Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Лабораторна робота №3

З «Моделювання складних систем»

Виконала студентка 3-го курсу
Групи ІПС-31
Величко Діана Сергіївна

Варіант 3

2024

Завдання:

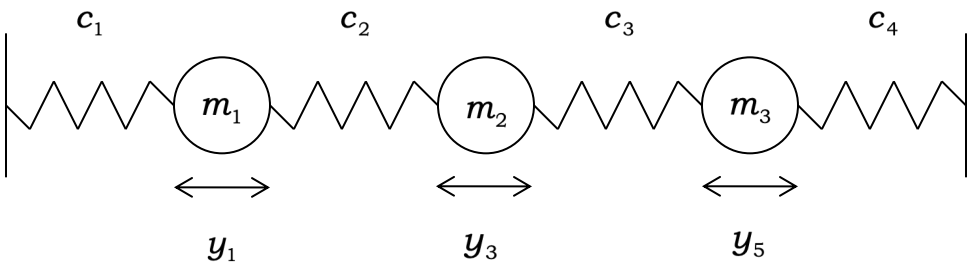
Для математичної моделі коливання трьох мас m_1, m_2, m_3, які поєднані між собою пружинами з відповідними жорсткостями c_1, c_2, c_3, c_4, і відомої функції спостереження координат моделі $\bar{y}(t)$, $t \in [t_0, t_k]$ потрібно оцінити частину невідомих параметрів моделі з використанням функції чутливості.

Вектор оцінюваних параметрів β = (c1,c3,m1)T, початкове наближення β0 = (0.1,0.1, 9)T, відомі параметри c2 = 0.3, c4 = 0.12, m2 = 28, m3 = 18, ім'я файлу з спостережуваними даними y3.txt.

Інтервал часу на якому проведені спостереження стану моделі – $t_0 = 0$, .$t_k = 50$, $\Delta t = 0.2$.

Теорія:

Математична модель коливання 3-х мас



Її можна описати наступною системою :

$$\frac{dy}{dt} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{(c_2+c_1)}{m_1} & 0 & \frac{c_2}{m_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{c_2}{m_2} & 0 & -\frac{(c_2+c_3)}{m_2} & 0 & \frac{c_3}{m_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{c_3}{m_3} & 0 & -\frac{(c_4+c_3)}{m_3} & 0 \end{pmatrix} y = Ay$$

Алгоритм :

Додаємо файл та ініціалізуємо зчитування даних з нього .

```
# Завантаження даних з файлу
data = np.loadtxt('y3.txt').T
```

Додаємо відомі за умовою параметри маси та жорсткостей , ініціалізуємо : початковий час, кінцевий час , крок по часу та необхідну точність для завершення ітерацій .

```
c2, c4, m2, m3 = 0.3, 0.12, 28, 18
t0, T, deltaT = 0, 50, 0.
epsilon = 1e-5
```

Вносимо (відоме за умовою) початкове наближення β0 = (0.1, 0.1, 9)T

```
c1, c3, m1 = 0.1, 0.1, 9
```

Створюємо функцію для розрахунку чутливості матриці

```
def SensMatrix(b):

    m1_inv, m3_inv, b2_inv = 1 / m1, 1 / m3, 1 / b[2] if b[2] != 0 else 0
    return np.array([
        [0, 1, 0, 0, 0, 0],
        [- (b[1] + b[0]) * m1_inv, 0, b[1] * m1_inv, 0, 0, 0],
        [0, 0, 0, 1, 0, 0],
        [b[1] * b2_inv, 0, -(b[1] + c3) * b2_inv, 0, c3 * b2_inv, 0],
        [0, 0, 0, 0, 0, 1],
        [0, 0, c3 * m3_inv, 0, -(c4 + c3) * m3_inv, 0]
    ])
```

Створюємо функцію для розрахунку похідних моделі відносно параметрів

```
def ModelDerivatives(y, b):
    #  матриці для обчислення похідних по кожному параметру (db0, db1, db2)
    db0 = np.zeros((6, 6))
    db1 = np.zeros((6, 6))
    db2 = np.zeros((6, 6))
```

