**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 №2.2**

з дисципліни  
«Інтелектуальні вбудовані системи»

на тему  
«ДОСЛІДЖЕННЯ АЛГОРИТМУ ШВИДКОГО ПЕРЕТВОРЕННЯ ФУР'Є З

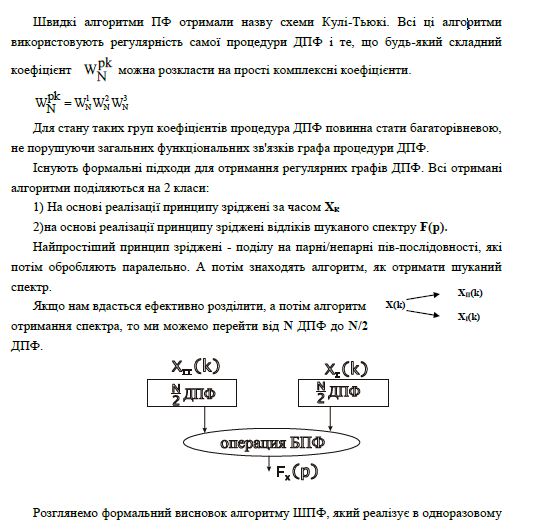
ПРОРІДЖУВАННЯМ ВІДЛІКІВ СИГНАЛІВ У ЧАСІ»

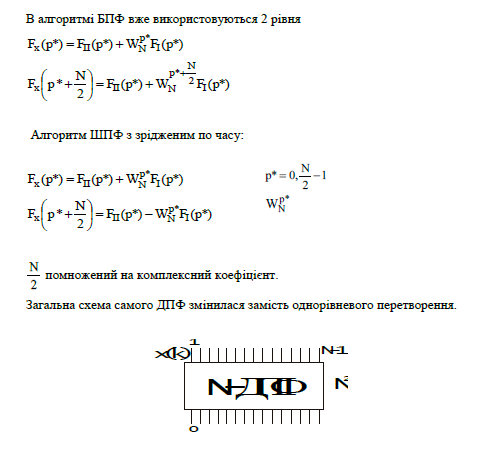
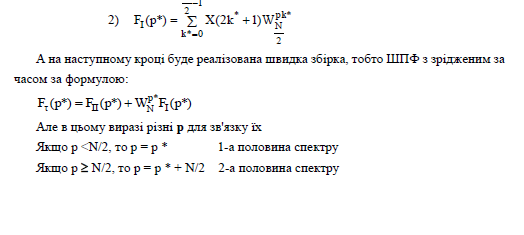
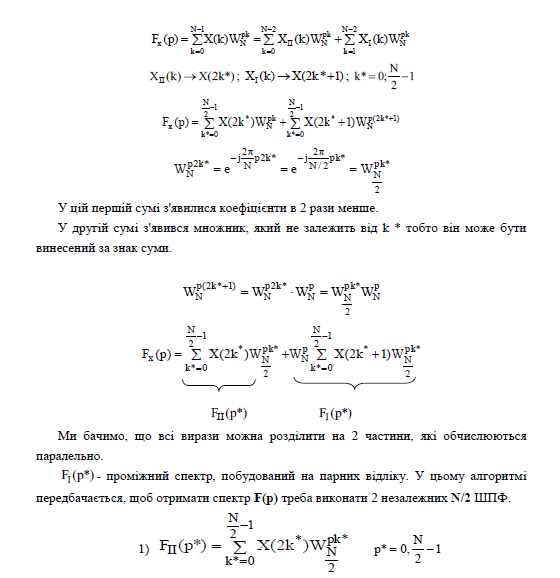
Виконав: Перевірив:

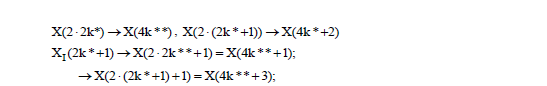
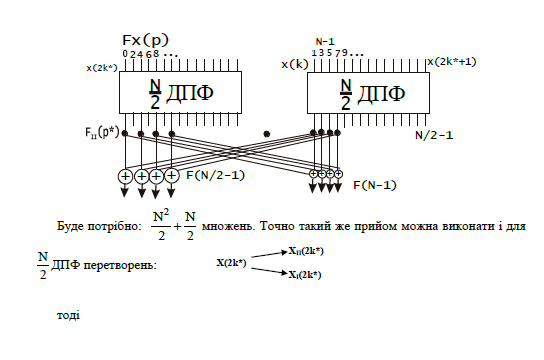
студент групи ІП-84 викладач  
Кучін Владислав Дмитрович Регіда Павло Геннадійович  
номер залікової книжки: 8415

