Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук і кібернетики

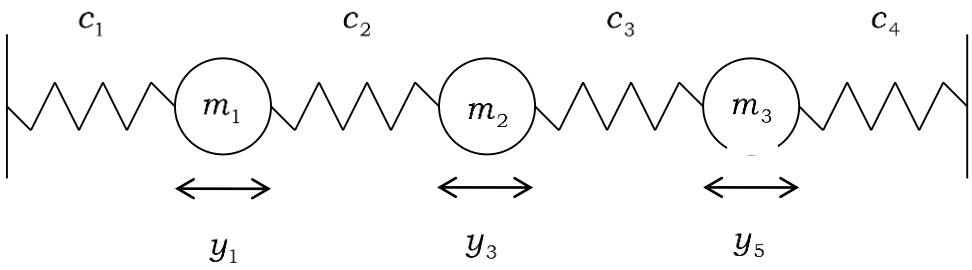
Звіт
з лабораторної роботи №3
з моделювання складних систем
Варіант 4

Виконав:
Студент групи ІПС-31
Вербицький Артем Віталійович

Київ
2024

1. Мета лабораторної роботи

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 $\overline{y}(t)$, $t \in [t_0, t_k]$ потрібно оцінити частину невідомих параметрів моделі з використанням функції чутливості.



2. Постановка задачі

Математична модель коливання трьох мас описується наступною системою

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

Показник якості ідентифікації параметрів невідомих параметрів β має вигляд

$$I(\beta) = \int_{t_0}^{t_k} (\overline{y}(t) - y(t))^T (\overline{y}(t) - y(t)) dt.$$

Якщо представити вектор невідомих параметрів $\beta = \beta_0 + \Delta\beta$, де β_0 – початкове наближення вектора параметрів,

$$\Delta\beta = \left(\int_{t_0}^{t_k} U^T(t) U(t) dt \right)^{-1} \int_{t_0}^{t_k} U^T(t) (\overline{y}(t) - y(t)) dt.$$

Матриці чутливості $U(t)$ визначається з наступної матричної системи диференціальних рівнянь

$$\frac{dU(t)}{dt} = \frac{\partial(Ay)}{\partial y^T} U(t) + \frac{\partial(Ay)}{\partial \beta^T},$$
$$U(t_0) = 0, \ \beta = \beta_0.$$

В даному випадку $\frac{\partial(Ay)}{\partial y^T} = A$.

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

Для чисельного інтегрування застосувати метод Рунге-Кутта 4-го порядку:

$$\frac{dy}{dt} = f(y,t), y(t_0) = y_0,$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

де

$$k_1 = hf(y_n, t_n).$$

$$k_2 = hf(y_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h),$$

$$k_3 = hf(y_n + \frac{1}{2}k_2, t_n + \frac{1}{2}h),$$

$$k_4 = hf(y_n + k_3, t_n + h),$$

$$t_{n+1} = t_n + h.$$

Вектор оцінюваних параметрів $\beta = (m_2, c_3, m_3)^T$, початкове наближення $\beta_0 = (21, 0.15, 11)^T$, відомі параметри $c_1 = 0.14, c_2 = 0.3,\ c_4 = 0.12$, $m_1 = 12$, ім'я файлу з спостережуваними даними y4.txt.

3. Хід роботи

Програмна реалізація виконана мовою Python. Програма реалізована у файлі **MCS_Lab_3.py**.

Методи:

ensure_logs_directory

- Перевіряє чи існує директорія 'logs'
- Якщо її немає - створює її
- Використовується для збереження результатів роботи програми

```
def ensure_logs_directory():
    """Create logs directory if it doesn't exist"""
    if not os.path.exists('logs'):
        os.makedirs('logs')
```

read_file

- Читає вхідний файл з даними
- Розбиває кожен рядок на числові значення
- Конвертує значення у float
- Повертає транспонований numpy масив даних

```
def read_file(file_name):
    with open(file_name, 'r') as file:
        lines = file.readlines()
        input_data = []
        for line in lines:
            values = line.strip().split()
            row = [float(value) for value in values]
            input_data.append(row)
    return np.array(input_data).T
```