Заповнюємо значення для кожної матриці

```
db0[1, 0] = -1 / m1
db1[1, 0] = -1 / m1
db1[1, 2] = 1 / m1
db2[3, 0] = -b[1] / (b[2] ** 2)
db2[3, 2] = (b[1] + c3) / (b[2] ** 2)
db2[3, 4] = -c3 / (b[2] ** 2)
```

Домножуємо на y , щоб отримати транспоновану

```
db0 = np.dot(db0, y)
db1 = np.dot(db1, y)
db2 = np.dot(db2, y)

return np.array([db0, db1, db2]).T
```

Створюємо функцію для обчислення чутливості за допомогою методу Рунге-Кутта

```
def Sensitivity_RK(A, db, uu, deltaT, timeStamps):
    for i in range(1, len(timeStamps)):
        k1 = deltaT * (np.dot(A, uu[i - 1]) + db[i - 1])
        k2 = deltaT * (np.dot(A, (uu[i - 1] + k1 / 2)) + db[i - 1])
        k3 = deltaT * (np.dot(A, (uu[i - 1] + k2 / 2)) + db[i - 1])
        k4 = deltaT * (np.dot(A, (uu[i - 1] + k3)) + db[i - 1])

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

    return uu
```

Створюємо функцію моделювання за допомогою методу Рунге-Кутта

```
def Model_RK(b, timeStamps, deltaT):
    #ініціалізуємо результати
    yy = np.zeros_like(data)
    yy[0] = data[0].copy()
    A = SensMatrix(b)  # Обчислення матриці чутливості для заданих параметрів
b

    # розв'язування рівнянь по методу Рунге-Куттf
    for i in range(1, len(timeStamps)):
        y_prev = yy[i - 1]
        k1 = deltaT * np.dot(A, y_prev)
        k2 = deltaT * np.dot(A, (y_prev + k1 / 2))
        k3 = deltaT * np.dot(A, (y_prev + k2 / 2))
        k4 = deltaT * np.dot(A, (y_prev + k3))
        yy[i] = y_prev + (k1 + 2 * k2 + 2 * k3 + k4) / 6
    return yy
```

Створюєму функцію для розрахунку deltaB (зміщень параметрів), та визначаємо їх

```
# Функція для розрахунку зміщень параметрів (deltaB)
def DeltaB(uu, db, deltaT, timeStamps, data, b):
    # Різниця між реальними даними та модельним прогнозом
    diff_y = data - Model_RK(b, timeStamps, deltaT)

    # Визначення зміщень параметрів
    du = (np.array([u.T @ u for u in uu]) * deltaT).sum(0)
    du_inv = np.linalg.inv(du) # Обернена матриця
    uY = (np.array([uu[i].T @ diff_y[i] for i in range(len(timeStamps))]) *
deltaT).sum(0)
    deltaB = du_inv @ uY # Обчислення зміщення

    return deltaB
```

Створюємо функцію та обчислюємо середньоквадратичну похибку MSE

```
def MSE(data, model):
    return np.mean((data - model) ** 2)
```

Створюємо функцію для пошуку параметрів

```
def Parameters(b, t0, T, deltaT, eps, max_iter=100):
    # Створення масиву часових міток
    timeStamps = np.linspace(t0, T, int((T - t0) / deltaT + 1))

    iteration_count = 0
    prev_b = b.copy()

    iteration_results = []  # Список для збереження результатів ітерацій
    iteration_times = []  # Зберігання часу виконання кожної ітерації
```

Проводимо пошук параметрів . Ініціалізуємо деякі значення .

```
while iteration_count < max_iter:
    iteration_count += 1

    start_iter_time = time.time()  # Початок часу для ітерації
    yy = Model_RK(b, timeStamps, deltaT)
    uu = np.zeros((len(timeStamps), 6, 3))
```

```
    db = ModelDerivatives(yy.T, b)
    A = SensMatrix(b)
    uu = Sensitivity_RK(A, db, uu, deltaT, timeStamps)
    deltaB = DeltaB(uu, db, deltaT, timeStamps, data, b)

    b += deltaB
    mse = MSE(data, Model_RK(b, timeStamps, deltaT))

    iteration_results.append((iteration_count, b.copy(), mse))

    end_iter_time = time.time()  # Кінець часу для ітерації
    iteration_times.append(end_iter_time - start_iter_time)

    if np.abs(deltaB).max() < eps:
        break

return b, iteration_count, iteration_results, iteration_times
```

