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 № 2.1-2.2

з дисципліни
“Інтелектуальні вбудовані системи”
на теми
“ДОСЛІДЖЕННЯ ПАРАМЕТРІВ АЛГОРИТМУ ДИСКРЕТНОГО ПЕРЕТВОРЕННЯ ФУР’Є",
“ДОСЛІДЖЕННЯ АЛГОРИТМУ ШВИДКОГО ПЕРЕТВОРЕННЯ ФУР'Є З ПРОРІДЖУВАННЯМ ВІДЛІКІВ СИГНАЛІВ У ЧАСІ”

Виконав:
Студент групи ІП-84
Павловський В.Є.
№ ЗК: ІП-8417

Перевірив:
викладач Регіда П.Г.

Київ 2021

Основні 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 $x(k)$

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

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

На всьому інтервалі подання сигналів **T**, **2π** - один період низьких частот. Щоб підвищити точність треба збільшити інтервал **T.**

$$t \to t_k \to k\Delta t \to k; \quad \Delta t = \frac{T}{N} = \frac{1}{k_{зап.}} \cdot f'гр.$$

ДПФ - проста обчислювальна процедура типу звірки (тобто Σ-е парних множень), яка за складністю також має оцінку $\mathbf{N^2 + N}$. 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.

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

W_N^{pk} не залежать від **T**, а лише від розмірності перетворення **N.** Ці коефіцієнти подаються не в експоненційній формі, а в тригонометричній.

$$W_N^{pk} = \cos\left(\frac{2\pi}{N}pk\right) - j\sin\left(\frac{2\pi}{N}pk\right)$$

Ці коефіцієнти повторюються (тому і **p** до **N-1**, і **k** до **N-1**, а **(N-1) • (N-1)**) з періодом **N(2π)**.. Т.ч. в ПЗУ треба зберігати N коефіцієнтів дійсних і уявних частин. Якщо винести знак коефіцієнта можна зберігати **N/2** коефіцієнтів.

2π/N- деякий мінімальний кут, на який повертаються ці коефіцієнти. У ПЗУ окремо зберігаються дійсні та уявні частини компілюють коефіцієнтів. Більш загальна форма ДПФ представляється як:

$$F_x(p) = \sum_{k=0}^{N-1} x(k) \cdot W_N^{pk}$$

ДПФ дуже зручно представити у вигляді відповідного графа.

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.

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

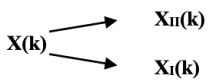
Для стану таких груп коефіцієнтів процедура ДПФ повинна стати багаторівневою, не порушуючи загальних функціональних зв'язків графа процедури ДПФ.

Існують формальні підходи для отримання регулярних графів ДПФ. Всі отримані алгоритми поділяються на 2 класи:

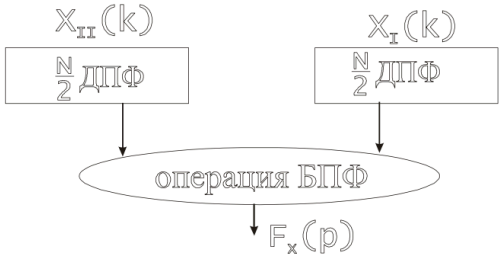
1) На основі реалізації принципу зріджені за часом $\mathbf{X_к}$

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.



Умови завдання для варіанту

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.

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

```
class RandomSignalGenerator {
  constructor(harmonicsNumber, frequencyCutoff, discreteSamplesNumber) {
    this.harmonicsNumber = harmonicsNumber;
    this.frequencyCutoff = frequencyCutoff;
    this.discreteSamplesNumber = discreteSamplesNumber;

    this.signals = [];
  }

  calculateSignal(amplitude, frequency, time, phase) {
    return amplitude * Math.sin(frequency * time + phase);
  }

  generateSignals() {
    let signals = Array(this.discreteSamplesNumber).fill(0);

    for (let i = 1; i ⩽ this.harmonicsNumber; i++) {
      let frequency = (i * this.frequencyCutoff) / this.harmonicsNumber;

      let phase = Math.random();
      let amplitude = Math.random();

      for (let time = 0; time < this.discreteSamplesNumber; time++) {
        let signal = this.calculateSignal(amplitude, frequency, time, phase);
```

