Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики

Звіт
лабораторної роботи №3
з моделювання складних систем
Варіант 9

Виконала:
студентка групи ІПС-31
Мельник Поліна Володимирівна

Київ 2024

Постановка задачі:

Для математичної моделі коливань 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.

Варіант 9: Вектор оцінюваних параметрів $\beta = (c_2, c_4, m_1)^T$, початкове наближення $\beta_0 = (0.2, 0.1, 9)^T$, відомі параметри $c_1 = 0.14$, $c_3 = 0.2$, $m_2 = 28$, $m_3 = 18$, ім'я файлу з спостережуваними даними y9.txt.

Спостереження стану моделі проведені на інтервалі часу $t_0 = 0$, $t_k = 50$, $\Delta t = 0.2$.

Реалізація з фрагментами коду:

Спочатку зчитуємо дані з файлу y9.txt та транспонуємо, щоб кожен стовпчик був окремим виміром.

```
data = np.loadtxt('y9.txt').T
```

Далі ініціалізуємо параметри, такі як маси (`m1`, `m2`, `m3`), коефіцієнти (`c1`, `c3`, `c2`, `c4`), часову сітку (`t0`, `T`, `deltaT`), а також точність зупинки (`epsilon`).

```
c1, c3, m2, m3 = 0.14, 0.2, 28, 18
t0, T, deltaT = 0, 50, 0.2
epsilon = 1e-5
c2, c4, m1 = 0.2, 0.1, 9
```

Далі реалізовуємо функції програми:

- Функція **SensMatrix:** створює матрицю чутливості для системи рівнянь, використовуючи параметри вектора `b`. Матриця використовує зворотні значення мас `m1`, `m3` і коефіцієнт `b[2]` (для `c2`).

```
def SensMatrix(b):  2 usages
    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 **ModelDerivatives**: Обчислює похідні моделі для параметрів, що залежить від значення вектора `y` і параметрів `b`. Функція використовує зворотні похідні для отримання чутливості.

```
def ModelDerivatives(y, b):  1 usage
    db0 = np.zeros((6, 6))
    db1 = np.zeros((6, 6))
    db2 = np.zeros((6, 6))

    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)

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

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

- Функція **Sensitivity_RK**: використовує метод Рунге-Кутта 4-го порядку для інтеграції чутливості. Обчислюється зміна параметрів чутливості за часом.

```
def Sensitivity_RK(A, db, uu, deltaT, timeStamps):  1 usage
    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 **Model_RK**: обчислює модель за допомогою методу Рунге-Кутта 4-го порядку. Це моделює поведінку системи за часом.

```
def Model_RK(b, timeStamps, deltaT):  2 usages
    yy = np.zeros_like(data)
    yy[0] = data[0].copy()
    A = SensMatrix(b)

    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 **DeltaB**: обчислює зміни параметрів `deltaB` на основі різниці між моделлю та фактичними даними. Для цього використовується чутливість, яка була обчислена раніше.

```python
def DeltaB(uu, db, deltaT, timeStamps, data, b):  1 usage
    diff_y = data - Model_RK(b, timeStamps, deltaT)

    du = (np.array([u.T @ u for u in uu]) * deltaT).sum(0)
    du_inv = np.linalg.inv(du)
    uY = (np.array([uu[i].T @ diff_y[i] for i in range(len(timeStamps))]) * deltaT).sum(0)
    deltaB = du_inv @ uY

    return deltaB
```