Створюємо графіки для аналізу отриманих даних

```
# Графік : Зміни параметрів (c1, c3, m1) за ітераціями
iterations = [iteration for iteration, _, _ in iteration_results]
c1_values = [params[0] for _, params, _ in iteration_results]
c3_values = [params[1] for _, params, _ in iteration_results]
m1_values = [params[2] for _, params, _ in iteration_results]

plt.figure(figsize=(10, 6))
plt.plot(iterations, c1_values, label='c1')
plt.plot(iterations, c3_values, label='c3')
plt.plot(iterations, m1_values, label='m1')
plt.xlabel('Ітерація')
plt.ylabel('Значення параметра')
plt.title('Зміни параметрів за ітераціями')
plt.legend()
plt.grid(True)
plt.show()
```

```
# Графік : MSE за ітераціями
mse_values = [mse for _, _, mse in iteration_results]

plt.figure(figsize=(10, 6))
plt.plot(iterations, mse_values, label='MSE', color='violet')
plt.xlabel('Ітерація')
plt.ylabel('MSE')
plt.title('Показник якості за ітераціями')
plt.grid(True)
plt.show()
```

```
# Графік : Зміна маси (m1) за ітераціями
plt.figure(figsize=(10, 6))
plt.plot(iterations, m1_values, label='m1', color='green')
plt.xlabel('Ітерація')
plt.ylabel('m1')
plt.title('Зміна маси m1 за ітераціями')
plt.grid(True)
plt.show()
```

```python
# Графік : Зміна жорсткості (c1, c3) за ітераціями
plt.figure(figsize=(10, 6))
plt.plot(iterations, c1_values, label='c1', color='blue')
plt.plot(iterations, c3_values, label='c3', color='orange')
plt.xlabel('Ітерація')
plt.ylabel('Значення жорсткості')
plt.title('Зміна жорсткості за ітераціями')
plt.legend()
plt.grid(True)
plt.show()
```

```python
# Графік : Зміна часу виконання за ітераціями
plt.figure(figsize=(10, 6))
plt.plot(iterations, iteration_times, label='Час виконання', color='red')
plt.xlabel('Ітерація')
plt.ylabel('Час (с)')
plt.title('Час виконання за ітераціями')
plt.grid(True)
plt.show()
```

Результати та аналіз :

Після запуску програми ми отримуємо такі результати :

Ітерація	Невідомі параметри	Показник якості
1	c1 = 0.257773, c3 = 0.113246, m1 = 9.909517	0.07005225
2	c1 = 0.277117, c3 = 0.099853, m1 = 12.354133	0.05343638
3	c1 = 0.269394, c3 = 0.116419, m1 = 13.643670	0.04667218
4	c1 = 0.258127, c3 = 0.124671, m1 = 14.711510	0.04352785
5	c1 = 0.251029, c3 = 0.131874, m1 = 15.357036	0.04213627
6	c1 = 0.246165, c3 = 0.135667, m1 = 15.790782	0.04151408
7	c1 = 0.242766, c3 = 0.138675, m1 = 16.034317	0.04117389
8	c1 = 0.240806, c3 = 0.140123, m1 = 16.198339	0.04101925
9	c1 = 0.239414, c3 = 0.141335, m1 = 16.286347	0.04091245
10	c1 = 0.238687, c3 = 0.141838, m1 = 16.349179	0.04086818
11	c1 = 0.238130, c3 = 0.142337, m1 = 16.380304	0.04082836
12	c1 = 0.237876, c3 = 0.142492, m1 = 16.405041	0.04081577
13	c1 = 0.237649, c3 = 0.142707, m1 = 16.415589	0.04079918
14	c1 = 0.237567, c3 = 0.142743, m1 = 16.425696	0.04079639
15	c1 = 0.237471, c3 = 0.142841, m1 = 16.428941	0.04078885
16	c1 = 0.237449, c3 = 0.142840, m1 = 16.433283	0.04078896
17	c1 = 0.237406, c3 = 0.142889, m1 = 16.434043	0.04078523
18	c1 = 0.237403, c3 = 0.142879, m1 = 16.436033	0.04078601
19	c1 = 0.237383, c3 = 0.142905, m1 = 16.436024	0.04078403
20	c1 = 0.237385, c3 = 0.142895, m1 = 16.437005	0.04078479
21	c1 = 0.237375, c3 = 0.142909, m1 = 16.436811	0.04078367
22	c1 = 0.237378, c3 = 0.142902, m1 = 16.437332	0.04078426
23	c1 = 0.237372, c3 = 0.142910, m1 = 16.437135	0.04078360
24	c1 = 0.237375, c3 = 0.142905, m1 = 16.437430	0.04078402
25	c1 = 0.237372, c3 = 0.142910, m1 = 16.437276	0.04078361

