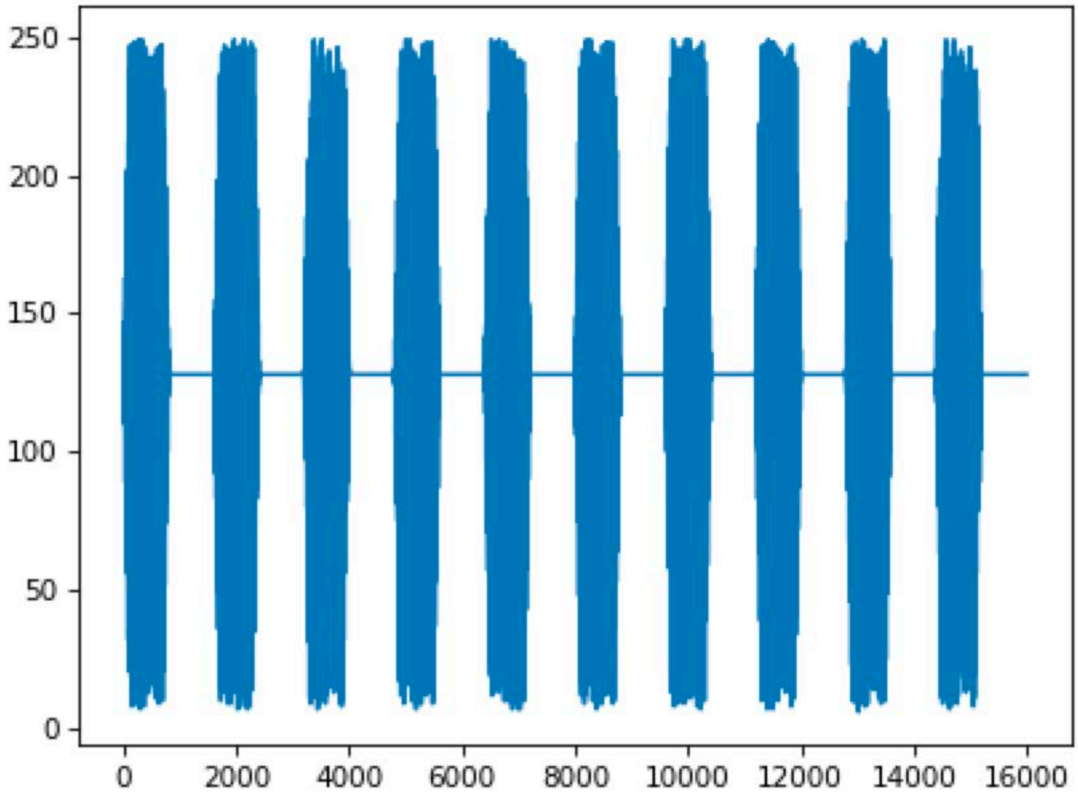

Рисунок 10 – Графическое представление исходного аудиосигнала

Рисунок 11 – Результат вычисления

В ходе выполнения данного пункта был произведен анализ аудио файла DTMF. Было построено графическое частотное представление аудиосигнала (рис.10). С помощью прямого преобразования Фурье были вычислены амплитудные спектры одиночных нажатий, в соответствии с которыми были восстановлены частоты кодирования, а затем и последовательность набранных цифр: *8-1-2-4-9-4-7-0-5*-2 (рис. 11*)*.

1.4 Выполнить оценку смещения между двумя изображениями путем анализа фазового спектра.

Рисунок 12 - Исходное изображение

Рисунок 13 - Смещенное изображение

В данном пункте необходимо оценить смещение между двумя изображениями в формате BMP24. Для оценки смещения применяется метод автокорреляции через спектр перекрестной мощности.

Для обоих изображений выполняется прямое преобразование Фурье по формуле (3). Далее необходимо найти спектр перекрестной мощности, вычисляемый по формуле (8):

$$R = \frac{U_1 \cdot U_2^*}{|U_1 \cdot U_2^*|} \tag{8}$$

U_1 и U_2 – матрицы спектральной плотности, U_2^* – комплексно-сопряженная к U_2 матрица.

Затем необходимо вычислить обратное преобразование Фурье для R. В результате получится матрица значений, среди которых необходимо найти максимальное. Индексы «пика» и будут являться смещением.