finite_diff

- Обчислює матрицю скінченних різниць
- Параметри:
  - **y_vec_func:** векторна функція
  - **b_vec:** вектор параметрів
  - **b_values:** значення параметрів
  - **delta:** крок для обчислення різниць
- Використовує центральну різницеву схему для обчислення похідних
- Повертає матрицю похідних

```python
def finite_diff(y_vec_func, b_vec, b_values, delta=1e-5):
    n = len(y_vec_func(b_values))
    m = len(b_vec)
    deriv_matrix = np.zeros((n, m))

    for j in range(m):
        original_value = b_values[b_vec[j]]
        b_values[b_vec[j]] = original_value + delta
        y_plus = y_vec_func(b_values)
        b_values[b_vec[j]] = original_value - delta
        y_minus = y_vec_func(b_values)
        b_values[b_vec[j]] = original_value
        deriv_matrix[:, j] = (y_plus - y_minus) / (2 * delta)

    return deriv_matrix
```

get_u_matr

- Реалізує метод Рунге-Кутти 4-го порядку для матриць
- Обчислює наступний крок для матриці U
- Використовує класичну формулу RK4 з чотирма коефіцієнтами k1-k4

```python
def get_u_matr(a_matr, b_matr, u_matr, h):
    b_arrayed = np.array(b_matr)
    k1 = h * (np.dot(a_matr, u_matr) + b_arrayed)
    k2 = h * (np.dot(a_matr, u_matr + k1 / 2) + b_arrayed)
    k3 = h * (np.dot(a_matr, u_matr + k2 / 2) + b_arrayed)
    k4 = h * (np.dot(a_matr, u_matr + k3) + b_arrayed)
    return u_matr + (k1 + 2 * k2 + 2 * k3 + k4) / 6
```

get_y

- Також реалізує метод Рунге-Кутти 4-го порядку
- Обчислює наступний крок для вектора Y
- Аналогічний до **get_u_matr**, але для векторного випадку

```python
def get_y(a_matr, y_cur, h):
    k1 = h * np.dot(a_matr, y_cur)
    k2 = h * np.dot(a_matr, y_cur + k1 / 2)
    k3 = h * np.dot(a_matr, y_cur + k2 / 2)
    k4 = h * np.dot(a_matr, y_cur + k3)
    return y_cur + (k1 + 2 * k2 + 2 * k3 + k4) / 6
```

init_matr

- Ініціалізує матрицю системи
- Приймає словник параметрів (c1-c4, m1-m3)
- Повертає numpy масив з матрицею системи

```
def init_matr(params):
    c1, c2, c3, c4, m1, m2, m3 = params['c1'], params['c2'], params['c3'], params['c4'], params['m1'], params['m2'], params['m3']
    matr = [
        [0, 1, 0, 0, 0, 0],
        [-(c2 + c1) / 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]
    ]
    return np.array(matr)
```

get_model_solution

- Обчислює розв'язок моделі
- Параметри:
  - **params:** параметри системи
  - **y0:** початкові умови
  - **t_points:** часові точки
  - **h:** крок інтегрування
- Використовує метод **get_y** для обчислення траєкторії
- Повертає масив розв'язків

```
def get_model_solution(params, y0, t_points, h=0.2):
    a_matrix = init_matr(params)
    y_current = y0
    y_solution = [y0]

    for _ in range(len(t_points) - 1):
        y_current = get_y(a_matrix, y_current, h)
        y_solution.append(y_current)

    return np.array(y_solution)
```

approximate

- Основна функція апроксимації
- Виконує ітеративний процес наближення параметрів
- Параметри:
  - **y_matr:** виміряні дані
  - **params:** фіксовані параметри
  - **beta_symbols:** символи параметрів для апроксимації
  - **beta_values:** початкові значення параметрів
  - **eps:** точність
  - **h:** крок інтегрування
- Зберігає проміжні результати в CSV файл