26	c1 = 0.237374, c3 = 0.142907, m1 = 16.437451	0.04078389
27	c1 = 0.237372, c3 = 0.142910, m1 = 16.437341	0.04078364
28	c1 = 0.237373, c3 = 0.142907, m1 = 16.437449	0.04078383
29	c1 = 0.237372, c3 = 0.142910, m1 = 16.437374	0.04078366
30	c1 = 0.237373, c3 = 0.142908, m1 = 16.437441	0.04078379
31	c1 = 0.237372, c3 = 0.142909, m1 = 16.437391	0.04078368
32	c1 = 0.237372, c3 = 0.142908, m1 = 16.437434	0.04078377
33	c1 = 0.237372, c3 = 0.142909, m1 = 16.437401	0.04078370
34	c1 = 0.237372, c3 = 0.142908, m1 = 16.437429	0.04078375
35	c1 = 0.237372, c3 = 0.142909, m1 = 16.437407	0.04078371
36	c1 = 0.237372, c3 = 0.142909, m1 = 16.437425	0.04078374
37	c1 = 0.237372, c3 = 0.142909, m1 = 16.437411	0.04078372
38	c1 = 0.237372, c3 = 0.142909, m1 = 16.437422	0.04078374
39	c1 = 0.237372, c3 = 0.142909, m1 = 16.437413	0.04078372

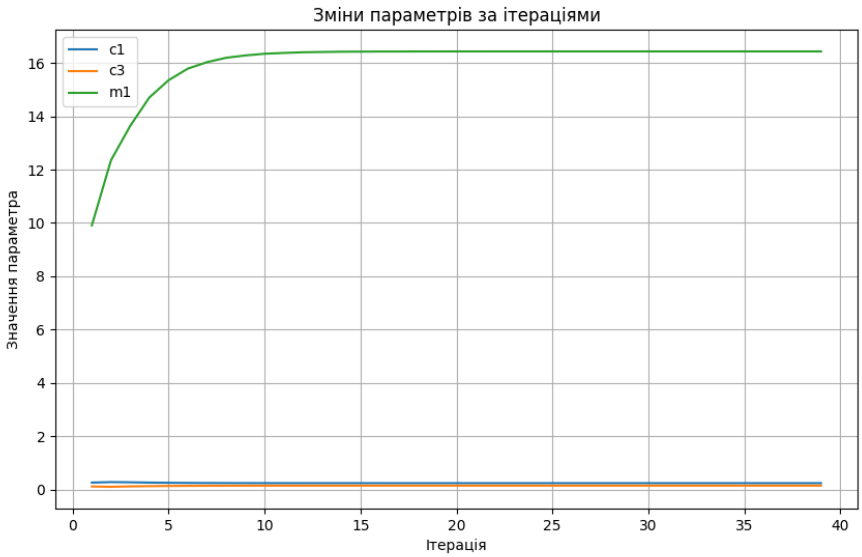
Знайдені параметри: c1 = 0.237400, c3 = 0.142900, m1 = 16.437400
Показник якості: Q = 0.04078444
Час виконання: 0.639274 секунд

Дана таблиця зображає такі результати : № ітерації , отримані результати та показник якості на конкретному етапі .

