Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **«Робототехника и комплексная автоматизация»**

КАФЕДРА **«Автоматизация технологических процессов и производств»**

Отчет по учебной практике

Студент Стрельников Иван Александрович
(фамилия, имя, отчество)

Тип практики учебная практика

Студент _____ И.А. Стрельников
Подпись, дата *и.о. фамилия*

2024 *г.*

Оглавление

1. Постановка задания....3
2. Выполнение задания....4
3. Вывод....9

1. Постановка задания

Я получил задание изучить алгоритм быстрого преобразования Фурье (БПФ), а также спроектировать анализ данных, полученных в результате измерения, используя программное обеспечение ПК.

Дискретное преобразование Фурье (в англоязычной литературе DFT, Discrete Fourier Transform) — это одно из преобразований Фурье, широко применяемых в алгоритмах цифровой обработки сигналов (его модификации применяются в сжатии звука в MP3, сжатии изображений в JPEG и др.), а также в других областях, связанных с анализом частот в дискретном (к примеру, оцифрованном аналоговом) сигнале.

Дискретные преобразования Фурье помогают решать дифференциальные уравнения в частных производных и выполнять такие операции, как свёртки. Дискретные преобразования Фурье также активно используются в статистике, при анализе временных рядов. Существуют многомерные дискретные преобразования Фурье.

Вспомним выражение для ДПФ:

$$X[m] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nm/N}$$

Из него видно, что для расчёта N-точечного ДПФ требуется N^2 вычислений с комплексными числами. Получается ресурсозатратно, особенно если мы имеем дело с большим количеством отсчётов. Учёным не нравилось делать много вычислений, и несколько человек (существуют разные мнения, кто был первым) предложили алгоритмы, позволяющие вычислять ДПФ, производя меньшее количество математических операций. Называются такие алгоритмы Быстрое преобразование Фурье (БПФ).

Для N-точечного ДПФ требуется N^2 операций комплексного умножения и сложения. А что, если разделить исходный сигнал на два длиной N/2 отсчётов каждый? Тогда получается, что для вычисления ДПФ от одной из половин потребуется $(N/2)^2 = N^2/4$ математических операций. Или $N^2/2$ для двух половин (если не считать процедуру разделения и объединения). Получается, эффективность вычислений возрастает в два раза! Но это ещё не предел. Мы можем делить наш сигнал до тех пор, пока не получим набор из сигналов, состоящих из двух отсчётов.

Мы рассмотрели алгоритмы БПФ по основанию два с прореживанием по времени и с прореживанием по частоте. Данные алгоритм очень эффективны, но они имеют существенное ограничение: длина входных данных и выходного вектора должна быть целой степенью двойки $N=2^t$.

В данном случае мы рассмотрим алгоритмы БПФ для произвольной составной длины N=ML, где M и L — произвольные положительные целые.

Данный подход позволит построить эффективные процедуры дискретного преобразования Фурье практически для всех длин, а не только для $N=2^t$.

2. Выполнение задания

Прибор измеряет напряжение на протяжении определенного времени. Информация поступает на ПК и записывается в таблицу формата csv. Данные поступают с двух каналов.

0.000000	-0.506366
0.006283	-0.500938
0.012566	-0.495490
0.018848	-0.490022
0.025130	-0.484536
0.031411	-0.479030
0.037690	-0.473505
0.043968	-0.467961
0.050244	-0.462399
0.056519	-0.456819

Пример входных данных

При анализе полученных данных удобнее всего воспользоваться программой, написанной на одном из языков программирования. Но возникают некоторые нюансы с обработкой данных. Программу я решил написать на языке программирования Python, так как считаю данный язык достаточно простым для понимания.

Хоть в Python и есть инструменты для решения подобных задач, я решил спроектировать алгоритм самостоятельно для большего понимания.