- Повертає знайдені параметри та статистику роботи

```
def approximate(y_matr, params, beta_symbols, beta_values, eps, h=0.2):···
```

plot_results

- Візуалізує результати
- Створює графіки порівняння виміряних даних та моделі
- Параметри:
  - **measured_data:** виміряні дані
  - **model_solution:** розв'язок моделі
  - **t_points:** часові точки
  - **save_prefix:** префікс для імені файлу
- Зберігає графіки у директорію logs

```
def plot_results(measured_data, model_solution, t_points, save_prefix):···
```

4. Асимптотичні оцінки

1. **Функція finite_diff:**
   - Обчислює числову похідну, використовуючи метод кінцевих різниць. Складність залежить від кількості елементів вектора b_vec *m* та кількості елементів у векторі y_vec_func(b_values) *n*.
   - Часова складність: O(n · m).
2. **Функція get_u_matr та get_y:**
   - Використовують метод Рунге-Кутта 4-го порядку для обчислення значень матриці u_matr та вектора y_cur. Обидві мають сталу кількість обчислень (4 обчислення векторного добутку та суми), отже, мають лінійну складність залежно від розміру матриць.
   - Часова складність: O(1).
3. **Функція approximate:**
   - Найбільш обчислювально інтенсивна, з циклом while, що завершується, коли quality_degree стає меншим за eps.
   - Внутрішній цикл виконує числову диференціацію (через finite_diff), яка має O(n · m), та кілька матричних обчислень, що залежать від кількості точок даних у y_matr (позначимо як T).
   - Залежно від кількості ітерацій k, асимптотична складність: O(k · T · n · m).
4. **Функція main:**
   - Виконує читання даних та виклик approximate, а також підсумкове моделювання результатів і візуалізацію.
   - Оскільки approximate є найбільш ресурсомісткою частиною, загальна складність алгоритму буде близька до складності approximate.

Отже, асимптотичні оцінки для функцій виглядають так:

- **Часова складність** T() = O(k · T · n · m).
- **Просторова складність** O(n · m) (для зберігання результатів обчислень).

5. Аналіз результатів

1. Збіжність алгоритму:

- Алгоритм зійшовся за 5 ітерацій
- Показник якості зменшився з 6.53 до 1.13e-8
- Швидкість збіжності:
    - Ітерація 0: 6.533
    - Ітерація 1: 0.792 (значне покращення)
    - Ітерація 2: 0.00592
    - Ітерація 3: 2.00e-6
    - Ітерація 4: 1.13e-8 (фінальне значення)
- Час виконання: 0.24 секунди (дуже швидко)

