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
Гудь В.В.
№ залікової книжки: ІП-8405

Перевірив:
викладач
Регіда П.Г.

Київ 2021

Теоретичні відомості

3.1. 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 $x(k)$

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

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

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

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

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:

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

Коефіцієнти зручно представити у вигляді таблиці:

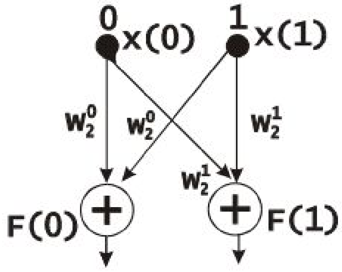
k \ p	0	1	2	3
0	W_4^0	W_4^0	W_4^0	W_4^0
1	W_4^0	W_4^1	W_4^2	W_4^3
2	W_4^0	W_4^2	W_4^0	W_4^2
3	W_4^0	W_4^3	W_4^2	W_4^1

Різних тут всього 4 коефіцієнта:

$$W_4^0 = \cos\left(\frac{2\pi}{4}\cdot 0\right) - j\sin\left(\frac{2\pi}{4}\cdot 0\right) = 1 \qquad (W_4^1 = -j;\ W_4^2 = -1;\ W_4^3 = +j)$$

Можна в пам'яті зберігати тільки 2, а решта брати з "-", якщо $\frac{N}{2} - 1 < pk$. 4 ДПФ це вироджені перетворення, по модулю ці коефіцієнти = 1 і всі 4 ДПФ можуть реалізуватися на 24-х суматора. Це буде далі використовуватися в реалізації ШПФ з основою 4.

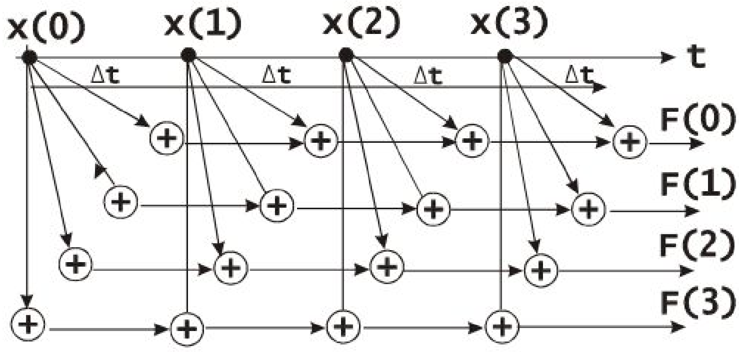
2ДПФ реалізується ще простіше:



($W_2^0 = +1$; $W_2^1 = -1$)

Спеціальна схема реалізації ДПФ з активним використанням пауз між відліками

При реалізації ДПФ можна організувати обробку в темпі надходження даних. Реалізація схеми в БПФ з активним використанням пауз на 4-х точках виглядає так:



Ця схема сильно залежить от Δt и N.

Умови завдання

Варіант 5:

