Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2.1

з дисципліни «Інтелектуальні вбудовані системи» на тему «Дослідження параметрів алгоритму дискретного перетворення Фур'є»

Виконав:

Студент групи ІП-84
Валигін Андрій

Залікова: 8503

Перевірив:
Регіда Павло Геннадійович

Київ 2021

Завдання

Для згенерованого випадкового сигналу з Лабораторної роботи N 1 відповідно до заданого варіантом (Додаток 1) побудувати його спектр, використовуючи процедуру дискретного перетворення Фур'є. Розробити відповідну програму і вивести отримані значення і графіки відповідних параметрів.

Теоретичні відомості

В основі спектрального аналізу використовується реалізація так званого дискретного перетворювача Фур'є (ДПФ) з неформальним (не формульним) поданням сигналів, тобто досліджувані сигнали представляються послідовністю відліків x(k)

$$F_x(p) = \sum_{k=0}^{N-1} x(k) \cdot e^{-jk\Delta tp\Delta\omega}$$

$$\omega \to \omega_p \to p\Delta\omega \to p \qquad \Delta\omega = \frac{2\pi}{T}$$

ДПФ - проста обчислювальна процедура типу звірки (тобто □-е парних множень), яка за складністю також має оцінку N^2 + N. Для реалізації ДПФ необхідно реалізувати поворотні коефіцієнти ДПФ:

$$W_N^{pk} = e^{-jk\Delta t\Delta\omega p}$$

Ці поворотні коефіцієнти записуються в ПЗУ, тобто є константами.

$$W_N^{pk} = e^{-jk\frac{T}{N}p\frac{2\pi}{T}} = e^{-j\frac{2\pi}{N}pk}$$

Лістинг Програми

```
const harmonicsAmount = 8

const frequency = 1100

const N = 256



const createArray = len => new Array(len).fill(0)

const average = arr => arr.reduce((a, c) => a + c) / arr.length

const mathAverageTrick = arr => Math.sqrt(average(arr.map(num => Math.pow((num
- average(arr)), 2))))



const howLong = cb => {

    const start = window.performance.now()

    cb()

    const end = window.performance.now()

    return end - start

}



const createSignal = len => {

    let array = createArray(len)

    const harmArray = createArray(harmonicsAmount)

    harmArray.forEach((_, i) => array = array.map((item, j) => item +
(Math.random() * Math.sin(((frequency / harmonicsAmount) * (i + 1)) * j +
Math.random()))))

    return array

}



const dftComplexity = number => number * (number - 1)
```

```
const dft = arr => arr.map((_, i) =>
    arr.map((num2, j) =>
        math.multiply(num2, math.exp(
            math.multiply(
                math.complex('-2i'),
                Math.PI * i * j / arr.length)))
    ).reduce((a, c) => math.add(a, c))
)



const fft = arr => {
    const N = arr.length

    if (N <= 1) return arr

    const evens = fft(arr.filter((_, i) => !(i & 1)))
    const odds = fft(arr.filter((_, i) => i & 1))

    const form = index => math.multiply(odds[index],
math.exp(math.multiply(math.complex('-2i'), Math.PI * (index / N))))

    const arr1 = createArray(N / 2).map((_, i) => math.add(evens[i], form(i)))
    const arr2 = createArray(N / 2).map((_, i) => math.subtract(evens[i],
form(i)))

    return [...arr1, ...arr2]
};
```