Далі реалізовуємо основну функцію Parameters, яка використовується для ітераційного знаходження параметрів. Використовує метод Рунге-Кутта для моделювання та обчислення чутливості, оновлюючи параметри на кожній ітерації. Коли різниця між параметрами на поточній та попередній ітераціях менша за поріг, зупиняється. Також відображає графік зміни параметрів.

```python
def Parameters(b, t0, T, deltaT, eps, max_iter=1000):  1 usage
    timeStamps = np.linspace(t0, T, int((T - t0) / deltaT + 1))

    iteration_count = 0
    history = []
    prev_b = b.copy()
    diff_y_history = []

    c2_history = []
    c4_history = []
    m1_history = []

    while iteration_count < max_iter:
        iteration_count += 1

        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

        history.append(b.copy())

        c2_history.append(b[0])
        c4_history.append(b[1])
        m1_history.append(b[2])

        if np.abs(deltaB).max() < eps:
            print(f"Convergence reached at iteration {iteration_count}")
            break

        if np.allclose(b, prev_b, atol=epsilon):
            break

        prev_b = b.copy()

    # Побудова графіка зміни параметрів
    plt.figure(figsize=(10, 6))

    plt.plot( *args: range(iteration_count), c2_history, label='c2', color='blue')
    plt.plot( *args: range(iteration_count), c4_history, label='c4', color='orange')
    plt.plot( *args: range(iteration_count), m1_history, label='m1', color='green')

    plt.xlabel('Ітерація')
    plt.ylabel('Значення параметру')
    plt.title('Зміна патаметрів протягом ітерацій')
    plt.legend()
    plt.grid(True)

    plt.show()

    np.savetxt( fname: "parameters_history.txt", np.column_stack((c2_history, c4_history, m1_history)))

    return b, iteration_count, history, diff_y_history
```

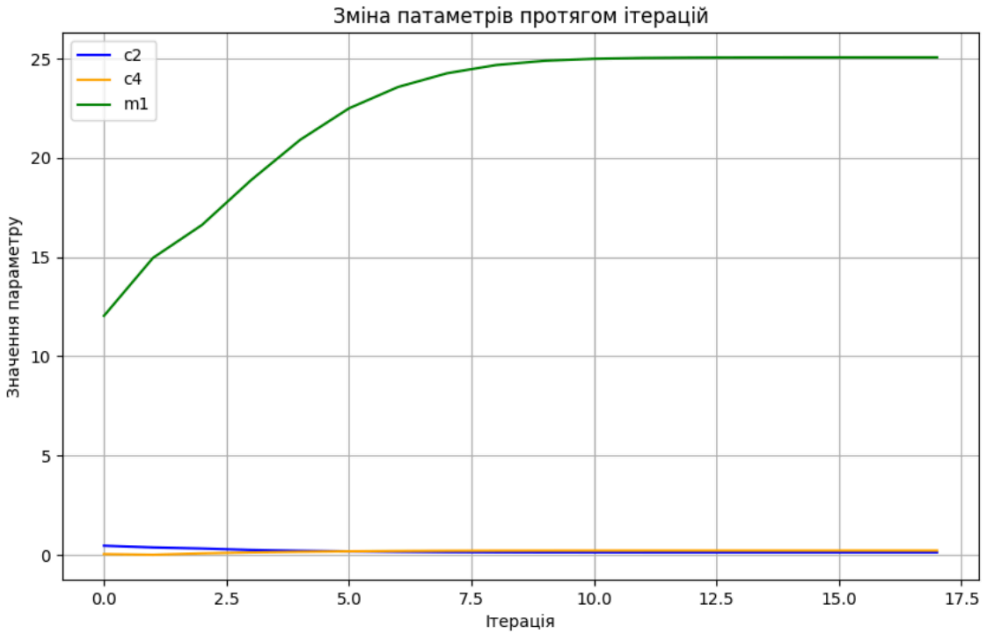
Далі реалізована основна частина коду, що викликає функцію `Parameters` для отримання розв'язку. Виводиться кількість ітерацій та кінцевий результат.
Також зберігається історія параметрів у файл `parameters_history.txt`.

```
if __name__ == "__main__":
    start_time = time.time()
    solution, iteration_count, history, diff_y_history = Parameters(np.array([c2, c4, m1]), t0, T, deltaT, epsilon)
    end_time = time.time()
    execution_time = end_time - start_time

    solution = np.round(solution, decimals: 4)

    print(f"Результати: c2 = {solution[0]}, c4 = {solution[1]}, m1 = {solution[2]}")
    print("Кількість ітерацій:", iteration_count)

    np.savetxt( fname: "final_parameters.txt", X: [solution], delimiter=",", header="c2,c4,m1", comments="")
```