Импортирую нужные библиотеки:

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

Вспомогательная функция для вычисления поворотного коэффициента:

```
def W(n,k,N):#поворотный коэффициент
    return np.around(np.exp(-2j*np.pi*n*k/N),9)
```

Когда я буду выполнять алгоритм БПФ, возникнет такая ситуация, что разделить данные уже будет невозможно (их количество равно простому числу). В этом случае я планирую использовать алгоритм ДПФ:

```
def X_k(x,k):#X[k]
    N = len(x)
    return sum(x[n]*W(n,k,N) for n in range(N))

def DFT(x):#преобразование Фурье
    N = len(x)
    X = np.array([])
    for k in range(N):
```

```
        X = np.append(X,X_k(x,k))
    return X
```

Вспомогательные функции, используемые в БПФ для разделения N на M и L:

```
def dividers(N):  # находим наименьший делитель
    for i in range(2, int(N ** 0.5) + 1):
        if N % i == 0:
            return i
    return False


def div_index(a, b):  # разделить индексы
    m_s = range(a)
    p_s = range(b)
    return [[p * a + m for p in p_s] for m in m_s]
```

В алгоритме я использую рекурсивный метод. Хотя он не является оптимальным, он наиболее прост для понимания, что является положительным фактором в учебном проекте. Сам алгоритм БПФ выглядит следующим образом:

```
def FFT(x):  # быстрое преобразование Фурье
    N = len(x)
    M = dividers(N)
    if not M:
        return DFT(x)

    L = N // M
    n_lists = div_index(M, L)  # делим индексы
    y_s_div = [FFT(x[n_list]) for n_list in n_lists]  # разделяем иксы по
индексам и тут же их преобразовываем по Фурье
    z_s_div = [np.array([W(m, q, N) * y_s_div[m][q] for m in range(M)]) for q
in
               range(L)]  # составляем L сигналов zq[m]
    v_s_div = np.array([FFT(z) for z in z_s_div])  # преобразовываем по Фурье
L сигналов zq[m]
    X = np.around(v_s_div.T.flatten(), 6)  # пишем в нужном виде
    return X
```

По оси ординат с помощью БПФ мы находим энергию. По оси абсцисс должна идти частота, которую мы находим преобразованием времени:

```
def FFTF(N,sample_spacing):#функция для нахождения частот
    f = np.arange(0,N,1/(N*sample_spacing))
    return np.hstack([f[:int(np.ceil(N/2))],f[:int(np.ceil(N/2+10e-
9))][1:][::-1]*(-1)])
```

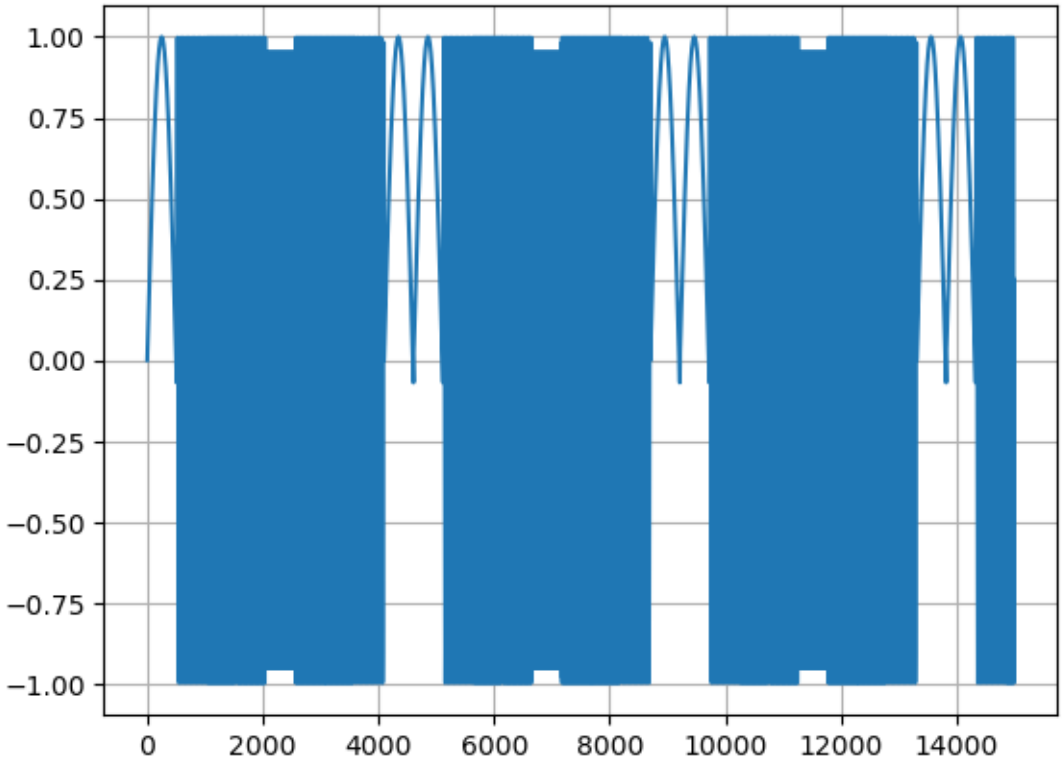
На вход нам приходит csv файл, нам нужно его выгрузить и разделить на 2 канала:

```
data = np.genfromtxt('1.csv', delimiter=';', dtype=None).T
x_1,x_2 = data[0],data[1]
t = np.arange(x_1.size)
```

