Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук і кібернетики

Звіт

з лабораторної роботи №1

з предмету «Моделювання складних систем»

Варіант 6

Виконав:

Студент групи ІПС-31

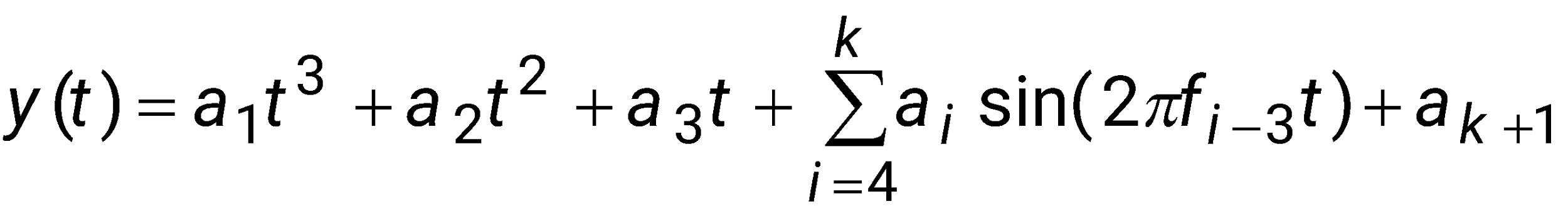
Дубина Андрій Володимирович

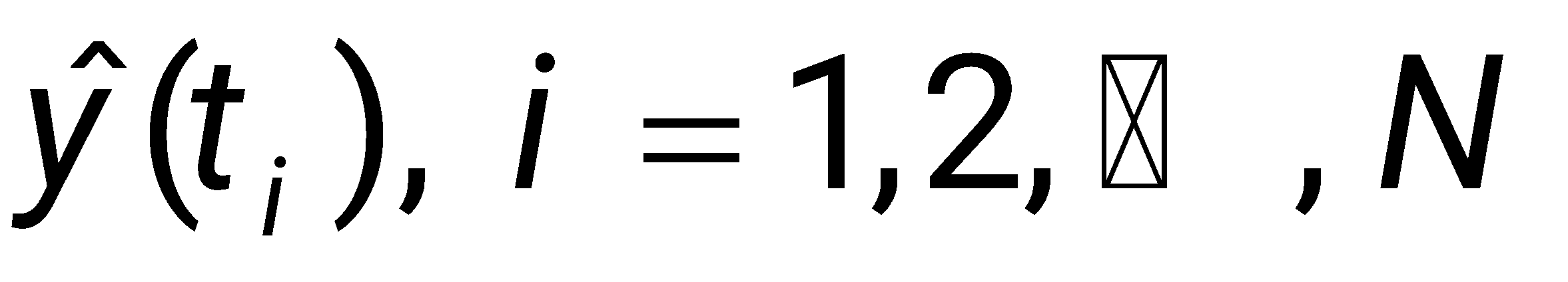