Так как изображение является двумерным сигналом, необходимо дважды сделать прямое преобразование Фурье. Затем найти спектр перекрестной мощности и также дважды сделать обратное преобразование Фурье. В полученной матрице необходимо найти аргументы максимума.

Рисунок 14 – Результат вычисления сдвига исходного изображения

Выводы

В ходе лабораторной работы были изучены методы Фурье-анализа дискретных и цифровых сигналов и выяснено, что дискретное преобразование Фурье является быстрым способом Фурье-анализа, которое можно применить к цифровым сигналам.

В пункте 1 данной лабораторной работы были изучены способы вычисления прямого и обратного преобразования Фурье в матричной форме. Используя матричное представление формул (1) и (2), была написана программа для проведения прямого и обратного дискретного преобразования Фурье. По полученным графикам, проверено, что программа работает корректно.

В ходе выполнения пункта 2 на основе написанной программы были доказаны свойства линейности, сдвига сигнала во времени и равенство Парсеваля, а также для демонстрации выполнения этих свойств были построены соответствующие графики. На рисунке 8 было продемонстрировано, что для двух сигналов с разностью фаз τ амплитудные спектры совпадают, а фазовые спектры различны.

В пункте 3 была проделана работа по проведению анализа аудио-файла DTMF - двухтональный многочастотный аналоговый сигнал, используемый для набора телефонного номера. Было построено графическое частотное представление аудиосигнала (рис. 10). С помощью прямого преобразования Фурье были вычислены амплитудные спектры одиночных нажатий, в соответствии с которыми были восстановлены частоты кодирования (рис. 11), а затем и последовательность набранных цифр: *8-1-2-4-9-4-7-0-5-2*.

В 4 пункте данной лабораторной работы путем анализа фазового спектра была произведена оценка смещения между двумя изображениями. Для оценки смещения был применен метод автокорреляции через спектр перекрестной мощности. Таким образом, исходное изображение сдвинуто по ширине на 66, по высоте на 137.