Мы должны определиться с размером выборки данных, я выбрал число, наиболее близкое к размеру данных, но подходящее для разложения на множители.

```
N = 2**14*3**3 #близкое значение к общему размеру, но легко раскладываемое
```

Построим график зависимости напряжения от времени. Время выберем от 0 до 15000 для более удобного восприятия:



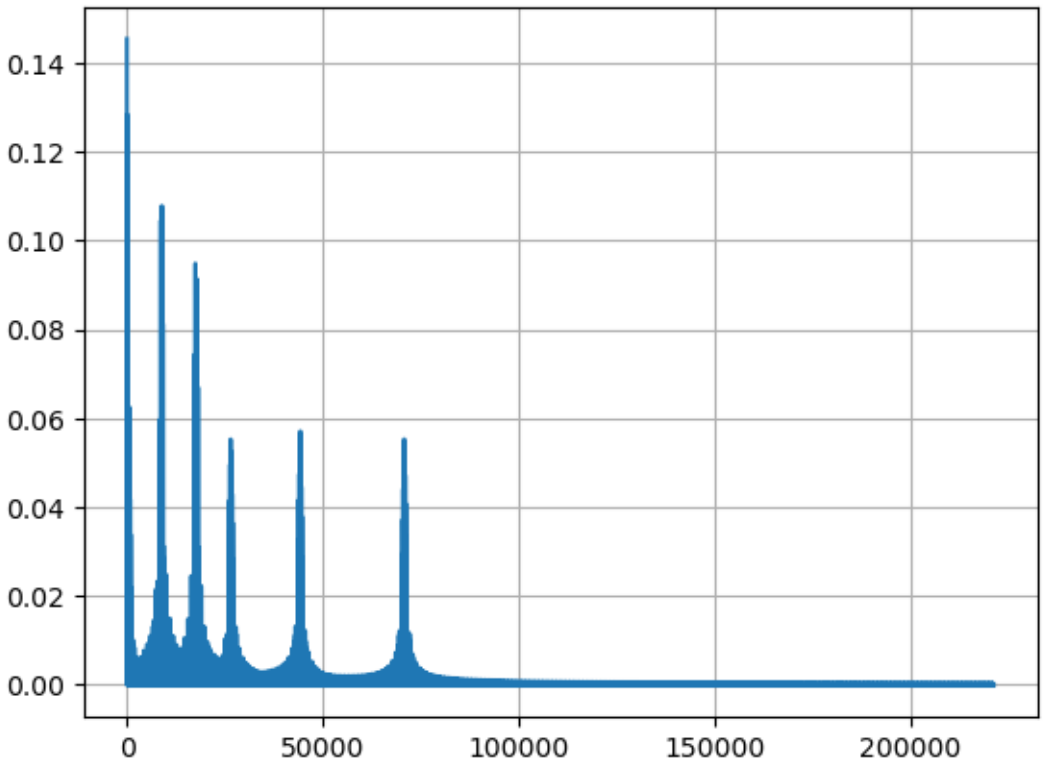
Самое время построить АЧХ

```
t1 = time.perf_counter()
X = FFT(x_1[:N])
t2 = time.perf_counter()
print('Прошло времени:',round(t2-t1,5),'c')
X_abs = np.abs(X)/N
fx = FFTF(N, 1/N)
```