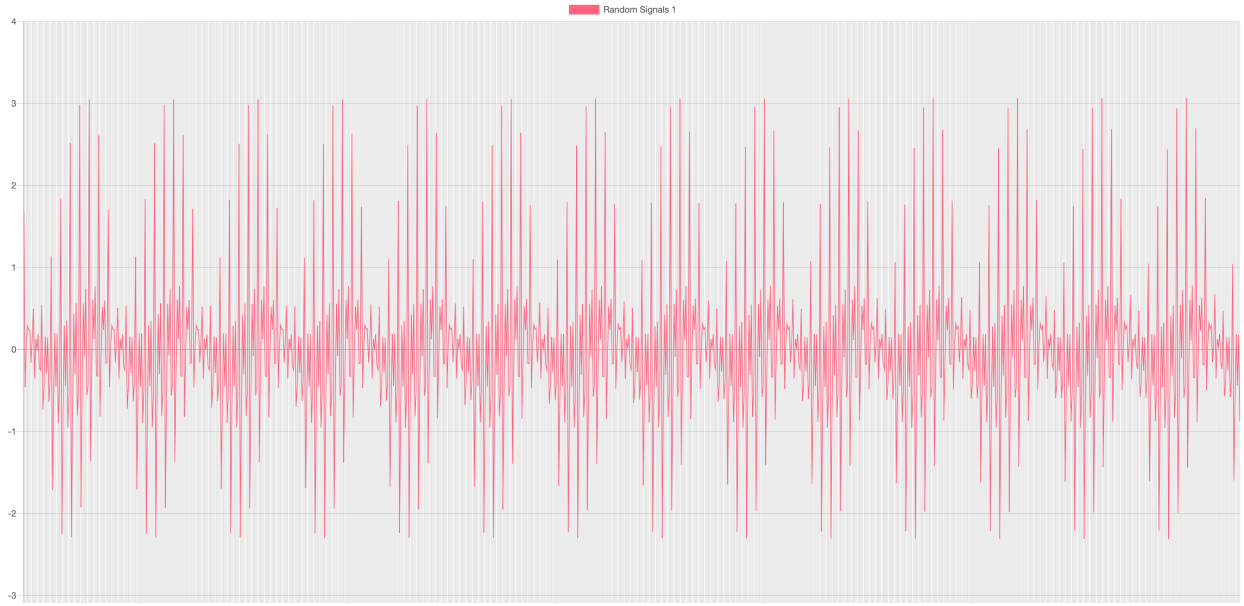
```
        signals[time] += signal;
      }
    }

    return signals;
  }
}

const mean = (signals) =>
  signals.reduce((prev, curr) => prev + curr, 0) / signals.length;

const variance = (signals) => {
  let Mx = mean(signals);

  return (
    signals
      .map((num) => Math.pow(num - Mx, 2))
      .reduce((prev, curr) => prev + curr, 0) /
    (signals.length - 1)
  );
};

function createComplex(real, imag) {
  return {
    real: real,
    imag: imag,
  };
}

const complexAdd = function (a, b) {
  return [a[0] + b[0], a[1] + b[1]];
};

const complexSubtract = function (a, b) {
  return [a[0] - b[0], a[1] - b[1]];
};

const complexMultiply = function (a, b) {
  return [a[0] * b[0] - a[1] * b[1], a[0] * b[1] + a[1] * b[0]];
};

const complexMagnitude = function (c) {
  return Math.sqrt(c[0] * c[0] + c[1] * c[1]);
```

```
};

var mapExponent = {},
  exponent = function (k, N) {
    var x = -2 * Math.PI * (k / N);

    mapExponent[N] = mapExponent[N] || {};
    mapExponent[N][k] = mapExponent[N][k] || [Math.cos(x), Math.sin(x)]; // [Real, Imaginary]

    return mapExponent[N][k];
  };

var dft = function (vector) {
  var X = [],
    N = vector.length;

  for (var k = 0; k < N; k++) {
    X[k] = [0, 0]; //Initialize to a 0-valued complex number.

    for (var i = 0; i < N; i++) {
      var exp = exponent(k * i, N);
      var term;
      if (Array.isArray(vector[i])) term = complexMultiply(vector[i], exp);
      //If input vector contains complex numbers
      else term = complexMultiply([vector[i], 0], exp); //Complex mult of the signal with the
exponential term.
      X[k] = complexAdd(X[k], term); //Complex summation of X[k] and exponential
    }
  }

  return X;
};

function fft(vector) {
  var X = [],
    N = vector.length;

  // Base case is X = x + 0i since our input is assumed to be real only.
  if (N == 1) {
    if (Array.isArray(vector[0]))
      //If input vector contains complex numbers
      return [[vector[0][0], vector[0][1]]];
    else return [[vector[0], 0]];
```