Приложение А

Листинг программы

```
# файл 2_1


import math
import matplotlib.pyplot as plt


def func(f, T, dt):
    l = []
    t = 0.0
    size = int(T / dt)
    for i in range(size):
        y = math.sin(2 * math.pi * f * t)
        l.append(y)
        t += dt
    return l


def ermit(F):
    res = []
    for i in range(len(F)):
        res.append([0] * len(F))
    for k in range(0, len(F)):
        for j in range(0, len(F[i])):
            res[k][j] = complex.conjugate(F[j][k])
    return res


def dft(signal):
    f = ermit(F(signal))
    result = [0] * len(f)
    for i in range(0, len(f)):
        for j in range(0, len(signal)):
            result[i] += signal[j] * f[j][i]
    return result


def F(signal):
    result = []
    for i in range(0, len(signal)):
        result.append([0] * len(signal))
    for i in range(0, len(signal)):
        for j in range(0, len(signal)):
            a = 2 * math.pi * i * j / len(signal)
            result[i][j] = complex(math.cos(a), math.sin(a))
    return result


def idft(signal):
    U = F(signal)
```

```
    res = signal
    result = [0] * len(U)
    l = len(U)
    for i in range(0, len(U)):
        for j in range(0, len(res)):
            result[i] += res[j] * U[j][i]
        result[i] = result[i] / complex(l)
    return result


def dsp1():
    f = 2
    T = 10
    dt = 0.01

    # Формирование сигнала
    signal = func(f, T, dt)

    # Прямое преобразование Фурье
    res1 = dft(signal)
    res4 = []
    for i in range(len(res1)):
        res4.append(abs(res1[i]))

    # Обратное преобразование Фурье
    res2 = idft(res1)

    fig, axes = plt.subplots(nrows=2, ncols=2)

    axes[0, 0].set(title='График исходного сигнала')
    axes[0, 1].set(title='Прямое преобразование Фурье')
    axes[1, 0].set(title='Обратное преобразование Фурье')

    fig.axes[0].grid()
    fig.axes[0].plot(signal)

    fig.axes[1].plot(res4)
    fig.axes[1].grid()

    fig.axes[2].plot(res2)
    fig.axes[2].grid()

    plt.show()


dsp1()
```

```
# файл 2_2


import matplotlib.pyplot as plt
import math


def func(f, T, dt):
    l = []
    t = 0.0
    size = int(T / dt)
    for i in range(size):
        y = math.sin(2 * math.pi * f * t)
        l.append(y)
        t += dt
    return l


def func_tau(f, T, dt, tau):
    l = []
    t = 0
    size = int(T / dt)
    for i in range(size):
        y = math.sin(2 * math.pi * f * (t - tau))
        l.append(y)
        t += dt
    return l


def ermit(F):
    res = []
    for i in range(len(F)):
        res.append([0] * len(F))
    for k in range(0, len(F)):
        for j in range(0, len(F[i])):
            res[k][j] = complex.conjugate(F[j][k])
    return res


def dft(signal):
    f = ermit(F(signal))
    result = [0] * len(f)
    for i in range(0, len(f)):
        for j in range(0, len(signal)):
            result[i] += signal[j] * f[j][i]
    return result


def F(signal):
    result = []
    for i in range(0, len(signal)):
        result.append([0] * len(signal))
    for i in range(0, len(signal)):
```

```
        for j in range(0, len(signal)):
            a = 2 * math.pi * i * j / len(signal)
            result[i][j] = complex(math.cos(a), math.sin(a))
    return result


def idft(signal):
    U = F(signal)
    res = signal
    result = [0] * len(U)
    l = len(U)
    for i in range(0, len(U)):
        for j in range(0, len(res)):
            result[i] += res[j] * U[j][i]
        result[i] = result[i] / complex(l)
    return result


def dsp2():
    f1 = 2
    f2 = 3
    T = 10
    dt = 0.01

    a = 3
    b = 6

    # Формирование сигнала
    s1 = func(f1, T, dt)
    s2 = func(f2, T, dt)

    s1_s2 = []
    for i in range(0, len(s1)):
        s1_s2.append(s1[i] * a + s2[i] * b)

    res = dft(s1)
    res1 = []
    for i in range(len(res)):
        res1.append(abs(res[i]))

    res2 = dft(s2)
    res4 = []
    for i in range(len(res)):
        res4.append(abs(res2[i]))

    result = []
    for i in range(0, len(res1)):
        result.append(res[i] * a + res2[i] * b)
    # Линейность
    r = idft(result)

    # Сдвиг
    f = 2
```

```
    T = 10
    dt = 0.01
    tau = math.pi / 2

    sig = func_tau(f, T, dt, tau)
    sig2 = func(f, T, dt)

    dft_signal = dft(sig)
    dft_signal2 = dft(sig2)

    for i in range(len(dft_signal2)):
        dft_signal[i] = dft_signal2[i] * math.e ** (1j * 2 * math.pi *
tau / len(dft_signal2))

    idft_signal = idft(dft_signal)