Київ 2021

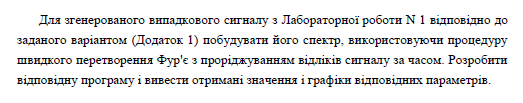
**Основні теоретичні відомості**

****

****

****

**Завдання на лабораторну роботу**



**Варіант-15**

Число гармонік в сигналі: 14.

Гранична частота: 2500.

Кількість дискретних відліків: 64

**Лістинг програми**

import time

import numpy as np

import matplotlib.pyplot as plp

W\_MAX = 2500

HARMONICS\_QUANTITY = 14

DISCRETE\_SAMPLES\_NUMBER = 64

def calculate\_value(amplitude, phase, frequency, time):

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

def create\_signal(w\_max, harmonics\_quantity, discrete\_samples\_number):

sum\_values = np.zeros(discrete\_samples\_number)

sampling\_frequency = 2 \* w\_max

for w in range(1, harmonics\_quantity + 1):

phase = np.random.uniform(-np.pi / 2, np.pi / 2)

amplitude = np.random.uniform(0.0, 1000.0)

frequency = 2 \* np.pi \* w\_max / harmonics\_quantity \* w

for t in range(discrete\_samples\_number):

sum\_values[t] += calculate\_value(amplitude, phase, frequency, t / sampling\_frequency)

return sum\_values

def calc\_w\_pkn(p, k, n):

return complex(np.cos(2 \* np.pi / n \* p \* k), np.sin(2 \* np.pi / n \* p \* k))

def discrete\_fourier\_transform(signal):

result = np.zeros(len(signal), dtype=np.complex)

for p in range(len(signal)):

for k in range(len(signal)):

w = calc\_w\_pkn(p, k, len(signal))

result[p] += w \* signal[k]

return result

twiddle\_factors = None

def calculate\_twiddle\_factors(k):

global twiddle\_factors

twiddle\_factors = dict()

size = 1

while size <= k:

twiddle\_factors[size] = dict()

for i in range(size):

twiddle\_factors[size][i] = calc\_w\_pkn(1, i, size)

size \*= 2

def fast\_fourier\_transform(signal):

if len(signal) == 2:

return discrete\_fourier\_transform(signal)

half\_N = int(len(signal) / 2)

even\_indexes = np.zeros(half\_N)

odd\_indexes = np.zeros(half\_N)

for m in range(half\_N):

even\_indexes[m] = signal[2 \* m]

odd\_indexes[m] = signal[2 \* m + 1]

even\_values = fast\_fourier\_transform(even\_indexes)

odd\_values = fast\_fourier\_transform(odd\_indexes)

result = np.zeros(len(signal), dtype=complex)

for k in range(half\_N):

twiddle\_value = odd\_values[k] \* twiddle\_factors[len(signal)][k]

result[k] = even\_values[k] + twiddle\_value

result[k + half\_N] = even\_values[k] - twiddle\_value

return result

EXPERIMENTS = 10

fft\_times = list()

dft\_times = list()

count = list()

signals = list()

for i in range(EXPERIMENTS):

signal\_length = DISCRETE\_SAMPLES\_NUMBER \* (2 \*\* i)

signals.append(create\_signal(W\_MAX, HARMONICS\_QUANTITY, signal\_length))

count.append(signal\_length)

for i in range(EXPERIMENTS):

calculate\_twiddle\_factors(len(signals[i]))

start\_time = time.perf\_counter\_ns()

fft = fast\_fourier\_transform(signals[i])

deltaTime = time.perf\_counter\_ns() - start\_time

fft\_times.append(deltaTime)

print(deltaTime)

if i < 6:

start\_time = time.perf\_counter\_ns()

dft = discrete\_fourier\_transform(signals[i])

deltaTime = time.perf\_counter\_ns() - start\_time

dft\_times.append(deltaTime)

