Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Лабораторна робота № 3

З курсу «Моделювання складних систем»

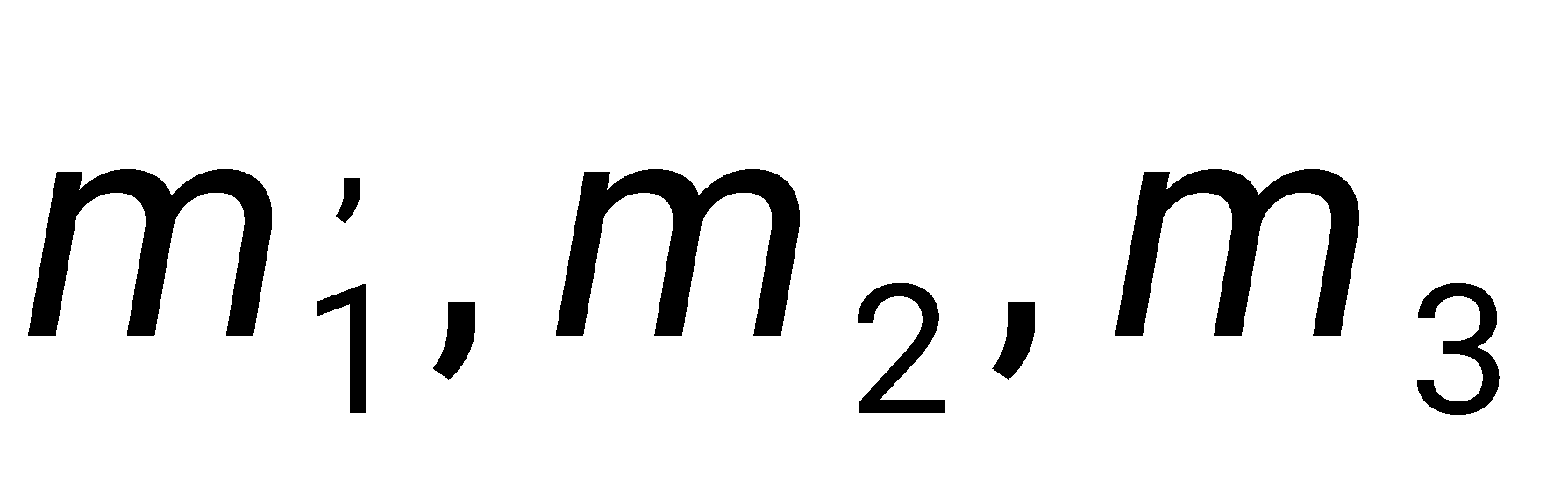
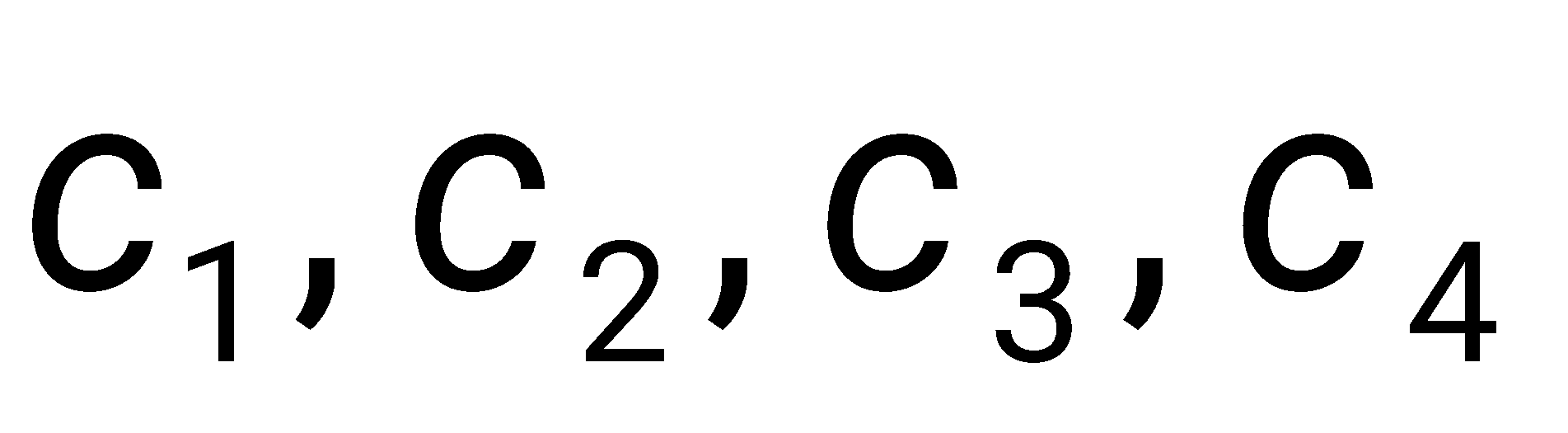
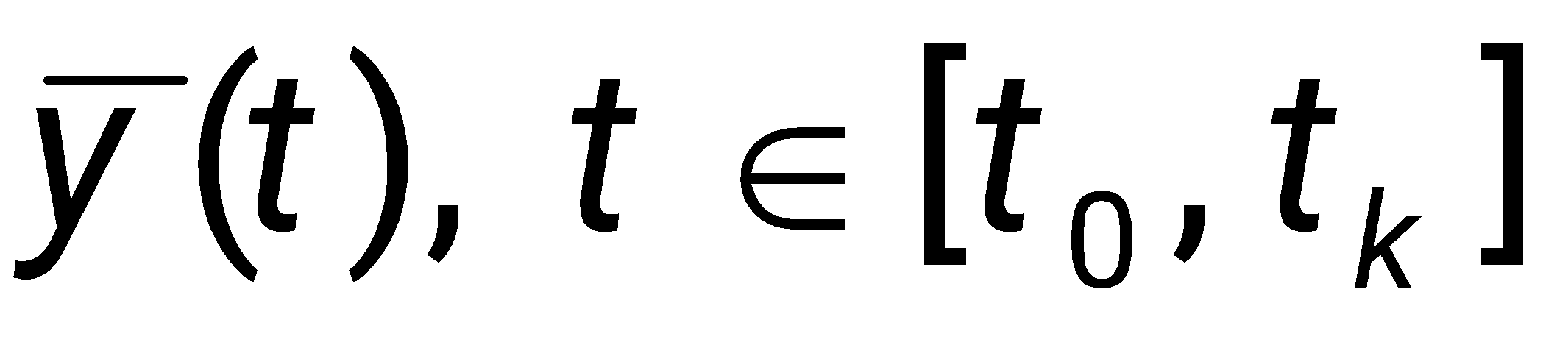
Варінт № 6