Отже ми бачимо , що необхідні нам результати , а саме c1 = 0.237400, c3 = 0.142900, m1 = 16.437400 були отримані на 39 ітерації з показником якості що \approx 0.478

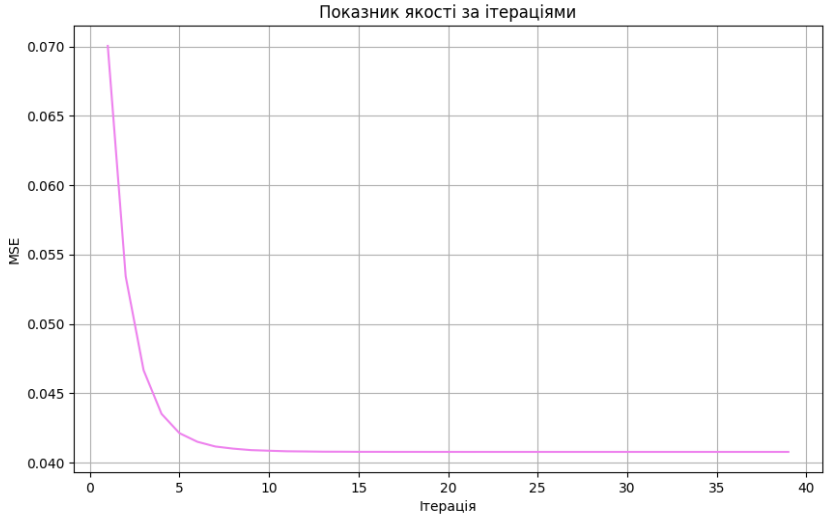
Також , в результаті роти програми ми отримали графіки , проаналізуємо :

Графік змін параметрів (c1, c3, m1) за ітераціями



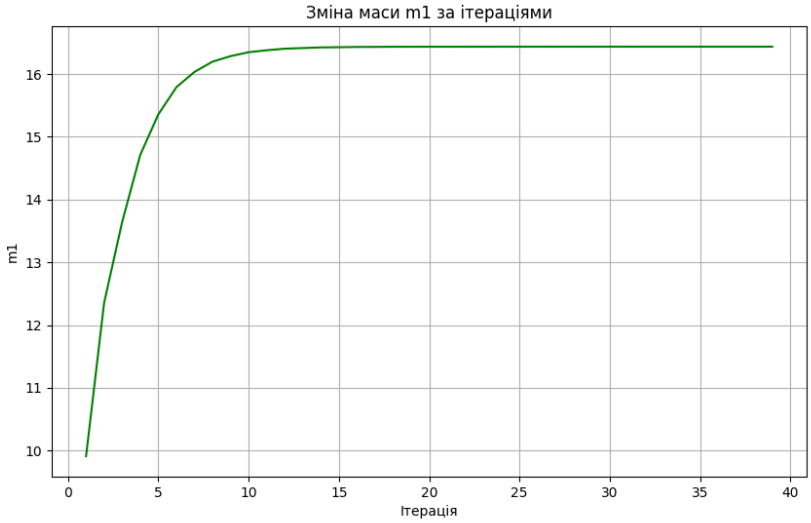
Цей графік показує, як змінюються значення параметрів моделі (c1, c3, m1) з кожною ітерацією методу пошуку параметрів. Зміни параметрів жорсткості стабільні та зменшуються з часом, це означає, що алгоритм наближається до оптимальних значень. Параметр маси спочатку (1-5 ітерація) змінюється доволі сильно , але надалі стає стабільнішим , це може свідчити про те, що на початку метод не сходиться або вибране початкове наближення далеко від оптимуму.

Графік зміни показника якості (MSE) за ітераціями



Цей графік показує зміни середньоквадратичної помилки (MSE) після кожної ітерації. В нашому випадку MSE зменшується, це свідчить про покращення точності моделі.