SIGNAL = create\_signal(W\_MAX, HARMONICS\_QUANTITY, DISCRETE\_SAMPLES\_NUMBER)

calculate\_twiddle\_factors(len(SIGNAL))

my\_fft = abs(fast\_fourier\_transform(SIGNAL))

fft = abs(np.fft.fft(SIGNAL))

time\_period = DISCRETE\_SAMPLES\_NUMBER / (2 \* W\_MAX)

plp.subplot(221)

plp.title('Час виконання FFT від кількості дискретних відліків')

plp.plot(count, fft\_times)

plp.subplot(222)

plp.title('Час виконання DFT від кількості дискретних відліків')

plp.plot(count[:6], dft\_times)

plp.subplot(223)

plp.title('Швидке перетворення Фур\'є (частотний спектр)')

plp.plot(np.linspace(0, 2 \* W\_MAX, num=DISCRETE\_SAMPLES\_NUMBER), my\_fft)

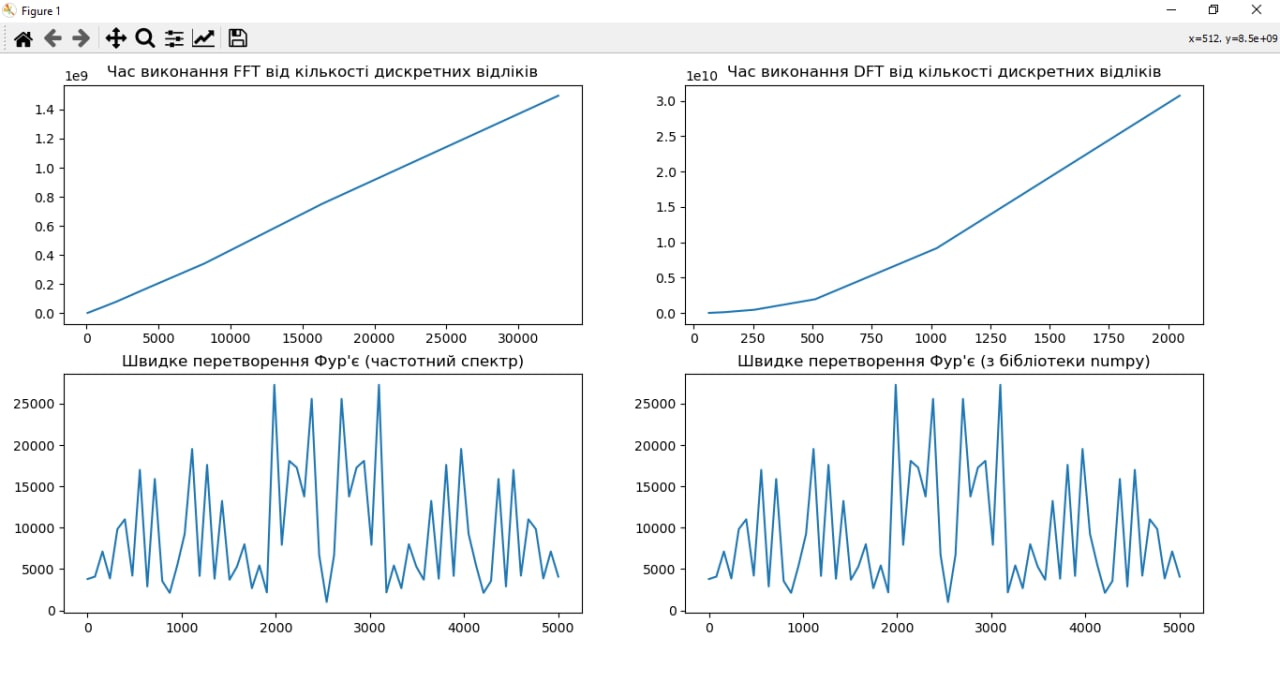
plp.subplot(224)

plp.title('Швидке перетворення Фур\'є (з бібліотеки numpy)')

plp.plot(np.linspace(0, 2 \* W\_MAX, num=DISCRETE\_SAMPLES\_NUMBER), fft)

plp.show()

**Результати роботи програми**



**Висновки**

Під час даної лабораторної роботи ми вивчили, як можна пришвидшити дискретне перетворення Фур’є (складність O(N2)) за допомогою алгоритму Кулі-Тьюкі (складність O(Nlog(N)).