НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2.2
“Дослідження алгоритму швидкого перетворення Фур'є з проріджуванням відліків сигналів у часі”

Виконав:
студент групи ІП-84
ІП-8408
Засько Євгеній

Київ 2021

Теоретичні відомості

Швидкі алгоритми ПФ отримали назву схеми Кулі-Тьюкі. Всі ці алгоритми використовують регулярність самої процедури ДПФ і те, що будь-який складний коефіцієнт W_N^{pk} можна розкласти на прості комплексні коефіцієнти.

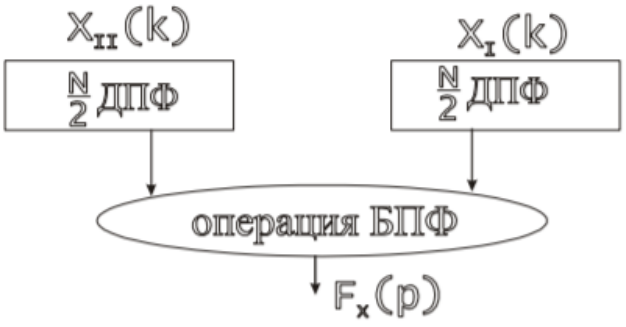
$$W_N^{pk} = W_N^1 W_N^2 W_N^3$$

Для стану таких груп коефіцієнтів процедура ДПФ повинна стати багаторівневою, не порушуючи загальних функціональних зв'язків графа процедури ДПФ. Існують формальні підходи для отримання регулярних графів ДПФ. Всі отримані алгоритми поділяються на 2 класи:

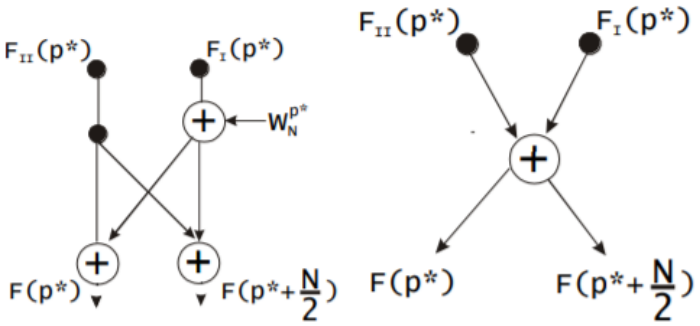
1) На основі реалізації принципу зріджені за часом Хк

2) На основі реалізації принципу зріджені відліків шуканого спектру F(p).

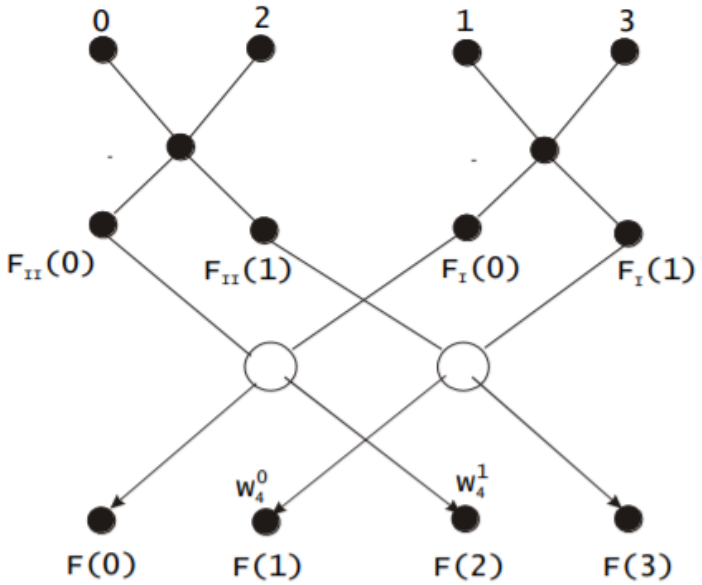
Найпростіший принцип зріджені - поділу на парні/непарні пів-послідовності, які потім обробляють паралельно. А потім знаходять алгоритм, як отримати шуканий спектр. Якщо нам вдасться ефективно розділити, а потім алгоритм отримання спектра, то ми можемо перейти від N ДПФ до N/2 ДПФ.



В процесорах цифрової обробки сигналів спеціальне АЛУ орієнтоване на виконання реалізації ШПФ або за часом, або по частоті. Функції, реалізовані базовими операціями БПФ, визначаються базовими алгоритмами. Операнди представлені як комплексні величини. Особливість операції - 2 операнда на вході, на виході так само 2 результату. Графічно базову операцію БПФ описують так:



Цю базову операцію називають метеликом через форми графа. Для спрощеного аналізу процедури БПФ використовують символічне зображення.