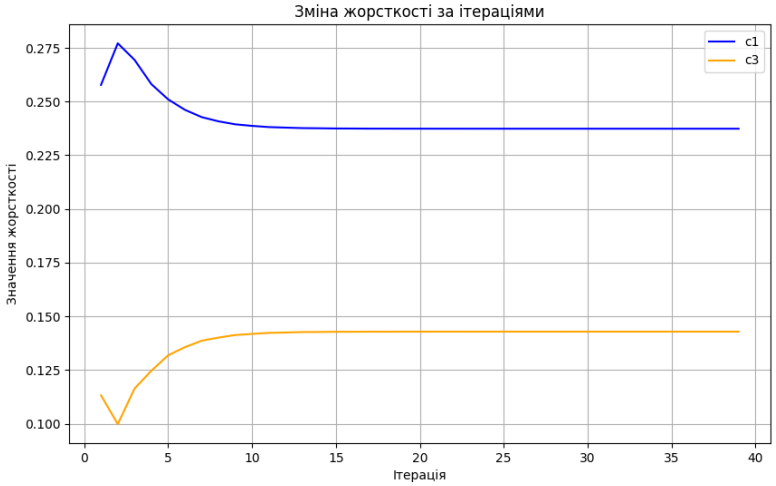
Графік зміни маси (m1) за ітераціями



Цей графік відображає, як змінюється значення маси (m1) за кожною ітерацією.

Він надає уявлення про те, чи стабільно коригується параметр маси. На початку помітне різке коливання що вказує на нестабільність чисельного методу, але поступово зміни стаютьі менш помітні, це вказує на збіжність алгоритму.

Графік зміни жорсткості (c1, c3) за ітераціями



Цей графік показує зміни параметрів жорсткості (c1, c3) за кожну ітерацію.

Параметри жорсткості на початку теж змінюються різко , допоки метод не продовжує свою дію . Надалі вони змінюються м’яко і стабільно, це означає, що модель знаходить оптимальні значення для цих параметрів.

Графік зміни часу виконання за ітераціями

Відображає час виконання кожної ітерації.

Час виконання зростає найбільше на 20 ітерації , але насправді ці значення є нестабільними , і щоразу кожна ітерація може займати різний час виконання .