Виконав

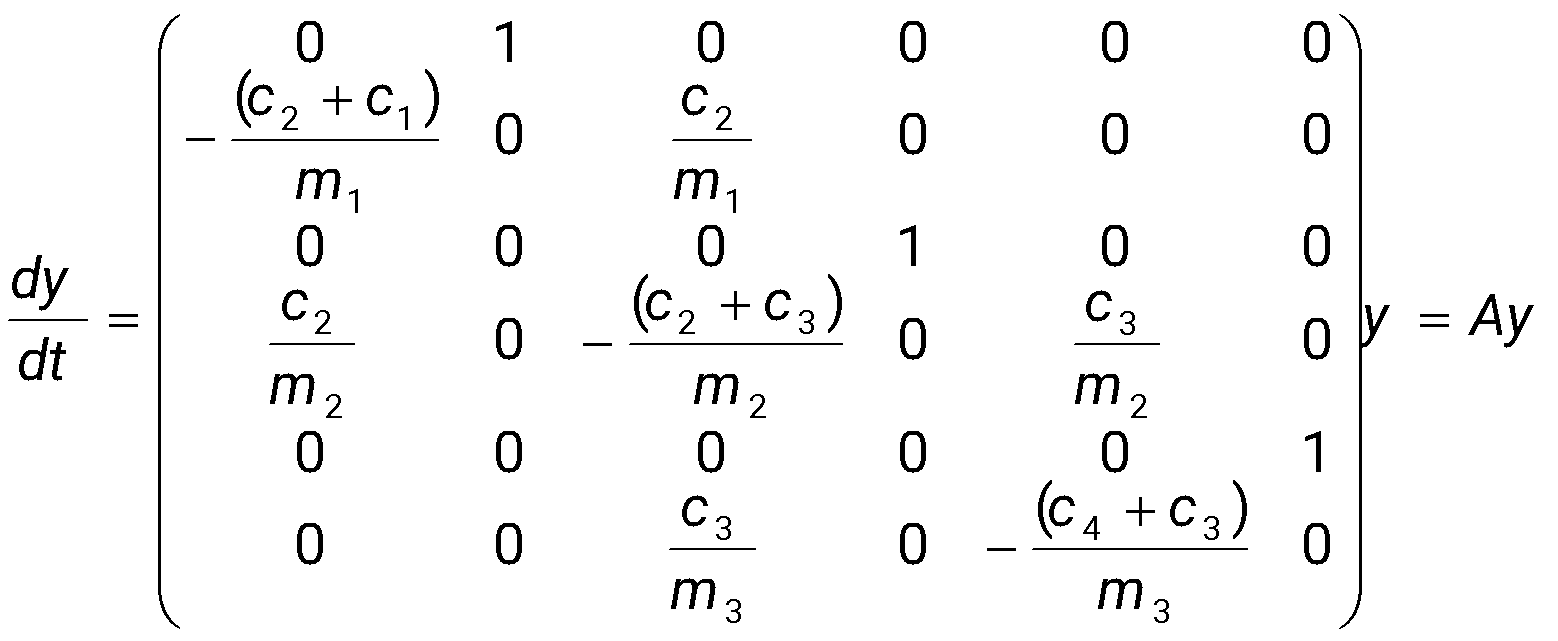
Студент групи ІПС-31

Дубина Андрій

**Постановка задачі**

Для математичної моделі коливання трьох мас , які поєднані між собою пружинами з відповідними жорсткостями , і відомої функції спостереження координат моделі  потрібно оцінити частину невідомих параметрів моделі з використанням функції чутливості. Для чисельного інтегрування необхідно використовувати метод Рунге-Кутта 4-го порядку.

Ця модель описується наступною системою:



**Хід роботи**

Ініціалізуємо параметри, надані варіантом та вказуємо точність. У варіанті 6 маємо: , початкове наближення , відомі параметри , .

*# Зчитуємо спостереження з файлу та перетворюємо їх у масив numpy*

with *open*('y6.txt') as file:

observations = np.*array*([line.*split*() for line in file.*readlines*()],

*float*).T

*# Відомі параметри*

c3 = 0.2

c4 = 0.12

m1 = 12

m3 = 18

*# Початкове наближення*

initial\_guess = np.*array*([0.1, 0.15, 19]) *# c1, c2, m2*

start\_time = 0 *# початок інтервалу*

end\_time = 50 *# кінець інтервалу*

time\_step = 0.2 *# крок*

tolerance = 1e-6 *# параметр точності*

Задаємо та обчислюємо необхідні матриці – матриця чутливості та похідні параметрів:

*# Матриця чутливості*

def computeSensitivityMatrix(*parameters*):

c1, c2, m2 = *parameters*

return np.*array*([

[0, 1, 0, 0, 0, 0],

[-(c1 + c2) / m1, 0, c2 / m1, 0, 0, 0],

[0, 0, 0, 1, 0, 0],

[c2 / m2, 0, -(c2 + c3) / m2, 0, c3 / m2, 0],

[0, 0, 0, 0, 0, 1],

[0, 0, c3 / m3, 0, -(c4 + c3) / m3, 0]

])

*# Похідні параметрів*

def computeParameterDerivatives(*states*, *parameters*):

c1, c2, m2 = *parameters*

*# Матриця похідних по c1*

derivatives\_param0 = np.*array*([

[0, 0, 0, 0, 0, 0],

[-1 / m1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]

])

*# Матриця похідних по c2*

derivatives\_param1 = np.*array*([

[0, 0, 0, 0, 0, 0],

[-1 / m1, 0, 1 / m1, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[1 / m2, 0, -1 / m2, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]

])

*# Матриця похідних по m2*

derivatives\_param2 = np.*array*([

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[-c2 / (m2 \*\* 2), 0, (c2 + c3) / (m2 \*\* 2), 0, -c3 / (m2 \*\* 2), 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]

])

*# Обчислення матриць похідних для поточного стану*

derivatives\_param0 = derivatives\_param0 @ *states*

derivatives\_param1 = derivatives\_param1 @ *states*

derivatives\_param2 = derivatives\_param2 @ *states*

return np.*array*([derivatives\_param0, derivatives\_param1,

derivatives\_param2]).T

Розв’язуємо задану систему методом Рунге-Кутта при . Знаходимо :

*# Модельні стани*

model\_states = np.*zeros\_like*(observations)

model\_states[0] = observations[0].*copy*()