Умови завдання для варіанту бригади

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

Лістинг програми із заданими умовами завдання

```
import kotlin.math.*
import kotlin.system.measureTimeMillis

/*
   Grade book number - 8408
   Variant - 8
 */
const val harmonic = 6
const val frequency = 1500
const val discreteCountDown = 1024

fun main() {
```

```
    val signalGenerator = SignalGenerator(harmonic, frequency, discreteCountDown)
    val plotsDrawer = PlotsDrawer()

    val signal = signalGenerator.generateArray().toList()
    val resultFft = fft(signal).map { it.abs() / signal.size }.filterIndexed { index, _ -> index < signal.size / 2 }
    val resultDft = dft(signal).map { it.abs() / signal.size }.filterIndexed { index, _ -> index < signal.size / 2 }

    plotsDrawer.createPlot("p(fft)", "A", resultFft)
    plotsDrawer.createPlot("p(dft)", "A", resultDft)

    val (fft, dft) = checkTime()

    plotsDrawer.createPlot("discreteCountDown", "time", fft, dft)

    for (k in dft.keys) {
        val d = dft[k]
        val f = fft[k]
        if (d == null || f == null) break
        println("""
            $k: dft - ${d}s, fft - ${f}s, diff - ${d/f}
        """.trimIndent())
    }

}

fun Double.round2(num: Int) = round(this * 10.0.pow(num)) / 10.0.pow(num)

fun W(n: Int): (Int) -> Complex {
    val table = mutableMapOf<Int, Complex>()
    val value = 2 * PI / n
```

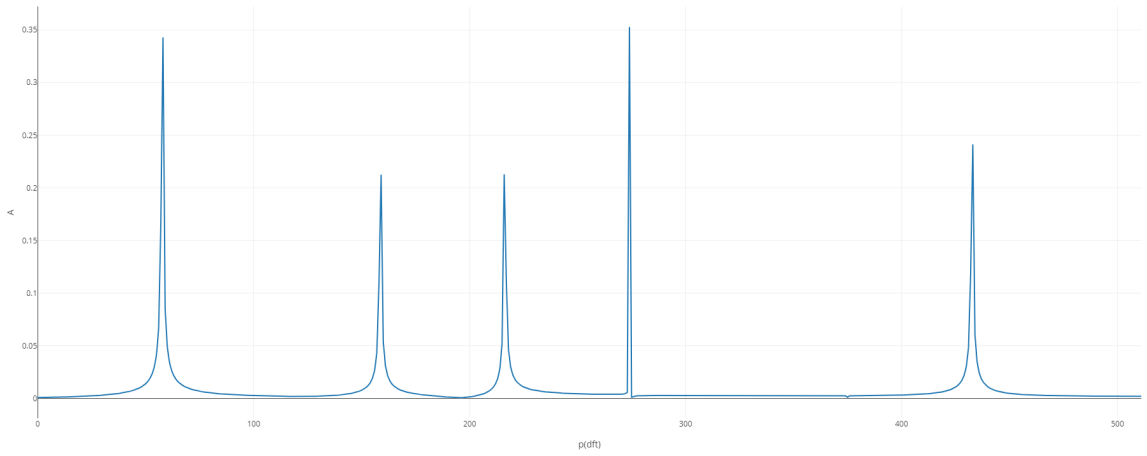
```
    return { k: Int ->
        val arg = (value * k).round2(5)
        table.getOrPut(k % n) {
            Complex(cos(arg), -sin(arg))
        }
    }
}

fun dft(signal: List<Double>): List<Complex> {
    val n = signal.size
    val result = MutableList(n) { complex }

    for (p in 0 until n) {
        var f = complex
        for (k in 0 until n) {
            f += result[p] + signal[k] *
                    (cos(2 * PI * p * k / n) - sin(2 * PI * p * k / n) * (1.0).j)
        }
        result[p] = f
    }
    return result
}

fun fft(signal: List<Double>): List<Complex> {
    val n = signal.size
    val m = n / 2
    val w = W(n)
    return when  {
        n <= 32 -> dft(signal)
        else -> {
            val even = MutableList(m) { 0.0 }
```

```
            val odd = MutableList(m) { 0.0 }

            for (i in 0 until m) {
                even[i] = signal[2 * i]
                odd[i] = signal[2 * i + 1]
            }

            val fftEven = fft(even)
            val fftOdd = fft(odd)

            val result = MutableList(n) { complex }

            for (i in 0 until m) {
                result[i] = fftEven[i] + w(i) * fftOdd[i]
                result[m + i] = fftEven[i] - w(i) * fftOdd[i]
            }

            result
        }
    }
}

fun checkTime(): Pair<Map<Double, Double>, Map<Double, Double>> {
    val timeFft = mutableMapOf<Double, Double>()
    val timeDft = mutableMapOf<Double, Double>()

    var df = 1024
    while (df < 1_048_576) {
        val signal = SignalGenerator(harmonic, frequency, df).generateArray().toList()
        val fft = measureTimeMillis {
            fft(signal)
        }
```