Код розв'язку:

```
import numpy as np
import time
import matplotlib.pyplot as plt

# Завантаження даних з файлу
data = np.loadtxt('y3.txt').T

# Відомі параметри системи
c2, c4, m2, m3 = 0.3, 0.12, 28, 18
t0, T, deltaT = 0, 50, 0.2   # Початковий час (t0), кінцевий час (T), крок по часу (deltaT)
epsilon = 1e-5   # Точність для завершення ітерацій

# Початкове наближення
c1, c3, m1 = 0.1, 0.1, 9

# Функція для розрахунку чутливості матриці для моделі
def SensMatrix(b):
    # Означення зворотних значень для m1, m3, b2 (в залежності від параметрів b)
    m1_inv, m3_inv, b2_inv = 1 / m1, 1 / m3, 1 / b[2] if b[2] != 0 else 0
    return np.array([
        [0, 1, 0, 0, 0, 0],
        [- (b[1] + b[0]) * m1_inv, 0, b[1] * m1_inv, 0, 0, 0],
        [0, 0, 0, 1, 0, 0],
        [b[1] * b2_inv, 0, -(b[1] + c3) * b2_inv, 0, c3 * b2_inv, 0],
        [0, 0, 0, 0, 0, 1],
        [0, 0, c3 * m3_inv, 0, -(c4 + c3) * m3_inv, 0]
    ])

# Функція для розрахунку похідних моделі щодо параметрів
def ModelDerivatives(y, b):
    #  матриці для обчислення похідних по кожному параметру (db0, db1, db2)
    db0 = np.zeros((6, 6))
    db1 = np.zeros((6, 6))
    db2 = np.zeros((6, 6))

    # Заповнення значень для кожної матриці
    db0[1, 0] = -1 / m1
    db1[1, 0] = -1 / m1
    db1[1, 2] = 1 / m1
    db2[3, 0] = -b[1] / (b[2] ** 2)
    db2[3, 2] = (b[1] + c3) / (b[2] ** 2)
    db2[3, 4] = -c3 / (b[2] ** 2)

    # Множення на y (дані)
    db0 = np.dot(db0, y)
    db1 = np.dot(db1, y)
```

```
    db2 = np.dot(db2, y)

    return np.array([db0, db1, db2]).T

# Функція для обчислення чутливості за допомогою методу Рунге-Кутта
def Sensitivity_RK(A, db, uu, deltaT, timeStamps):
    # Розв'язання чутливості по методу Рунге-Кутти
    for i in range(1, len(timeStamps)):
        k1 = deltaT * (np.dot(A, uu[i - 1]) + db[i - 1])
        k2 = deltaT * (np.dot(A, (uu[i - 1] + k1 / 2)) + db[i - 1])
        k3 = deltaT * (np.dot(A, (uu[i - 1] + k2 / 2)) + db[i - 1])
        k4 = deltaT * (np.dot(A, (uu[i - 1] + k3)) + db[i - 1])

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

    return uu

# Функція для моделювання за допомогою методу Рунге-Кутта
def Model_RK(b, timeStamps, deltaT):
    #ініціалізуємо результати
    yy = np.zeros_like(data)
    yy[0] = data[0].copy()
    A = SensMatrix(b)  # Обчислення матриці чутливості для заданих параметрів
b

    # розв'язування рівнянь по методу Рунге-Куттf
    for i in range(1, len(timeStamps)):
        y_prev = yy[i - 1]
        k1 = deltaT * np.dot(A, y_prev)
        k2 = deltaT * np.dot(A, (y_prev + k1 / 2))
        k3 = deltaT * np.dot(A, (y_prev + k2 / 2))
        k4 = deltaT * np.dot(A, (y_prev + k3))
        yy[i] = y_prev + (k1 + 2 * k2 + 2 * k3 + k4) / 6
    return yy

# Функція для розрахунку зміщень параметрів (deltaB)
def DeltaB(uu, db, deltaT, timeStamps, data, b):
    # Різниця між реальними даними та модельним прогнозом
    diff_y = data - Model_RK(b, timeStamps, deltaT)

    # Визначення зміщень параметрів
    du = (np.array([u.T @ u for u in uu]) * deltaT).sum(0)
    du_inv = np.linalg.inv(du) # Обернена матриця
    uY = (np.array([uu[i].T @ diff_y[i] for i in range(len(timeStamps))]) *
deltaT).sum(0)
    deltaB = du_inv @ uY # Обчислення зміщення

    return deltaB

# Функція для обчислення середньоквадратичної помилки (MSE)
def MSE(data, model):
    return np.mean((data - model) ** 2)

# Основна функція для пошуку параметрів
def Parameters(b, t0, T, deltaT, eps, max_iter=100):
    # Створення масиву часових міток
    timeStamps = np.linspace(t0, T, int((T - t0) / deltaT + 1))
```

```
    iteration_count = 0
    prev_b = b.copy()

    iteration_results = []  # Список для збереження результатів ітерацій
    iteration_times = []  # Зберігання часу виконання кожної ітерації

    # Основний цикл для пошуку параметрів
    while iteration_count < max_iter:
        iteration_count += 1

        start_iter_time = time.time()  # Початок часу для ітерації
        yy = Model_RK(b, timeStamps, deltaT)
        uu = np.zeros((len(timeStamps), 6, 3))
        db = ModelDerivatives(yy.T, b)
        A = SensMatrix(b)
        uu = Sensitivity_RK(A, db, uu, deltaT, timeStamps)