*# Інтегруємо модель за методом Рунге-Кутта; знаходимо стани моделі у кожен часовий відрізок*

for i in *range*(1, *len*(time\_points)):

previous\_state = model\_states[i - 1]

k1 = *time\_step* \* *computeModelDynamics*(previous\_state, *initial\_parameters*)

k2 = *time\_step* \* *computeModelDynamics*(previous\_state + k1 / 2, *initial\_parameters*)

k3 = *time\_step* \* *computeModelDynamics*(previous\_state + k2 / 2, *initial\_parameters*)

k4 = *time\_step* \* *computeModelDynamics*(previous\_state + k3, *initial\_parameters*)

current\_state = previous\_state + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6

model\_states[i] = current\_state

Підставляємо у рівняння чутливості та знову використовуємо метод Рунге-Кутта для знаходження його розв’язку :

*# Інтегруємо функції чутливості*

sensitivity\_matrix = np.*zeros*((*len*(time\_points), 6, 3))

parameter\_derivatives = *computeParameterDerivatives*(model\_states.T, *initial\_parameters*)

sensitivity\_coefficients = *computeSensitivityMatrix*(*initial\_parameters*)

for i in *range*(1, *len*(time\_points)):

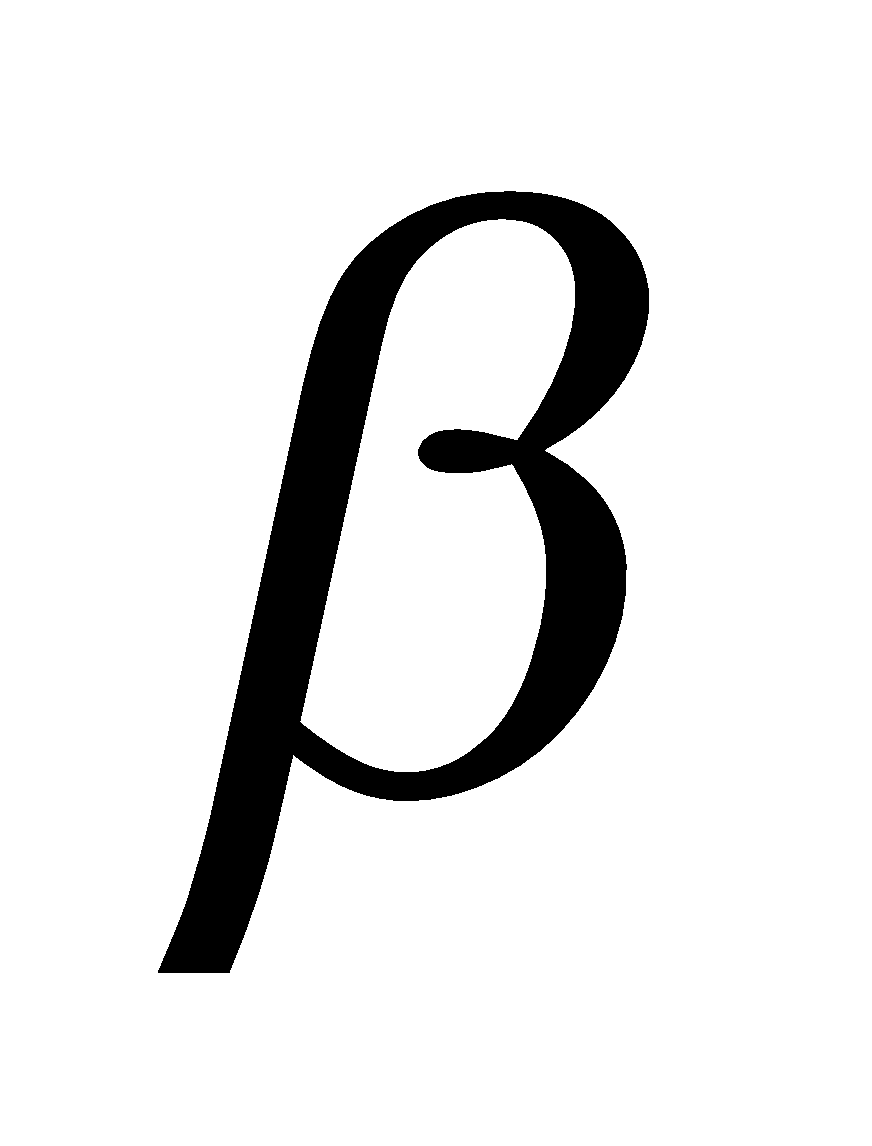
k1 = *time\_step* \* (sensitivity\_coefficients @ sensitivity\_matrix[i - 1] + parameter\_derivatives[i - 1])

k2 = *time\_step* \* (sensitivity\_coefficients @ (sensitivity\_matrix[i - 1] + k1 / 2) + parameter\_derivatives[i - 1])

k3 = *time\_step* \* (sensitivity\_coefficients @ (sensitivity\_matrix[i - 1] + k2 / 2) + parameter\_derivatives[i - 1])

k4 = *time\_step* \* (sensitivity\_coefficients @ (sensitivity\_matrix[i - 1] + k3) + parameter\_derivatives[i - 1])

sensitivity\_matrix[i] = sensitivity\_matrix[i - 1] + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6

Знаходимо. Також знаходимо показник якості ідентифікації параметрів невідомих параметрів :

.