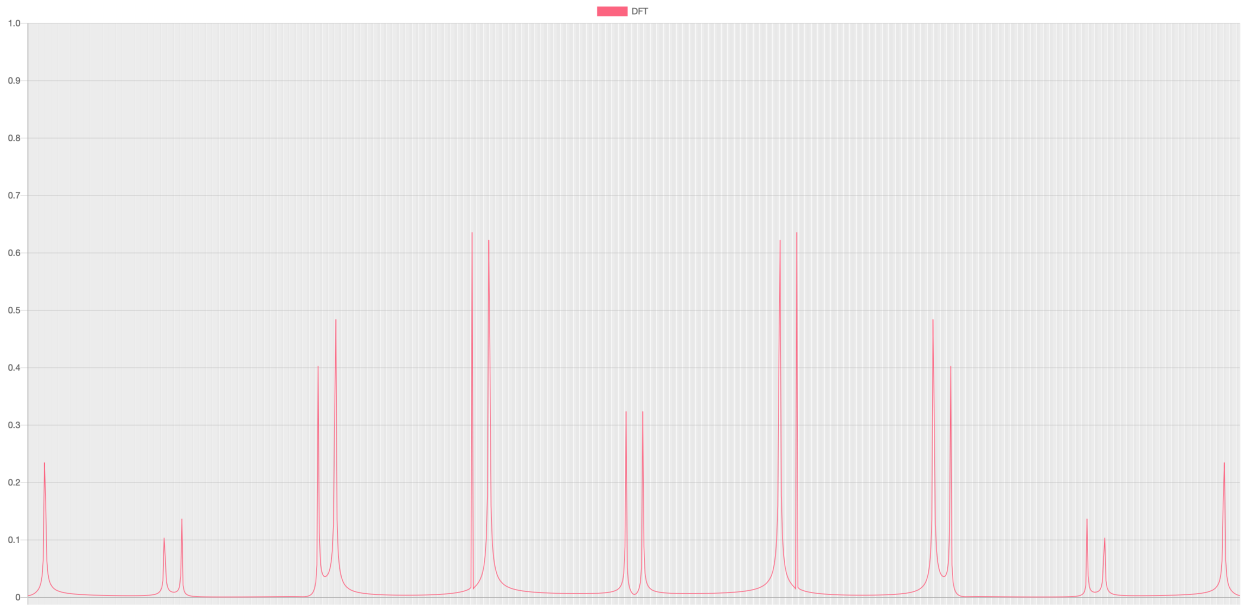
```
  }

  // Recurse: all even samples
  var X_evens = fft(vector.filter(even)),
    // Recurse: all odd samples
    X_odds = fft(vector.filter(odd));

  // Now, perform N/2 operations!
  for (var k = 0; k < N / 2; k++) {
    // t is a complex number!
    var t = X_evens[k],
      e = complexMultiply(exponent(k, N), X_odds[k]);

    X[k] = complexAdd(t, e);
    X[k + N / 2] = complexSubtract(t, e);
  }

  function even(__, ix) {
    return ix % 2 == 0;
  }

  function odd(__, ix) {
    return ix % 2 == 1;
  }

  return X;
}
```

Результати виконання програми

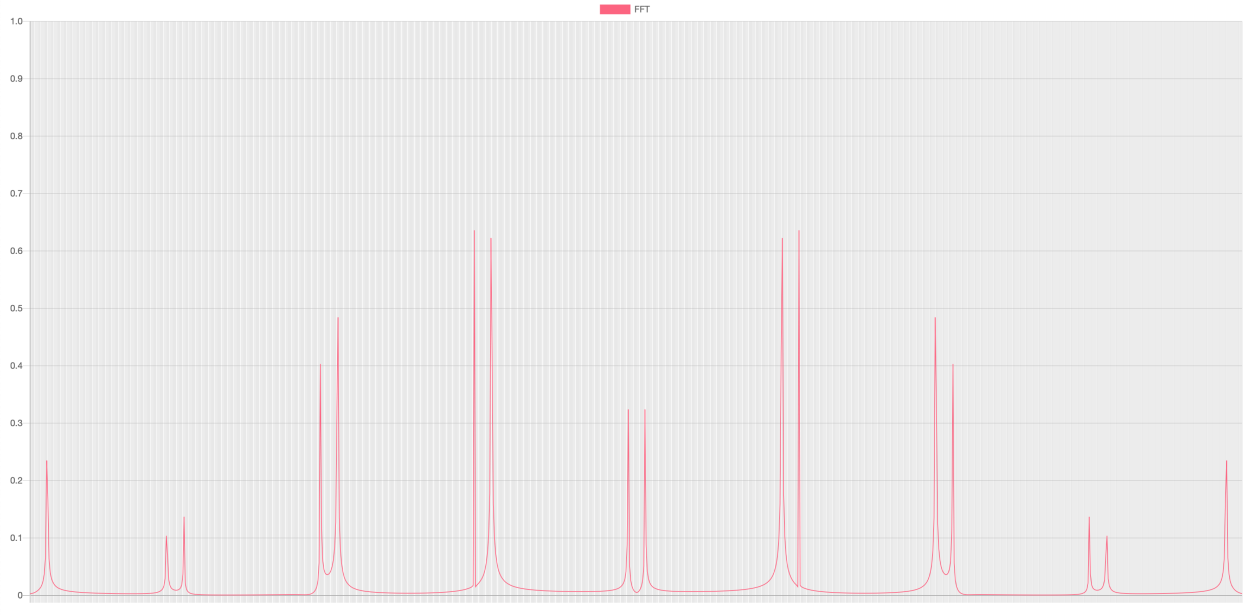
Сигнали



DFT



FFT



Висновки

Під час даної лабораторної роботи ми вивчили, як виділяти частотний спектр з сигналу за допомогою дискретного перетворення Фур’є та як можна пришвидшити дискретне перетворення Фур’є.