Результати виконання:

```
Результати: c2 = 0.1231, c4 = 0.2185, m1 = 25.0607
Кількість ітерацій: 18
```

Та графік для показу зміни значень параметрів протягом виконання ітерацій:



Тепер більш детально розглянемо функції та їх елементи:

- Функція SensMatrix(b)

Функція генерує матрицю чутливості системи, яка використовується для обчислення чутливості розв'язку на зміни параметрів b, де $b = [c_2, c_4, m_1]$.

- **Матриця чутливості (A)**:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{b_1+b_0}{m_1} & 0 & \frac{b_1}{m_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{b_1}{b_2} & 0 & -\frac{b_1+c_3}{b_2} & 0 & \frac{c_3}{b_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{c_3}{m_3} & 0 & -\frac{c_3+c_4}{m_3} & 0 \end{bmatrix}$$

- Вектори $b = [c_2, c_4, m_1]$задають значення параметрів, на основі яких розраховуються елементи матриці.
- **Розмірність**: матриця розміром 6×6, оскільки система складається з 6 рівнянь.

- Функція ModelDerivatives(y, b)

Функція обчислює похідні моделі, що описують зміну стану системи з часом. Це допомагає при обчисленні чутливості та підвищує точність чисельного інтегрування.

- **Вхідні параметри**:
  - y — вектор стану системи (розмірність 6) на поточному кроці часу.
  - b — параметри моделі (масштаби і коефіцієнти) розмірністю 3.
- **Вихід**: Матриця розміром 3×6, яка містить похідні для трьох параметрів системи.

- Функція Sensitivity_RK
Функція для обчислення чутливості за допомогою методу Рунге-Кутти для системи рівнянь. Тут враховуються похідні та зміни параметрів, що дозволяють коригувати значення параметрів b за допомогою методу найменших квадратів.

- **Вхідні параметри**:
  - A — матриця чутливості, розмірність 6×6.
  - db — матриця похідних параметрів, розмірність 3×6.
  - uu — масив чутливостей для кожного часу, розмірність (n×6×3), де n — кількість часових кроків.
- **Вихід**: Оновлені чутливості uu, розмірність n×6×3.

- Функція Model_RK
Функція для чисельного інтегрування рівнянь руху системи за допомогою методу Рунге-Кутти (4-го порядку). Вона надає рішення рівнянь для всіх часових моментів.

- **Вхідні параметри**:
  - b — параметри моделі (масштаби і коефіцієнти), розмірність 3.
  - timeStamps — часові мітки для обчислення розв'язку, розмірність n.
  - deltaT — крок часу для чисельного інтегрування.
- **Вихід**: Масив станів системи в кожен момент часу, розмірність n×6.

- Функція DeltaB
Функція для обчислення зміни параметрів Δb на основі розбіжностей між моделлю та даними, за допомогою методу найменших квадратів.

- **Вхідні параметри**:
  - uu — чутливості для кожного моменту часу, розмірність n×6×3.
  - db — похідні параметрів, розмірність 3×6.
  - deltaT — крок часу.
  - data — набір даних, розмірність n×6.
  - b — поточні значення параметрів.
- **Вихід**: Оновлення параметрів Δb, розмірність 3.

- Функція Parameters

Головна функція для оцінки параметрів c_2, c_4 та m_1 за допомогою ітераційної процедури. Вона запускає моделювання та обчислює необхідні параметри через порівняння обчислених даних із реальними даними з файлу.

На вході приймає вектор початкових значень параметрів $b = [c_2, c_4, m_1]$ та інші числові параметри часу.

Використовує Model_RK() для обчислення поточного стану моделі і Sensitivity_RK() для оцінки чутливості.

Розмірність:

- b: 3×1 — вектор початкових значень параметрів.
- timeStamps: N×1 — часові мітки, де N — кількість часових кроків.

Повертає кінцеві значення параметрів та історію ітерацій.