Санкт-Петербургский государственный политехнический
университет Петра Великого.
**Высшая школа интеллектуальных сисем и
суперкомпьютерных технологий**

Лабораторная работа

Дискретное преобразование Фурье.

Работу выполнила студентка:
_____ А. И.Луцкевич
«____» _____ 2021 г.

Преподаватель лабораторных
работ:
_____ Н. В.Богач
«___» _____ 2021 г.

Санкт-Петербург, 2021 г.

Суть работы 7.1:

Прочитать пояснения и запустить примеры.

Был прочитан весь параграф, были запущены все примеры и просмотрены все результаты.

Суть работы 7.2:

Во втором пункте седьмой лабораторной работы было продемонстрировано использование ДПФ (дискретное преобразование Фурье) и обратное ДПФ в виде произведения матриц. Такие операции занимают время N^2, где N - длина массива, что достаточно быстро для большинства применений, но есть более быстрый алгоритм: Быстрое Преобразование Фурье (БПФ), занимающий $Nlog(N)$.
Ключевая вещь в БПФ это лемма Даниелсона-Ланкзоса, которая предлагает рекурсивный алгоритм для DFT:
1. Входящий массив y разделяется на четное число элементов e, и на нечётные элементы o.
2. Вычислить DFT e и o с помощью рекурсивных запросов.
3. Вычислить DFT(y) для каждого значения n используя лемму Даниелсона-Ланкзоса.
В случае если длина исходного массива равна 1, DFT(y) = y. Или если длина y очень мала, можно вычислить её DFT с помощью матричного умножения, используя предварительно вычисленную матрицу.

Возьмем небольшой сигнал и вычислим его БПФ.

In [9]:
```
ys = [-0.8, 0.2, 0.3, -0.2]
hs = np.fft.fft(ys)
print(hs)
```

```
[-0.5+0.j  -1.1-0.4j -0.5+0.j  -1.1+0.4j]
```

Возьмем функцию dft, которая вычисляет матрицу синтеза. Сразу применим ее к нашему сигналу.

In [10]:
```
def dft(ys):
    N = len(ys)
    ts = np.arange(N) / N
    freqs = np.arange(N)
    args = np.outer(ts, freqs)
    M = np.exp(1j * PI2 * args)
    amps = M.conj().transpose().dot(ys)
    return amps
```

In [11]:
```
hs2 = dft(ys)
np.sum(np.abs(hs - hs2))
```

Out[11]:
```
3.5650309568570493e-16
```

Далее напишем функцию ff_norec: она разбивает входной массив и использует np.fft.fft, чтобы вычислить БПФ двух половин. И опять же сразу ее

используем.

In [12]:
```
def fft_norec(ys):
    N = len(ys)
    He = np.fft.fft(ys[::2])
    Ho = np.fft.fft(ys[1::2])

    ns = np.arange(N)
    W = np.exp(-1j * PI2 * ns / N)

    return np.tile(He, 2) + W * np.tile(Ho, 2)
```

In [13]:
```
hs3 = fft_norec(ys)
np.sum(np.abs(hs - hs3))
```

Out[13]:
```
0.0
```

И последним шагом заменим np.fft.fft на рекурсию. Это будет реализовано в новой функции fft.

In [14]:
```
def fft(ys):
    N = len(ys)
    if N == 1:
        return ys

    He = fft(ys[::2])
    Ho = fft(ys[1::2])

    ns = np.arange(N)
    W = np.exp(-1j * PI2 * ns / N)

    return np.tile(He, 2) + W * np.tile(Ho, 2)
```

In [15]:
```
hs4 = fft(ys)
np.sum(np.abs(hs - hs4))
```

Out[15]:
```
1.1102230246251565e-16
```

Новая реализация БПФ, которая у нас получилась, занимает время при создании массива или его копировании, пропорциональное $Nlog(N)$. Занимаемое место осталось таким же.

Заключение:

В данной лабороторной работы были изучены разные алгоритмы, такие как ДПФ и БПФ. Были проверены все примеры из файла chap07.ipynb, а также прочитана вся информация в файле. Была создана функция для вычисления БПФ, которая работает быстрее и за время $Nlog(N)$.