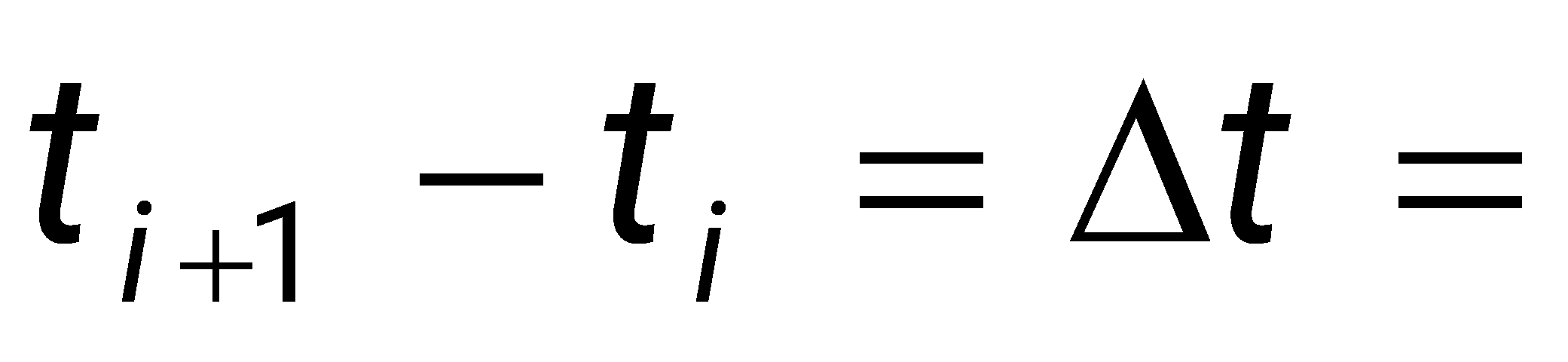
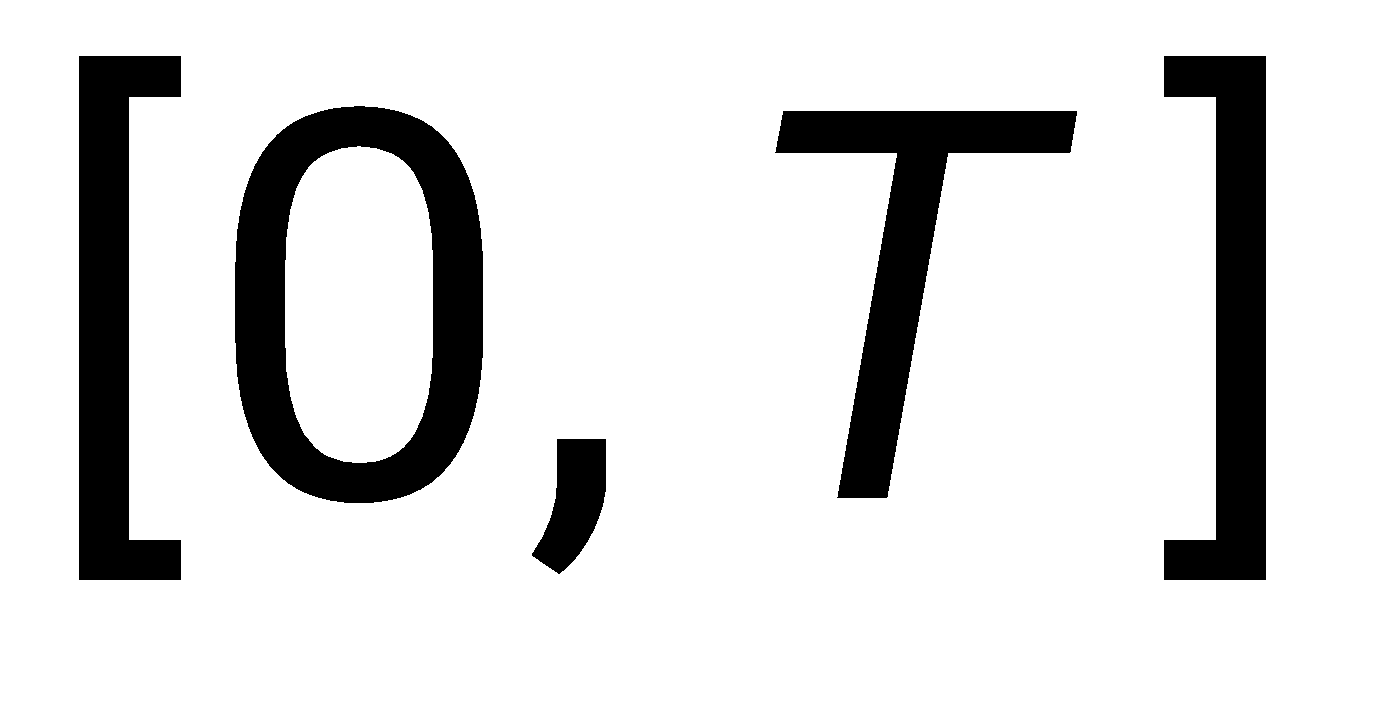
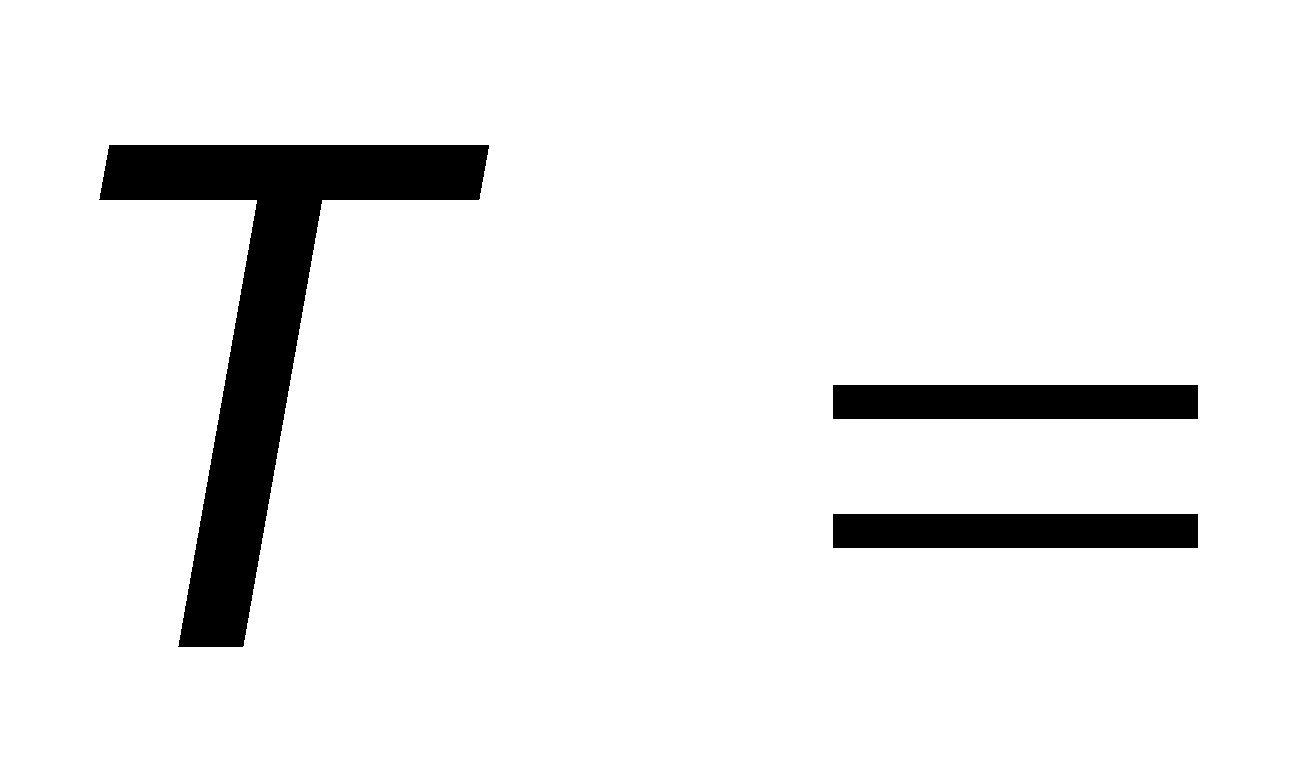
Київ