    idft_signal2 = idft(dft_signal2)

    for i in range(len(dft_signal2)):
        dft_signal[i] = abs(dft_signal[i])
        dft_signal2[i] = abs(dft_signal2[i])

    # равенство Парсеваля
    sum = 0
    sum2 = 0
    for i in range(len(s1)):
        sum += s1[i] * s1[i]
        sum2 += (abs(res1[i]) * abs(res1[i])) / len(res1)
    print(abs(sum - sum2) < 1e-6)

    fig, axes = plt.subplots(nrows=2, ncols=3)

    axes[0, 0].set(title='Линейность')
    axes[0, 1].set(title='Преобразование Фурье')
    axes[0, 2].set(title='Обратное преобразование Фурье')
    axes[1, 0].set(title='Сдвиг')
    axes[1, 1].set(title='Преобразование Фурье')
    axes[1, 2].set(title='Обратное преобразование Фурье')

    fig.axes[0].plot(s1_s2)
    fig.axes[0].grid()

    fig.axes[1].plot(res1)
    fig.axes[1].plot(res4)
    fig.axes[1].grid()

    fig.axes[2].plot(r)
    fig.axes[2].grid()

    fig.axes[3].plot(sig)
    fig.axes[3].plot(sig2)
    fig.axes[3].grid()

    fig.axes[4].plot(dft_signal)
```

```
    fig.axes[4].plot(dft_signal2)
    fig.axes[4].grid()

    fig.axes[5].plot(idft_signal)  # со сдвигом
    fig.axes[5].plot(idft_signal2)  # восстановленный
    fig.axes[5].grid()

    plt.show()


dsp2()


# файл 2_3

import math
from scipy.io import wavfile
import matplotlib.pyplot as plt


def find1(signal):
    maxi = -1
    index = 0
    for i in range(1, 100):
        if signal[i] > maxi:
            maxi = signal[i].real
            index = i

    index *= 10
    frq = [697, 770, 852, 941]

    delta = abs(frq[0] - index)
    result = frq[0]
    for i in range(4):
        if abs(frq[i] - index) < delta:
            delta = abs(frq[i] - index)
            result = frq[i]

    return result


def find2(signal):
    maxi = -1
    index = 0
    for i in range(101, 200):
        if signal[i] > maxi:
            maxi = signal[i].real
            index = i

    index *= 10
    frq = [1209, 1336, 1477, 1633]

    delta = abs(frq[0] - index)
```

```
    result = frq[0]
    for i in range(4):
        if abs(frq[i] - index) < delta:
            delta = abs(frq[i] - index)
            result = frq[i]

    return result


def which(f1, f2):
    w = [0] * 4
    for i in range(4):
        w[i] = [0] * 4

    if f1 == 697:
        f1 = 0
    if f1 == 770:
        f1 = 1
    if f1 == 852:
        f1 = 2
    if f1 == 941:
        f1 = 3

    if f2 == 1209:
        f2 = 0
    if f2 == 1336:
        f2 = 1
    if f2 == 1477:
        f2 = 2
    if f2 == 1633:
        f2 = 3

    w[0][0] = 1
    w[0][1] = 2
    w[0][2] = 3
    w[0][3] = "A"

    w[1][0] = 4
    w[1][1] = 5
    w[1][2] = 6
    w[1][3] = "B"

    w[2][0] = 7
    w[2][1] = 8
    w[2][2] = 9
    w[2][3] = "C"

    w[3][0] = "*"
    w[3][1] = 0
    w[3][2] = "#"
    w[3][3] = "D"

    return w[f1][f2]
```

```
def ermit(F):
    res = []
    for i in range(len(F)):
        res.append([0] * len(F))
    for k in range(0, len(F)):
        for j in range(0, len(F[i])):
            res[k][j] = complex.conjugate(F[j][k])
    return res


def dft(signal):
    f = ermit(F(signal))
    result = [0] * len(f)
    for i in range(0, len(f)):
        for j in range(0, len(signal)):
            result[i] += signal[j] * f[j][i]
    return result


def F(signal):
    result = []
    for i in range(0, len(signal)):
        result.append([0] * len(signal))
    for i in range(0, len(signal)):
        for j in range(0, len(signal)):
            a = 2 * math.pi * i * j / len(signal)
            result[i][j] = complex(math.cos(a), math.sin(a))
    return result


def dsp3():
    sample_rate, data = wavfile.read('11.wav')
    plt.plot(data)
    plt.show()
    for i in range(10):
        border = 1600 * i
        sample = []
        for j in range(800):
            sample.append(data[border + j])

        df = dft(sample)

        d = []
        for i in range(len(df)):
            d.append(abs(df[i]))

        print(which(find1(d), find2(d)))
        # plt.plot(d)
        # plt.show()
```

```
dsp3()


# файл 2_4


import math
import matplotlib.pyplot as plt


def multiplyElem(U1, U2):
    result = [0] * len(U1)
    for i in range(len(U1)):
        result[i] = [0] * 1
    for i in range(len(U1)):
        for j in range(1):
            result[i].pop(0)

    for i in range(len(U1)):
        for j in range(len(U1[0])):
            result[i].append(U1[i][j] * U2[i][j])
    return result


def ermConjMatrix(matrix):