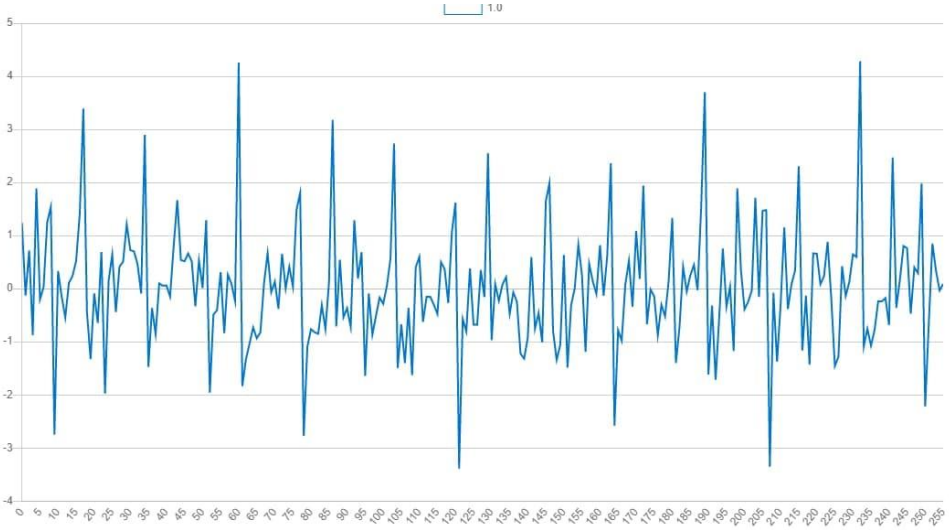
```
const complexToReal = arr => arr.map(obj =>
Math.sqrt(Math.pow(parseFloat(obj.im), 2) + Math.pow(parseFloat(obj.re), 2)))


const option = {
    maintainAspectRatio: false,
    elements: { point: { radius: 0 } },
    scales: {
        yAxes: [{ gridLines: { color: 'rgba(0,0,0,0.2)' } }],
        xAxes: [{ gridLines: { display: false } }]
    }
}


const conf = (name, len, arr) => ({
    type: 'line',
    data: {
        labels: createArray(len).map((_, i) => i),
        datasets: [{
            lineTension: 0,
            label: name,
            backgroundColor: window.chartColors = '#fff',
            borderColor: window.chartColors = '#07c',
            borderWidth: 2,
            fill: false,
            data: arr,
        }]
    },
    options: option,
})
```

```
const newGraphic = (name, arr) => ({
    lineTension: 0,
    label: name,
    backgroundColor: window.chartColors = '#fff',
    borderColor: window.chartColors = '#07c',
    borderWidth: 2,
    fill: false,
    data: arr,
})


//Usage
window.onload = function() {
    const ctx = document.querySelector('#myChart').getContext('2d')
    const sig1 = createSignal(N)
    const graphic = new Chart(ctx, conf('1.0', N, sig1))


    const ctx2 = document.querySelector('#secChart').getContext('2d')
    const dftSig1 = complexToReal(dft(sig1))
    const graphic2 = new Chart(ctx2, conf('2.0', N, dftSig1))



    const ctx3 = document.querySelector('#thirdChart').getContext('2d')
    const fftSig1 = complexToReal(fft(sig1))
    const graphic3 = new Chart(ctx3, conf('3.0', N, fftSig1))


    const ctx4 = document.querySelector('#fourChart').getContext('2d')
```

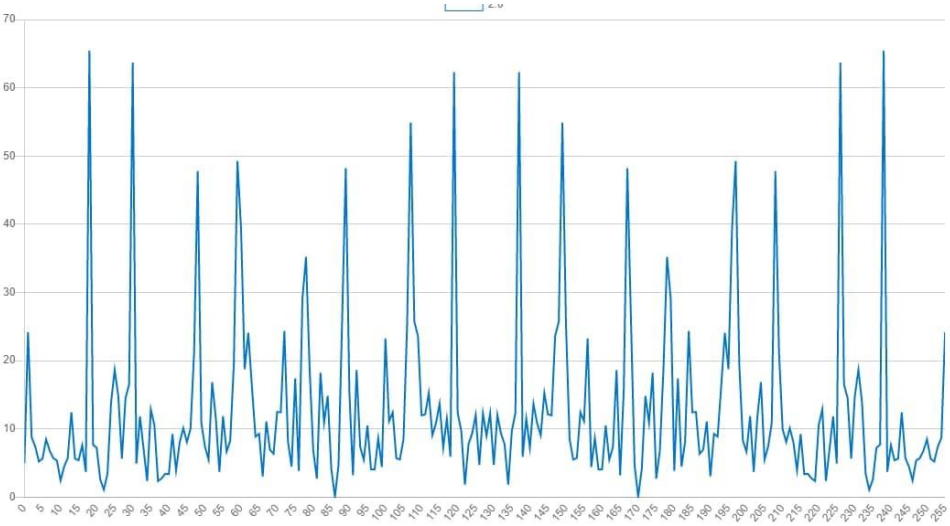
```
    const complexityGFTArr = createArray(N).map((_, i) => howLong(() =>
createSignal(createArray(i + 1))))


    const graphic4 = new Chart(ctx4, conf('4.0', N, complexityGFTArr))
}
```

Скріншоти

Випадковий сигнал



Дискретне перетворення Фур’є



Висновок

У даній роботі розглядалися основні теоритичні відомості про дискретне перетворення Фур’є. У ході роботи створено відповідну програму побудував відповідні графіки. Усі значення були перевірені на мові Python за допомогою вбудованих бібліотек(Нажаль не зміг знайти готової бібліотеки JS) та на сайті

https://www.easycalculation.com/engineering/mechanical/discrete-fourier-transform.php