n = 14, ω гр = 2000, N = 264

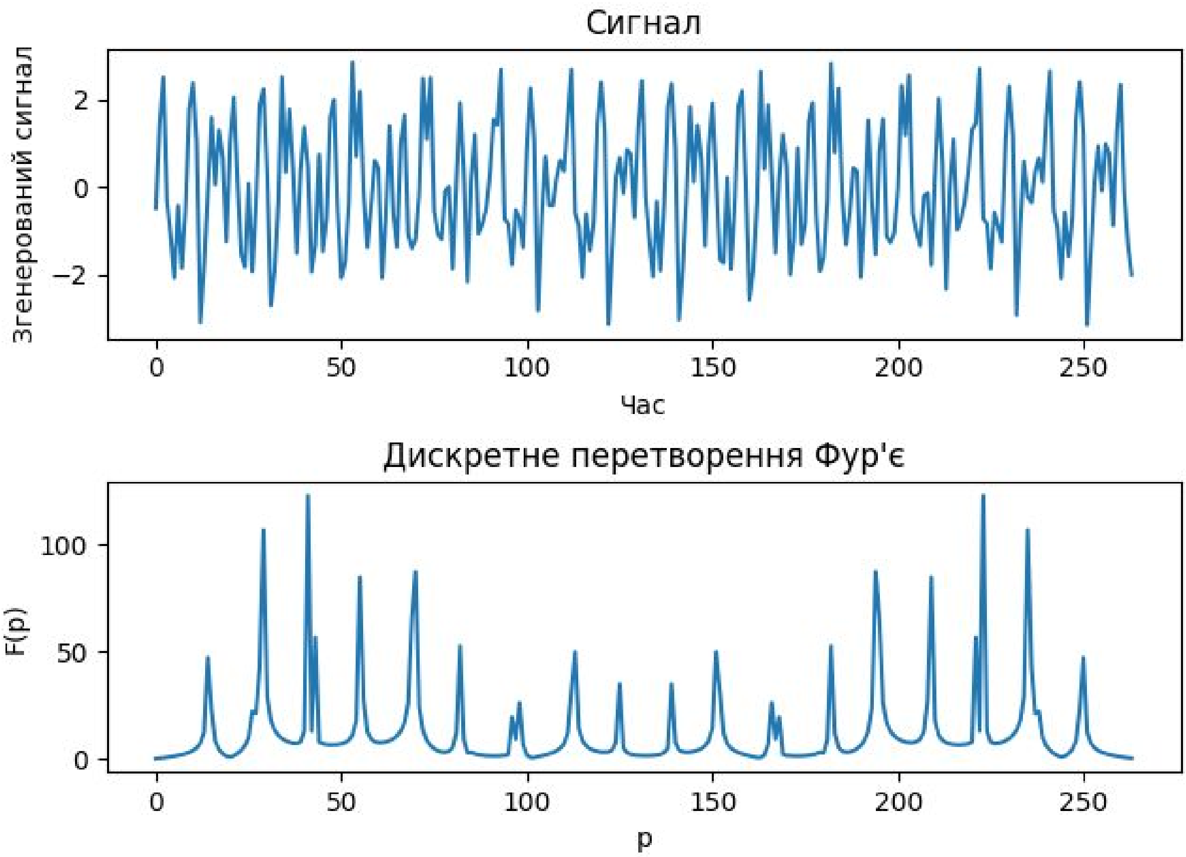
Вихідний код

```
import numpy as np
import matplotlib.pyplot as plotter
n = 14
w = 2000
N = 264


def generate_signal(amplitude, phase, frequency, time):
```

```
    return amplitude * np.sin(frequency * time + phase)


def signal_generator(harmonics, frequency_max, samples_amount):
    signal_collection = np.zeros(samples_amount)
    for f in range(1, harmonics + 1):
        phase = np.random.uniform((-np.pi / 2), (np.pi / 2))
        amplitude = np.random.uniform(0, 1)
        frequency = (f * frequency_max) / harmonics
        for time in range(samples_amount):
            signal_collection[time] += generate_signal(amplitude, phase, frequency, time)
    return signal_collection

import numpy as np


def expectation(signal):
    sig_sum = 0
    for i in range(len(signal)):
        sig_sum += signal[i]
    expect = sig_sum / len(signal)
    return expect


def correlate(x, y, t):
    x_expectation = expectation(x)
    y_expectation = expectation(y)
    x_length = len(x)
    sig_sum = 0
    for time in range(x_length - t):
        sig_sum += (x[time] - x_expectation) * (y[time + t] - y_expectation)
    corr = sig_sum / (x_length - 1)
    return corr


def correlation(x, y, step):
    correlations = np.zeros(step)
    for t in range(step):
        correlations[t] = correlate(x, y, t)
    return correlations


def autocorrelation(x, step):
    return correlation(x, x, step)


def w(p, k, N):
    internal = 2 * np.pi * p * k / N
```

```
    return complex(np.cos(internal), np.sin(internal))


def discrete_fourier_transform(signal):
    N = len(signal)
    spectre = np.zeros(N)
    for p in range(N):
        sum = 0
        for k in range(N):
            xk = signal[k]
            wpkN = w(p, k, N)
            sum += xk * wpkN
        spectre[p] = abs(sum)
    return spectre


signal = signal_generator(n, w, N)
spectre = discrete_fourier_transform(signal)

figure, axis = plotter.subplots(2, 1)

plotter.subplots_adjust(left=0.1, top=0.9, bottom=0.1, right=0.99, hspace=0.5)

axis[0].plot(range(N), signal)
axis[0].set_title("Сигнал")
axis[0].set(xlabel='Час', ylabel='Згенерований сигнал')

axis[1].plot(range(N), spectre)
axis[1].set_title("Дискретне перетворення Фур'є")
axis[1].set(xlabel='p', ylabel='F(p)')
plotter.show()
```

Результати роботи програми



Висновки

У ході виконання лабораторної роботи ми ознайомилися з принципами реалізації спектрального аналізу випадкових сигналів на основі алгоритму перетворення Фур'є, вивчили та дослідили особливості даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок.

.