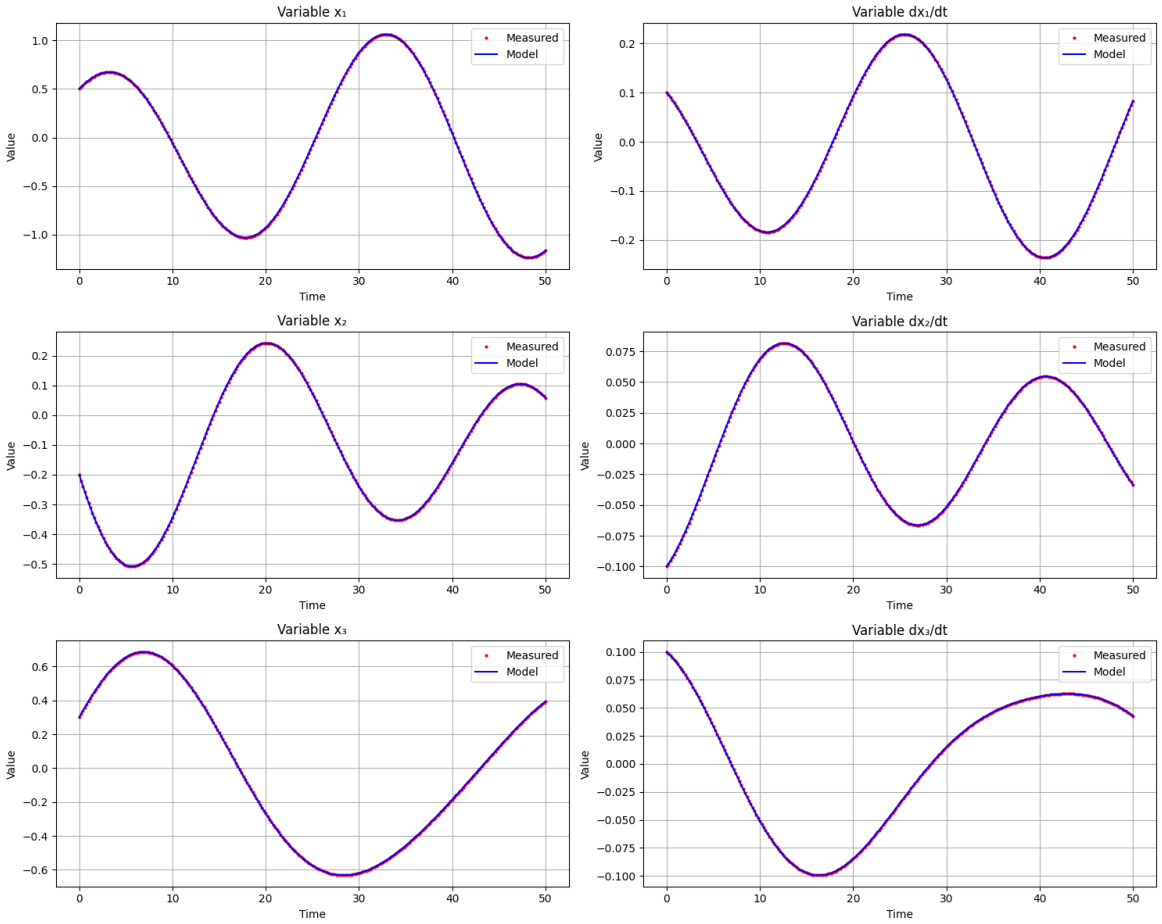
2. Аналіз графіків:

- Для x_1 (перша маса):
    - Амплітуда коливань: від -1.0 до 1.0
    - Чітко видно періодичний характер руху
    - Ідеальне співпадіння моделі з вимірами
- Для x_2 (друга маса):
    - Менша амплітуда коливань: від -0.5 до 0.2
    - Фаза коливань відрізняється від x_1
    - Відмінне співпадіння моделі з даними
- Для x_3 (третя маса):
    - Амплітуда коливань: від -0.6 до 0.6
    - Складніша форма коливань
    - Повне співпадіння моделі з вимірами
- Для швидкостей (dx_1/dt, dx_2/dt, dx_3/dt):
    - Гладкі криві без розривів
    - Менші амплітуди порівняно з положеннями
    - Ідеальне співпадіння модельних та виміряних даних

```
Parameter Identification Results:
--------------------------------

Identified parameters:
m2: 27.999972
c3: 0.200000
m3: 17.999969

Quality indicator: 1.128093e-08
Total iterations: 5
Execution time: 0.24 seconds
```



6. Висновки

Отож, ми програмно реалізували оцінку частини невідомих параметрів моделі з використанням функції чутливості, а також порівняли виміряні дані та результати моделювання.

У ході роботи було реалізовано алгоритм ідентифікації параметрів, який включає Метод Рунге-Кутти 4-го порядку для чилового інтегрування, метод скінченних різниць для обчислення похідних, ітераційний процес оптимізації параметрів.

Було успішно визначено невідомі параметри системи та досягнуто високої точності ідентифікації. Проведено візуалізацію та аналіз результатів, зокрема, побудовано графіки положень та швидкостей мас.

Ми виявили характерні особливості системи: періодичний характер коливань, різні амплітуди коливань для кожної маси, складна взаємодія між масами через пружні зв’язки.