Я решил узнать, сколько времени займет обработка данных. Код вернул следующий результат:

```
Прошло времени: 959.11367 c
```

Теперь можем построить сам график. По абсциссе fx, по ординате X_abs:



На данном изображении мы видим, что энергия имеет свои пиковые значения.

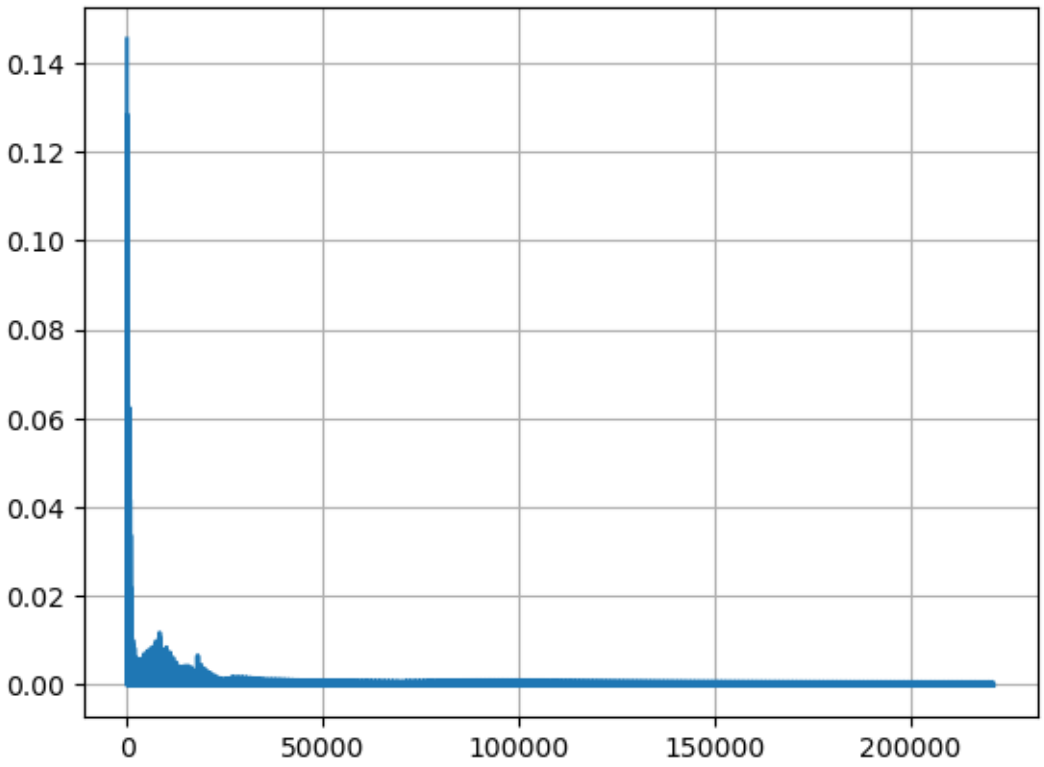
Для удаления лишних частот я решил применить алгоритм режекторного фильтра. Режекторный фильтр, или полосно-заграждающий фильтр, (профессиональный жаргон — фильтр-пробка) — электронный или любой другой фильтр, не пропускающий колебания некоторой определённой полосы частот, и пропускающий колебания с частотами, выходящими за пределы этой полосы.

```
def notch_filter(fs, f, BW, X):#Режекторный фильтр fs-частота дискретизации, f-резонансная, BW - ширина пропускания
    f = f/fs
    BW = BW/fs
    R = 1-3*BW
    K = (1-2*R*np.cos(2*np.pi*f)+R**2)/(2-2*np.cos(2*np.pi*f))
    a_0,a_1,a_2 = K,-2*K*np.cos(2*np.pi*f),K
    b_1,b_2 = 2*R*np.cos(2*np.pi*f),-(R**2)
    Y=np.copy(X)
    for i in range(2,X.size):
        Y[i] = a_0*X[i] + a_1*X[i-1] + a_2*X[i-2] + b_1*Y[i-1] + b_2*Y[i-2]
    return Y
```