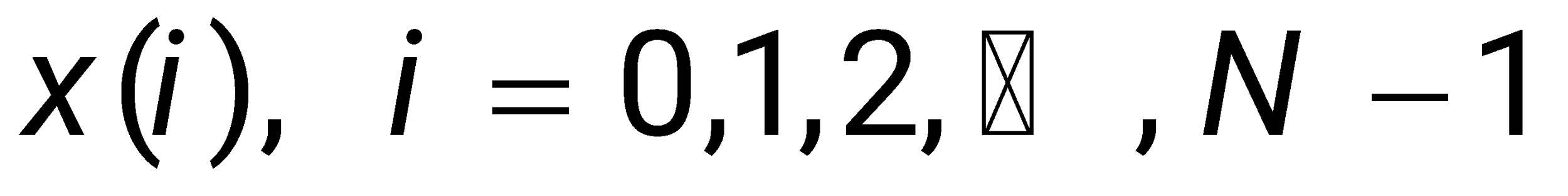
2024

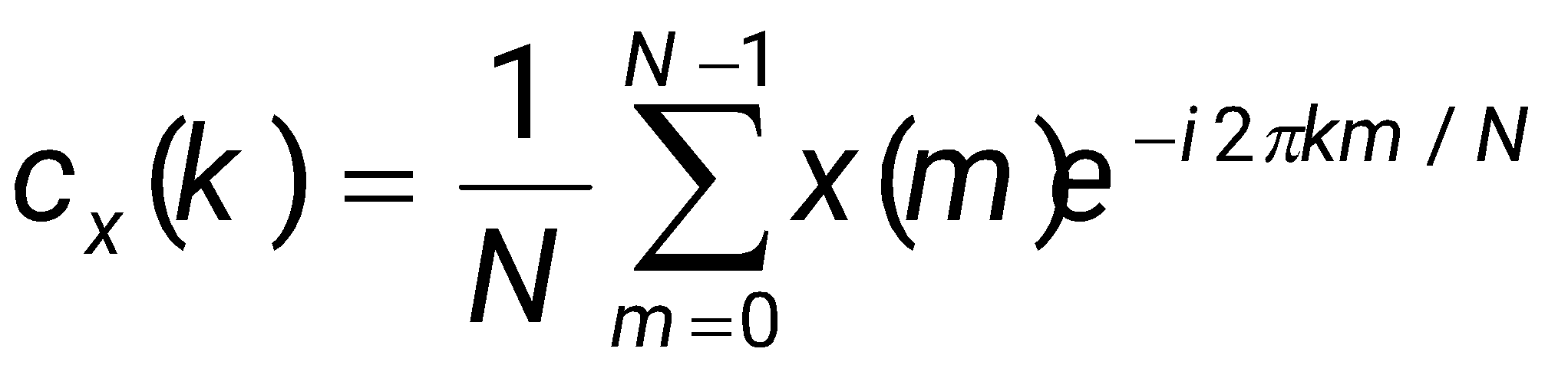
1. Умова

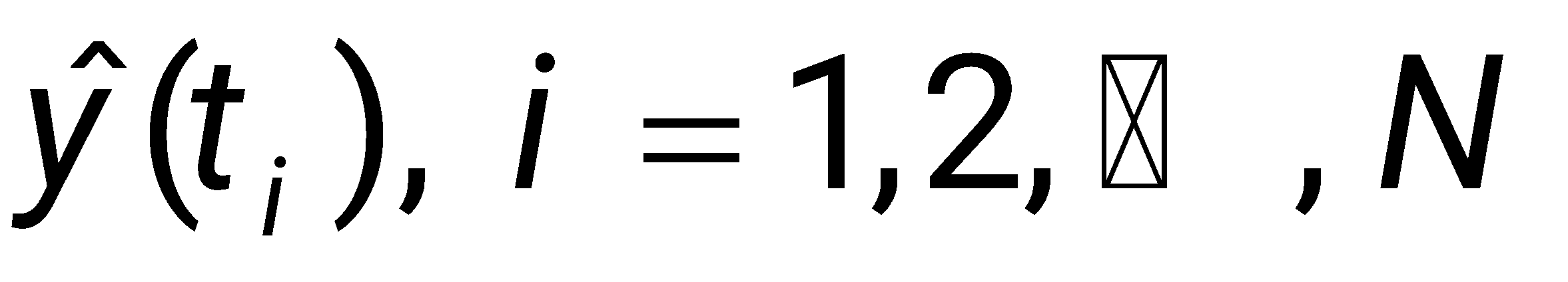
Визначити модель в класі функцій



для спостережуваної дискретної функції , (відповідний файл fk.txt), 0.01, інтервал спостереження , 5.

Дискретне перетворення Фур’є для дискретної послідовності 

.

Для виконання завдання маємо для дискретної послідовності  виконати задане дискретне перетворення Фур’є, після чого визначити місця, де присутні екстремуми («піки» на графіку), при чому шукаємо такі частоти у першій половині графіку (друга половина дискретного перетворення Фур’є симетрична першій), і додаємо такі частоти в список через формулу

При найявності списку екстремумів, апроксимуємо функцію. Для апроксимації функції використовується система рівнянь, що враховує поліноміальні члени (до ) та синусоїдальні компоненти на знайдених частотах. Формується матриця коефіцієнтів системи та вектор правих частин. Матриця має розмірність 5 (так як маємо 5 коефіцієнтів). В першому рядку використовуються суми степенів , оскільки ми апроксимуємо поліномом третього ступеня. Кожен член моделі містить час 𝑡, тому необхідно піднімати 𝑡 до степенів, що відображають різні компоненти полінома. Наприклад:  
lля першого коефіцієнта ​, що стоїть при , в рівнянні буде з'являтися сума , адже . Аналогічно, для потрібен степінь, для —, і так далі.

Відрізняються четвертий і п’ятий рядок. У четвертому, через умову, використовуються сінуси, а в п’ятому, так як , останнім елементом стоїть N, тобто кількість точок.

Вектор же собою представляє суму добутку спостережуваних значень y(t) на відповідні базові функції.

Матриця і вектор потрібні для вирішення рівняння:

*,*

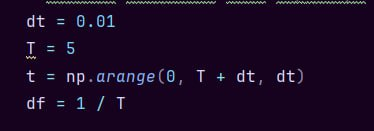
де А – матриця коефіцієнтів; c – вектор правих частин; а – вектор шуканих коефіцієнтів.

Шукаємо похибку, і впевнюючись, що вона достатньо мала, записуємо результат.