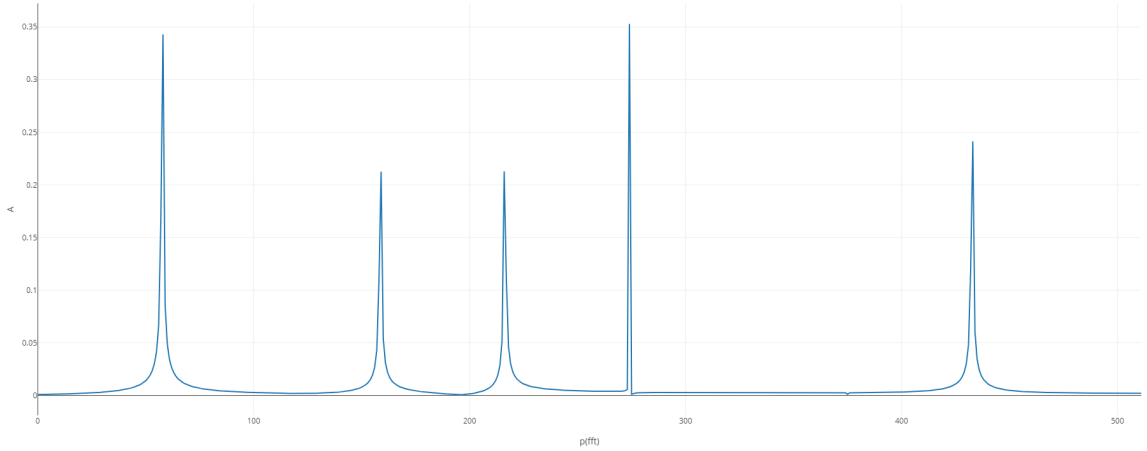
```
        timeFft[df.toDouble()] = fft / 1000.0
        df *= 2
        if (fft > 1_000) break
    }

    var dd = 1024
    while (dd < 1_048_576) {
        val signal = SignalGenerator(harmonic, frequency, dd).generateArray().toList()

        val dft = measureTimeMillis {
            dft(signal)
        }
        timeDft[dd.toDouble()] = dft / 1000.0
        dd *= 2
        if (dft > 5_000) break
    }

    return timeFft to timeDft
}
```

Результати виконання кожної програми

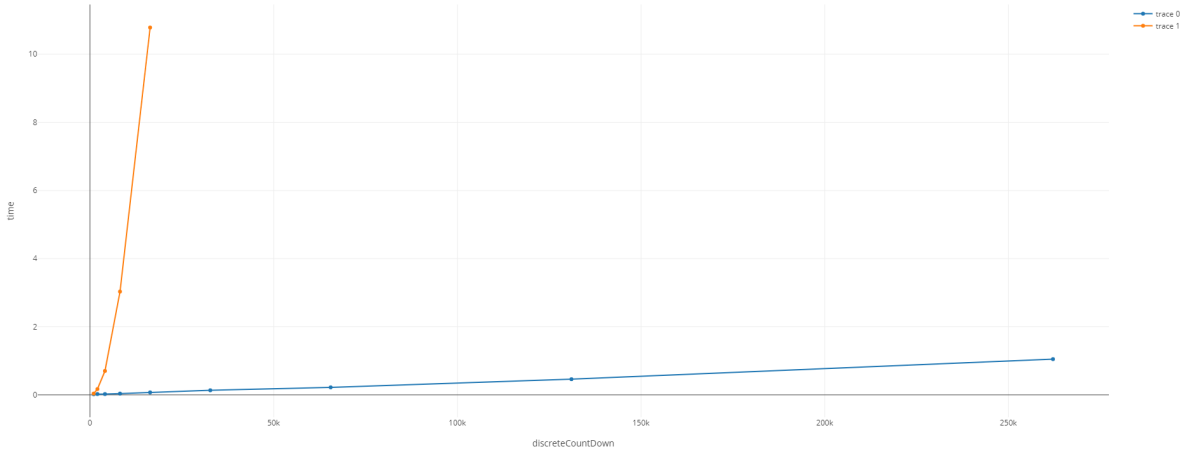
DFT



FFT



Comparing




```
1024.0: dft - 0.057s, fft - 0.024s, diff - 2.375
2048.0: dft - 0.512s, fft - 0.028s, diff - 18.285714285714285
4096.0: dft - 1.745s, fft - 0.031s, diff - 56.29032258064517
8192.0: dft - 8.752s, fft - 0.042s, diff - 208.38095238095238
```

Висновки щодо виконання лабораторної роботи

В ході виконання лабораторної роботи повторив принцип роботи перетворення Фур’є. Було створено програму, яка виконує швидке перетворення Фур’є за складність O(nlogn) замість O(n^2) та будує два графіки dft та fft, а також створено функцію порівняння dft та fft для різною кількості дискретних відліків.