Применим фильтрацию к исходным данным

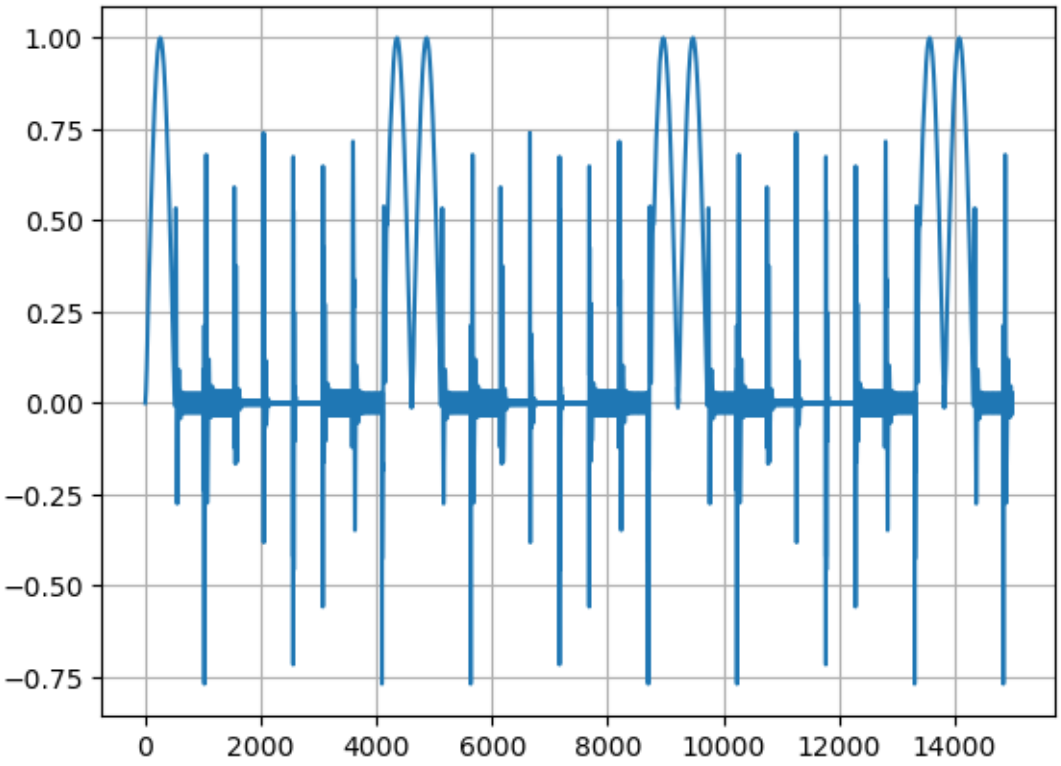
```
y=np.copy(x_1[:N])
for i in range(1000,100000):#тут я ищу пики, от 1000 чтобы синусоиду оставить, до 10*5 чтобы меньше операций было
    if X_abs[i]>0.3*X_abs[0] and (X_abs[i]==max(X_abs[i-500:i+500])):
        y = notch_filter(N,i,i//1.6,y)# ширину в i//1.6 выбрал подбором
```

Теперь мы можем применить алгоритм БПФ к отфильтрованным данным.