1. Хід роботи

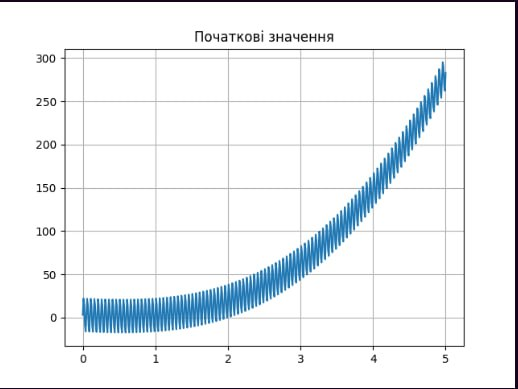
Робота виконана мовою Python.

Спочатку введемо задані умови:

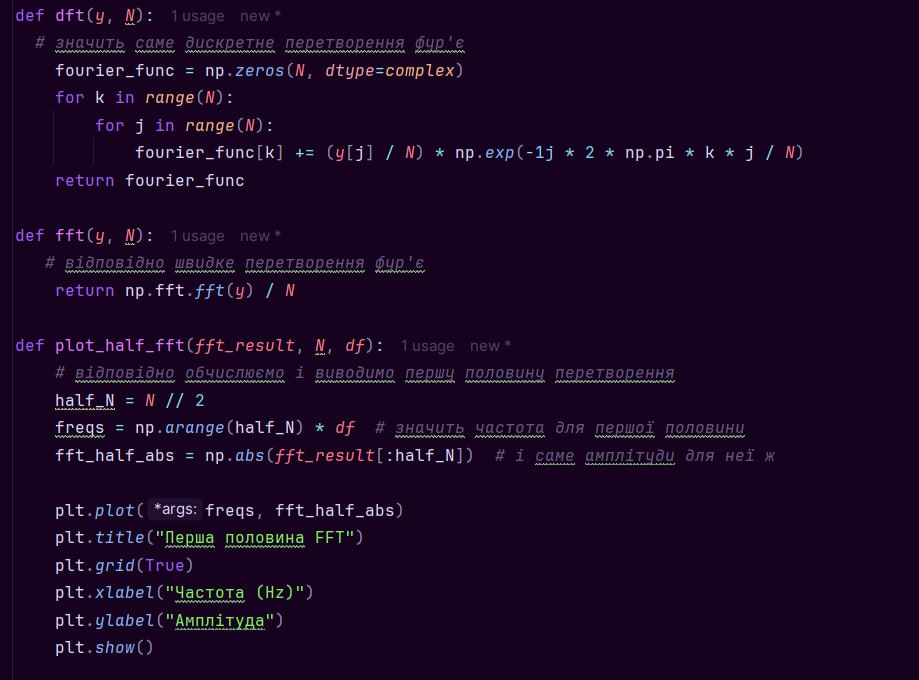


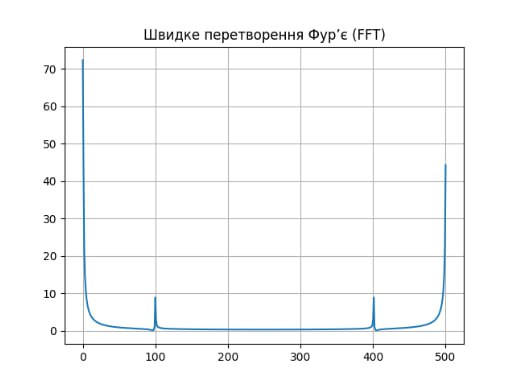
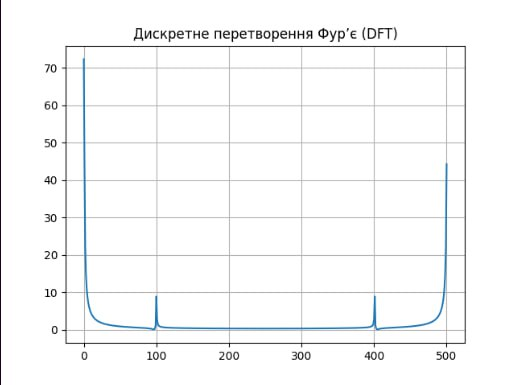
Підвантажуємо дані з наданого текстового файлу та виводимо графік заданої функції (тут і надалі використовую лише імплементації функцій, без викликів у main).