*# Оновлення параметрів*

sensitivity\_derivatives = (np.*array*([u.T @ u for u in sensitivity\_matrix]) \* *time\_step*).*sum*(0)

sensitivity\_derivatives = np.linalg.*inv*(sensitivity\_derivatives)

state\_difference = (observations - model\_states)

sensitivity\_observations = (np.*array*(

[sensitivity\_matrix[i].T @ state\_difference[i] for i in *range*(*len*(time\_points))]) \* *time\_step*).*sum*(0)

quality\_pointer = (np.*array*(

[state\_difference[i].T @ state\_difference[i] for i in *range*(*len*(time\_points))]) \* *time\_step*).*sum*(0)

parameter\_update = sensitivity\_derivatives @ sensitivity\_observations

Обчислюємо , та перевіряємо умову зупинки: . Якщо умова виконується, то ми знайшли ідентифікували невідомі параметри і робота програми на цьому закінчується. Інакше, переходимо на наступну ітерацію.

*initial\_parameters* += parameter\_update

if np.*abs*((quality\_pointer)).*max*() < *tolerance*:

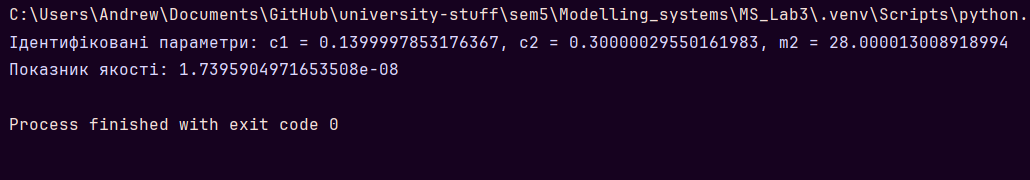
return *initial\_parameters*, quality\_pointer

if np.*abs*((parameter\_update)).*max*() < *tolerance*:

return *initial\_parameters*, quality\_pointer

**Результат роботи програми**

Для варіанту 6, а тобто, вектор оцінюваних параметрів , початкове наближення , відомі параметри , , ім’я файлу з спостережуваними даними y6.txt, маємо наступні результати виконання:



**Повний код програми**

import numpy as np

*# Матриця чутливості*

def computeSensitivityMatrix(*parameters*):

c1, c2, m2 = *parameters*

return np.*array*([

[0, 1, 0, 0, 0, 0],

[-(c1 + c2) / m1, 0, c2 / m1, 0, 0, 0],

[0, 0, 0, 1, 0, 0],

[c2 / m2, 0, -(c2 + c3) / m2, 0, c3 / m2, 0],

[0, 0, 0, 0, 0, 1],

[0, 0, c3 / m3, 0, -(c4 + c3) / m3, 0]

])

*# Похідні параметрів*

def computeParameterDerivatives(*states*, *parameters*):

c1, c2, m2 = *parameters*

*# Матриця похідних по c1*

derivatives\_param0 = np.*array*([

[0, 0, 0, 0, 0, 0],

[-1 / m1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]

])

*# Матриця похідних по c2*

derivatives\_param1 = np.*array*([

[0, 0, 0, 0, 0, 0],

[-1 / m1, 0, 1 / m1, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[1 / m2, 0, -1 / m2, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]

])

*# Матриця похідних по m2*

derivatives\_param2 = np.*array*([

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0],

[-c2 / (m2 \*\* 2), 0, (c2 + c3) / (m2 \*\* 2), 0, -c3 / (m2 \*\* 2), 0],

[0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0]

])

*# Обчислення матриць похідних для поточного стану*

derivatives\_param0 = derivatives\_param0 @ *states*

derivatives\_param1 = derivatives\_param1 @ *states*

derivatives\_param2 = derivatives\_param2 @ *states*

return np.*array*([derivatives\_param0, derivatives\_param1, derivatives\_param2]).T