        deltaB = DeltaB(uu, db, deltaT, timeStamps, data, b)

        b += deltaB
        mse = MSE(data, Model_RK(b, timeStamps, deltaT))

        iteration_results.append((iteration_count, b.copy(), mse))

        end_iter_time = time.time()  # Кінець часу для ітерації
        iteration_times.append(end_iter_time - start_iter_time)

        if np.abs(deltaB).max() < eps:
            break

    return b, iteration_count, iteration_results, iteration_times


# Головна функція
if __name__ == "__main__":
    start_time = time.time()
    solution, iteration_count, iteration_results, iteration_times =
Parameters(
        np.array([c1, c3, m1]), t0, T, deltaT, epsilon
    )
    end_time = time.time()
    execution_time = end_time - start_time

    solution = np.round(solution, 4)

    # Виведення результатів
    print("  Ітерація |                  Невідомі параметри
|    Показник якості")
    print("-" * 90)

    for iteration, params, mse in iteration_results:
        print(f"{iteration:10} |     c1 = {params[0]:.6f}, c3 =
{params[1]:.6f}, m1 = {params[2]:.6f}     |      {mse:.8f}")

    print("-" * 90)
    print(f"Знайдені параметри: c1 = {solution[0]:.6f}, c3 =
{solution[1]:.6f}, m1 = {solution[2]:.6f}")
```

```
    print(
        f"Показник якості: Q = {MSE(data, Model_RK(solution, np.linspace(t0,
T, int((T - t0) / deltaT + 1)), deltaT)):.8f}")
    print(f"Час виконання: {execution_time:.6f} секунд")

    # Графік : Зміни параметрів (c1, c3, m1) за ітераціями
    iterations = [iteration for iteration, _, _ in iteration_results]
    c1_values = [params[0] for _, params, _ in iteration_results]
    c3_values = [params[1] for _, params, _ in iteration_results]
    m1_values = [params[2] for _, params, _ in iteration_results]

    plt.figure(figsize=(10, 6))
    plt.plot(iterations, c1_values, label='c1')
    plt.plot(iterations, c3_values, label='c3')
    plt.plot(iterations, m1_values, label='m1')
    plt.xlabel('Ітерація')
    plt.ylabel('Значення параметра')
    plt.title('Зміни параметрів за ітераціями')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Графік : MSE за ітераціями
    mse_values = [mse for _, _, mse in iteration_results]

    plt.figure(figsize=(10, 6))
    plt.plot(iterations, mse_values, label='MSE', color='violet')
    plt.xlabel('Ітерація')
    plt.ylabel('MSE')
    plt.title('Показник якості за ітераціями')
    plt.grid(True)
    plt.show()

    # Графік : Зміна маси (m1) за ітераціями
    plt.figure(figsize=(10, 6))
    plt.plot(iterations, m1_values, label='m1', color='green')
    plt.xlabel('Ітерація')
    plt.ylabel('m1')
    plt.title('Зміна маси m1 за ітераціями')
    plt.grid(True)
    plt.show()

    # Графік : Зміна жорсткості (c1, c3) за ітераціями
    plt.figure(figsize=(10, 6))
    plt.plot(iterations, c1_values, label='c1', color='blue')
    plt.plot(iterations, c3_values, label='c3', color='orange')
    plt.xlabel('Ітерація')
    plt.ylabel('Значення жорсткості')
    plt.title('Зміна жорсткості за ітераціями')
    plt.legend()
    plt.grid(True)
    plt.show()

# Графік : Зміна часу виконання за ітераціями
plt.figure(figsize=(10, 6))
plt.plot(iterations, iteration_times, label='Час виконання', color='red')
plt.xlabel('Ітерація')
```

```
plt.ylabel('Час (с)')
plt.title('Час виконання за ітераціями')
plt.grid(True)
plt.show()
```

Оптимізація :

Умова цієї лабораторної роботи полягала в тому , щоб створити код для вирішення задачі та максимально оптимізувати його для покращення швидкодії .

У своїй роботі я використовувала такі методи оптимізації :

1. **Зменшення зайвих обчислень** – замість того , щоб багато разів обчислювати одні й ті ж значення, наприклад, `A` для кожної ітерації (у функціях Sensitivity_RK та Model_RK) - можна обчислити один раз перед циклом і використовувати в циклі, щоб зменшити витрати на обчислення.

2. **Мінімізація використання циклів і покращення роботи з масивами** – замінила обрахунки (дозволені за умовою роботи) на векторизовані операції з NumPy .

3. **Оптимізація функцій, що пов'язані з обчисленням чутливості та похибок** - перевела обчислення `MSE` на векторизовану форму для зменшення витрат часу.