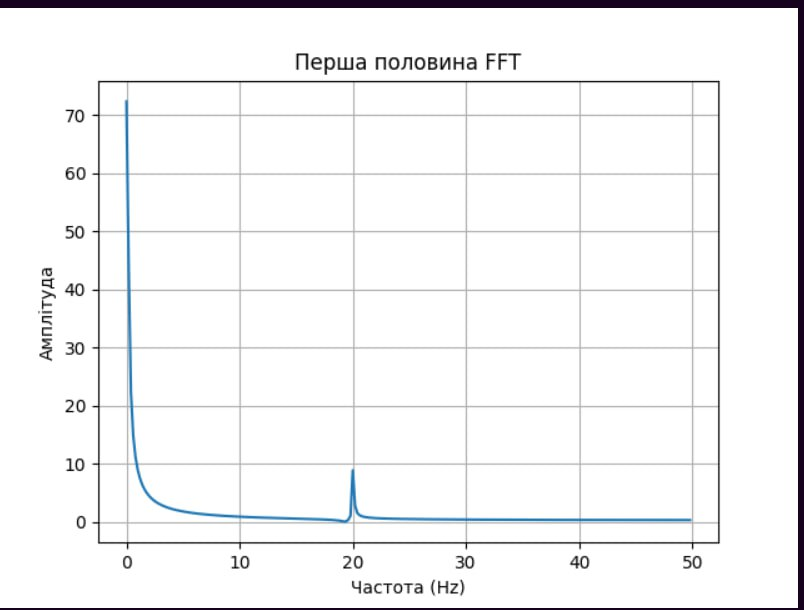




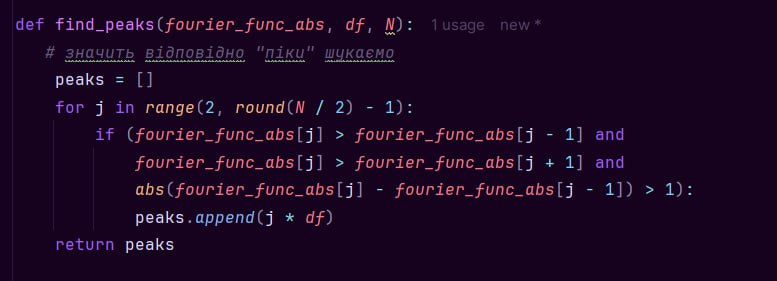
Виконуємо дискретне перетворення Фур’є. Для порівняння, виведемо також швидке перетворення Фур’є з бібліотеки numpy. Після цього виведемо першу половину перетворення.

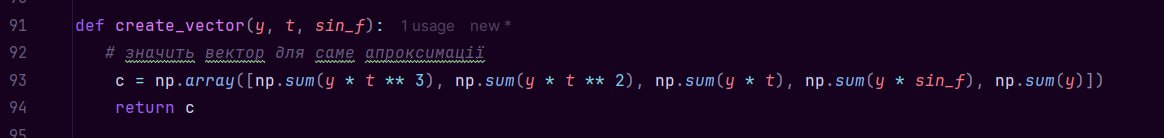
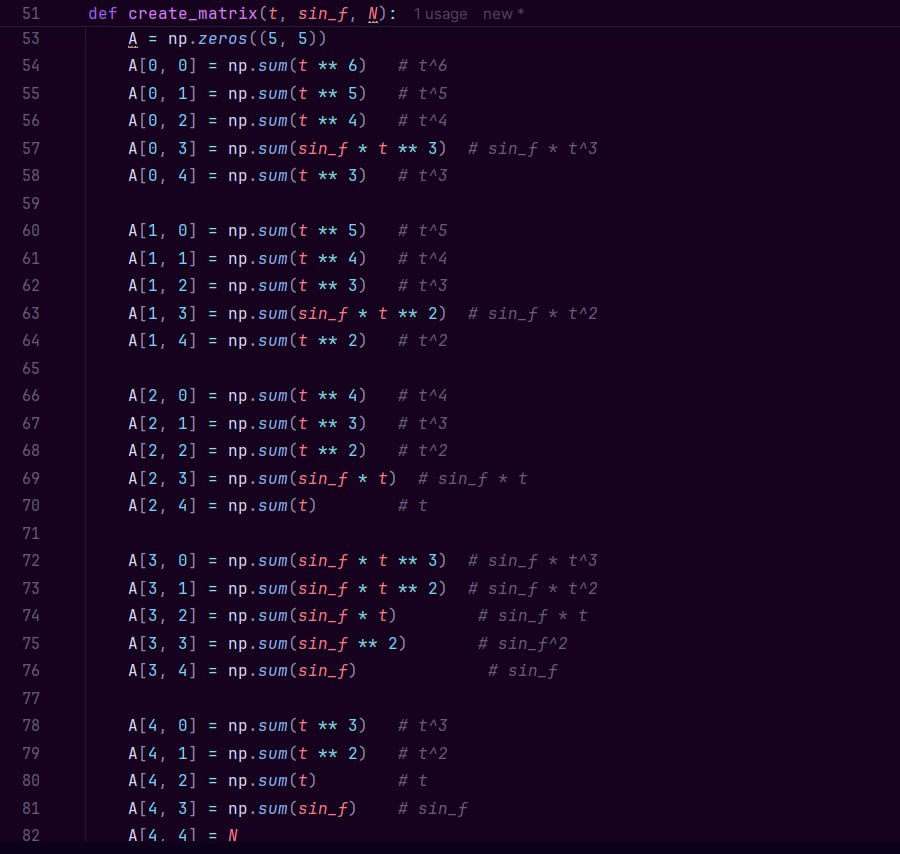