*# Динаміка моделі*

def computeModelDynamics(*states*, *parameters*):

return *computeSensitivityMatrix*(*parameters*) @ *states*

*# Оптимізація параметрів*

def optimizeParameters(*initial\_parameters*, *start\_time*, *end\_time*, *time\_step*, *tolerance*):

time\_points = np.*linspace*(*start\_time*, *end\_time*, *int*((*end\_time* - *start\_time*) / *time\_step* + 1))

while True:

*# Модельні стани*

model\_states = np.*zeros\_like*(observations)

model\_states[0] = observations[0].*copy*()

*# Інтегруємо модель за методом Рунге-Кутта; знаходимо стани моделі у кожен часовий відрізок*

for i in *range*(1, *len*(time\_points)):

previous\_state = model\_states[i - 1]

k1 = *time\_step* \* *computeModelDynamics*(previous\_state, *initial\_parameters*)

k2 = *time\_step* \* *computeModelDynamics*(previous\_state + k1 / 2, *initial\_parameters*)

k3 = *time\_step* \* *computeModelDynamics*(previous\_state + k2 / 2, *initial\_parameters*)

k4 = *time\_step* \* *computeModelDynamics*(previous\_state + k3, *initial\_parameters*)

current\_state = previous\_state + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6

model\_states[i] = current\_state

*# Інтегруємо функції чутливості*

sensitivity\_matrix = np.*zeros*((*len*(time\_points), 6, 3))

parameter\_derivatives = *computeParameterDerivatives*(model\_states.T, *initial\_parameters*)

sensitivity\_coefficients = *computeSensitivityMatrix*(*initial\_parameters*)

for i in *range*(1, *len*(time\_points)):

k1 = *time\_step* \* (sensitivity\_coefficients @ sensitivity\_matrix[i - 1] + parameter\_derivatives[i - 1])

k2 = *time\_step* \* (sensitivity\_coefficients @ (sensitivity\_matrix[i - 1] + k1 / 2) + parameter\_derivatives[i - 1])

k3 = *time\_step* \* (sensitivity\_coefficients @ (sensitivity\_matrix[i - 1] + k2 / 2) + parameter\_derivatives[i - 1])

k4 = *time\_step* \* (sensitivity\_coefficients @ (sensitivity\_matrix[i - 1] + k3) + parameter\_derivatives[i - 1])

sensitivity\_matrix[i] = sensitivity\_matrix[i - 1] + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6

*# Оновлення параметрів*

sensitivity\_derivatives = (np.*array*([u.T @ u for u in sensitivity\_matrix]) \* *time\_step*).*sum*(0)

sensitivity\_derivatives = np.linalg.*inv*(sensitivity\_derivatives)

state\_difference = (observations - model\_states)

sensitivity\_observations = (np.*array*(

[sensitivity\_matrix[i].T @ state\_difference[i] for i in *range*(*len*(time\_points))]) \* *time\_step*).*sum*(0)

quality\_pointer = (np.*array*(

[state\_difference[i].T @ state\_difference[i] for i in *range*(*len*(time\_points))]) \* *time\_step*).*sum*(0)

parameter\_update = sensitivity\_derivatives @ sensitivity\_observations

*initial\_parameters* += parameter\_update

if np.*abs*((quality\_pointer)).*max*() < *tolerance*:

return *initial\_parameters*, quality\_pointer

if np.*abs*((parameter\_update)).*max*() < *tolerance*:

return *initial\_parameters*, quality\_pointer

if \_\_name\_\_ == "\_\_main\_\_":

*# Зчитуємо спостереження з файлу та перетворюємо їх у масив numpy*

with *open*('y6.txt') as file:

observations = np.*array*([line.*split*() for line in file.*readlines*()], *float*).T

*# Відомі параметри*

c3 = 0.2

c4 = 0.12

m1 = 12

m3 = 18

*# Початкове наближення*

initial\_guess = np.*array*([0.1, 0.15, 19]) *# c1, c2, m2*

start\_time = 0 *# початок інтервалу*

end\_time = 50 *# кінець інтервалу*

time\_step = 0.2 *# крок*

tolerance = 1e-6 *# параметр точності*

solution, quality\_pointer = *optimizeParameters*(initial\_guess, start\_time, end\_time, time\_step, tolerance)

*print*(f"Ідентифіковані параметри: c1 = {solution[0]}, c2 = {solution[1]}, m2 = {solution[2]}")

*print*(f"Показник якості: {quality\_pointer}")