Мы видим, что все пики энергии были отфильтрованы, и у нас осталась только нужная частота

Сам график отфильтрованных данных выглядит так (с выборкой времени от 0 до 15000)

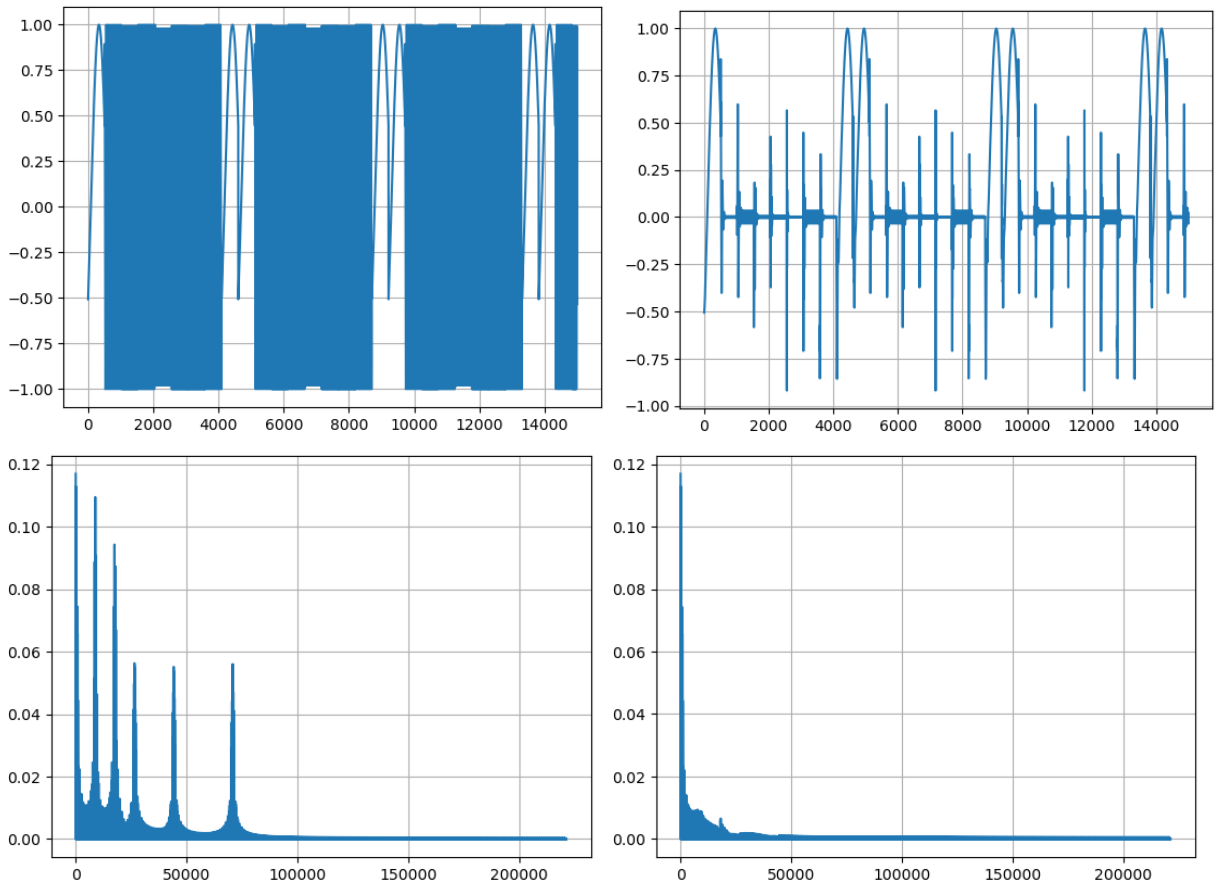


Большую часть лишних данных нам удалось убрать. Это показывает, что с помощью алгоритма БПФ можно исследовать частотные характеристики сигналов, выделять особенности, отделять доминирующие частоты,

фильтровать шумы и проводить компрессию файлов.

Результаты для второго канала до и после фильтрации выглядят следующим образом:

До фильтрации: После фильтрации:



3. Вывод

Данная практика позволила мне изучить основы цифровой обработки сигнала и применить её алгоритм для анализа реальных данных. Я ознакомился с работой техника-программиста, поверхностно изучил рабочий процесс производства. Также, я оценил рентабельность этой профессии на российском рынке и квалифицированность большей части сотрудников.