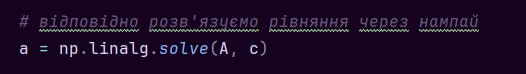


Далі шукаємо екстремуми на цій половині:

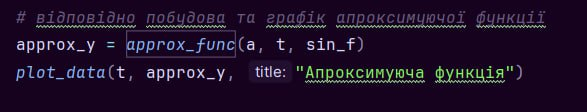
При знаходженні створюємо матрицю і вектор для апроксимації:



Вирішуємо це рівняння за допомогою numpy:



За допомогою ініціалізованих раніше функцій будуємо та виводимо графік апроксимованої функції:



В консолі бачимо коефіцієнти та похибку.



Апроксимуюча функція:

Маємо відносно малу похибку (приблизно ), тому апроксимуюча функція досить близька до спостережень.

Повний код:

import numpy as np

import matplotlib.pyplot as plt

def load\_data(*file\_name*):

*#відповідно завантажуємо файл*

return np.*loadtxt*(*file\_name*)

def plot\_data(*t*, *y*, *title*):

*#значить малюємо саме графік*

plt.*plot*(*t*, *y*)

plt.*title*(*title*)

plt.*grid*(True)

plt.*show*()

def dft(*y*, *N*):

*# значить саме дискретне перетворення фур'є*

fourier\_func = np.*zeros*(*N*, *dtype*=*complex*)

for k in *range*(*N*):

for j in *range*(*N*):

fourier\_func[k] += (*y*[j] / *N*) \* np.*exp*(-1j \* 2 \* np.pi \* k \* j / *N*)

return fourier\_func

def fft(*y*, *N*):

*# відповідно швидке перетворення фур'є*

return np.fft.*fft*(*y*) / *N*

def plot\_half\_fft(*fft\_result*, *N*, *df*):

*# відповідно обчислюємо і виводимо першу половину перетворення*

half\_N = *N* // 2

freqs = np.*arange*(half\_N) \* *df # значить частота для першої половини*

fft\_half\_abs = np.*abs*(*fft\_result*[:half\_N]) *# і саме амплітуди для неї ж*

plt.*plot*(freqs, fft\_half\_abs)

plt.*title*("Перша половина FFT")

plt.*grid*(True)

plt.*xlabel*("Частота (Hz)")

plt.*ylabel*("Амплітуда")

plt.*show*()

def find\_peaks(*fourier\_func\_abs*, *df*, *N*):

*# значить відповідно "піки" шукаємо*

peaks = []

for j in *range*(2, *round*(*N* / 2) - 1):

if (*fourier\_func\_abs*[j] > *fourier\_func\_abs*[j - 1] and

*fourier\_func\_abs*[j] > *fourier\_func\_abs*[j + 1] and

*abs*(*fourier\_func\_abs*[j] - *fourier\_func\_abs*[j - 1]) > 1):

peaks.*append*(j \* *df*)

return peaks

def create\_matrix(*t*, *sin\_f*, *N*):

*# відповідно створення матриці*

A = np.*zeros*((5, 5))