    height = len(matrix[0])
    result = [0] * height
    for i in range(height):
        result[i] = [0] * 1
    for i in range(height):
        for j in range(1):
            result[i].pop(0)

    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            result[j].append(matrix[i][j].conjugate())
    return result


def dft2(image, ermit, ermit2):
    result = [0] * len(image)
    for i in range(len(image)):
        result[i] = [0] * len(image[0])

    for i in range(len(image)):
        for k in range(len(image[0])):
            for j in range(len(ermit[0])):
                tmp = image[i][k] * ermit[k][j]
                result[i][j] += tmp

    matrix = ermConjMatrix(result)
```

```
    U1 = [0] * len(result[0])
    for i in range(len(result[0])):
        U1[i] = [0] * 1

    for i in range(len(result[0])):
        for j in range(1):
            U1[i].pop(0)
    for i in range(len(matrix)):
        for j in range(len(ermit2[0])):
            template = complex(0, 0)
            for k in range(len(matrix[0])):
                template = template + ermit2[k][j] * matrix[i][k]
            U1[i].append(template)
    return U1


def divElem(U):
    result = [0] * len(U)
    for i in range(len(U)):
        result[i] = [0] * 1
    for i in range(len(U)):
        for j in range(1):
            result[i].pop(0)

    for i in range(len(U)):
        for j in range(len(U[0])):
            tmp = abs(U[i][j])
            Complex = U[i][j]
            result[i].append(Complex / tmp)
    return result


def ermMatrix(U):
    result = [0] * len(U[0])
    for i in range(len(U[0])):
        result[i] = [0] * 1
    for i in range(len(U[0])):
        for j in range(1):
            result[i].pop(0)
    for i in range(len(U)):
        for j in range(len(U[0])):
            result[j].append(U[i][j])
    return result


def MultiplyForIfft(G, value):
    matrix = [0] * len(G)
    for i in range(len(G)):
        matrix[i] = [0] * 1
    for i in range(len(G)):
        for j in range(1):
            matrix[i].pop(0)
```

```
    for i in range(len(G)):
        for j in range(len(G[0])):
            matrix[i].append(G[i][j] * value)
    return matrix


def MultiplyForIfft2(matrix, F):
    result = [0] * len(matrix)
    for i in range(len(matrix)):
        result[i] = [0] * 1
    for i in range(len(matrix)):
        for j in range(1):
            result[i].pop(0)

    for i in range(len(matrix)):
        for j in range(len(F[0])):
            tmp = complex(0, 0)
            for k in range(len(matrix[0])):
                tmp = tmp + F[k][j] * matrix[i][k]
            result[i].append(tmp)
    return result


def idft2(G, height, width, F1, F2):
    value = 1 / width
    matrix = MultiplyForIfft(G, value)
    r = MultiplyForIfft2(matrix, F1)
    g = ermMatrix(r)
    value = 1 / height
    matrix = MultiplyForIfft(g, value)
    result = MultiplyForIfft2(matrix, F2)
    result = ermMatrix(result)
    return result


def findMaxIndex(U):
    x = 0
    y = 0
    maximum = 0.0
    result = [0] * 2
    for i in range(len(U)):
        for j in range(len(U[0])):
            if abs(U[i][j]) > maximum:
                maximum = U[i][j]
                x = i
                y = j
    result[0] = x
    result[1] = y
    return result


def mass(x):
    # создаём массивы и удаляем нули
```

```
    F1 = []
    FH1 = []
    for i in range(x):
        F1.append([0] * 1)
        FH1.append([0] * 1)
    for i in range(x):
        for j in range(1):
            FH1[i].pop(0)
            F1[i].pop(0)

    for i in range(x):
        for j in range(x):
            complex_res = complex(math.cos(2 * math.pi * i * j / x),
math.sin(2 * math.pi * i * j / x))
            F1[i].append(complex_res)
            FH1[j].append(complex_res.conjugate())
    return F1, FH1


def dsp4(filename1, filename2):
    image1 = plt.imread(filename1)
    image2 = plt.imread(filename2)
    r1 = image1[:, :, 0]
    r2 = image2[:, :, 0]

    height = len(r1)
    width = len(r1[0])

    F1, FH1 = mass(width)

    F2, FH2 = mass(height)

    U1 = dft2(r1, FH1, FH2)
    U2 = dft2(r2, FH1, FH2)

    U = multiplyElem(ermMatrix(U1), ermConjMatrix(U2))
    U = divElem(U)
    U = idft2(U, height, width, F1, F2)

    plt.plot(U)
    plt.show()
    result = findMaxIndex(U)
    x = width - result[1]
    y = result[0]

    print(x)
    print(y)


dsp4("truck_first.bmp", "truck_second.bmp")
```