A[0, 0] = np.*sum*(*t* \*\* 6) *# t^6*

A[0, 1] = np.*sum*(*t* \*\* 5) *# t^5*

A[0, 2] = np.*sum*(*t* \*\* 4) *# t^4*

A[0, 3] = np.*sum*(*sin\_f* \* *t* \*\* 3) *# sin\_f \* t^3*

A[0, 4] = np.*sum*(*t* \*\* 3) *# t^3*

A[1, 0] = np.*sum*(*t* \*\* 5) *# t^5*

A[1, 1] = np.*sum*(*t* \*\* 4) *# t^4*

A[1, 2] = np.*sum*(*t* \*\* 3) *# t^3*

A[1, 3] = np.*sum*(*sin\_f* \* *t* \*\* 2) *# sin\_f \* t^2*

A[1, 4] = np.*sum*(*t* \*\* 2) *# t^2*

A[2, 0] = np.*sum*(*t* \*\* 4) *# t^4*

A[2, 1] = np.*sum*(*t* \*\* 3) *# t^3*

A[2, 2] = np.*sum*(*t* \*\* 2) *# t^2*

A[2, 3] = np.*sum*(*sin\_f* \* *t*) *# sin\_f \* t*

A[2, 4] = np.*sum*(*t*) *# t*

A[3, 0] = np.*sum*(*sin\_f* \* *t* \*\* 3) *# sin\_f \* t^3*

A[3, 1] = np.*sum*(*sin\_f* \* *t* \*\* 2) *# sin\_f \* t^2*

A[3, 2] = np.*sum*(*sin\_f* \* *t*) *# sin\_f \* t*

A[3, 3] = np.*sum*(*sin\_f* \*\* 2) *# sin\_f^2*

A[3, 4] = np.*sum*(*sin\_f*) *# sin\_f*

A[4, 0] = np.*sum*(*t* \*\* 3) *# t^3*

A[4, 1] = np.*sum*(*t* \*\* 2) *# t^2*

A[4, 2] = np.*sum*(*t*) *# t*

A[4, 3] = np.*sum*(*sin\_f*) *# sin\_f*

A[4, 4] = *N*

return A

*#[t^6 t^5 t^4 sin(f)\*t^3 t^3}*

*#[t^5 t^4 t^3 sin(f)\*t^2 t^2}*

*#[t^4 t^3 t^2 sin(f)\*t t }*

*#[sin(f)\*t^3 sin(f)\*t^2 sin(f)\*t sin^2(f) sin(f)}*

*#[t^3 t^2 t sin(f) N }*

def create\_vector(*y*, *t*, *sin\_f*):

*# значить вектор для саме апроксимації*

c = np.*array*([np.*sum*(*y* \* *t* \*\* 3), np.*sum*(*y* \* *t* \*\* 2), np.*sum*(*y* \* *t*), np.*sum*(*y* \* *sin\_f*), np.*sum*(*y*)])

return c

def approx\_func(*a*, *t*, *sin\_f*):

*# відповідно обчислення функції*

return *a*[0] \* *t* \*\* 3 + *a*[1] \* *t* \*\* 2 + *a*[2] \* *t* + *a*[3] \* *sin\_f* + *a*[4]

def main():

*# значить початкові саме параметри з умови відповідно*

dt = 0.01

T = 5

t = np.*arange*(0, T + dt, dt)

df = 1 / T

*# відповідно закачка даних*

y = *load\_data*('f6.txt')

N = *len*(y)

*# саме побудова початкового графіку*

*plot\_data*(t, y, "Початкові значення")

*# значить рахуємо і будуємо саме ДПФ*

fourier\_func = *dft*(y, N)

*plot\_data*(np.*arange*(N), np.*abs*(fourier\_func), "Дискретне перетворення Фур’є (DFT)")

*# відповідно те саме з ШПФ*

fft\_result = *fft*(y, N)

*plot\_data*(np.*arange*(N), np.*abs*(fft\_result), "Швидке перетворення Фур’є (FFT)")

*#значить сама половина ШПФ, т.я. вони співпадають з ДПФ*

*plot\_half\_fft*(fft\_result, N, df)

*# значить шукаємо екстремуми*

fourier\_func\_abs = np.*abs*(fourier\_func)

peaks = *find\_peaks*(fourier\_func\_abs, df, N)

*# і відповідно якщо знайшли екстремуми*

if peaks:

sin\_f = np.*sin*(2 \* np.pi \* peaks[0] \* t)

else:

sin\_f = np.*zeros\_like*(t)

*# значить робимо матрицю і вектор для апроксимації*

A = *create\_matrix*(t, sin\_f, N)

c = *create\_vector*(y, t, sin\_f)

*# відповідно розв'язуємо рівняння через нампай*

a = np.linalg.*solve*(A, c)

*# значить виводимо коефіцієнти в консоль*

*print*(f"Коефіцієнти: {a}")

*# відповідно побудова та графік апроксимуючої функції*

approx\_y = *approx\_func*(a, t, sin\_f)

*plot\_data*(t, approx\_y, "Апроксимуюча функція")

*# і значить саме похибка в консоль*

error\_value = np.*sum*((approx\_y - y) \*\* 2)

*print*(f"Похибка: {error\_value}")

if \_\_name\_\_ == '\_\_main\_\_':

*main*()