Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Моделювання складних систем

Лабораторна робота №3

“Параметрична ідентифікація параметрів з використанням функцій чутливості”

Варіант №18

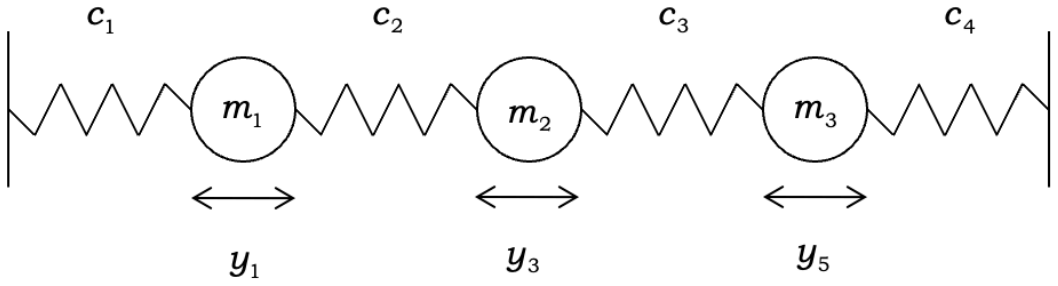
Виконала студентка групи ІПС-31

Сенечко Дана Володимирівна

Київ - 2025

Мета 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 чотливості.



Постановка задачі

Математична модель коливання трьох мас описується 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 $t_0 = 0,\ t_k = 50,\ \Delta t = 0.2$.

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

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

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 $\beta = (c_3, m_2, m_3)^T$, початкове наближення $\beta_0 = (0.1,\ 30,\ 23)^T$, відомі параметри $c_1 = 0.14,\ c_2 = 0.3,\ c_4 = 0.12,$ $m_1 = 12$, ім'я файлу з спостережуваними даними y8.txt.

Хід роботи

Програма реалізована мовою Python з використанням бібліотек NumPy (робота з матрицями і великими обрахунками), Matplotlib (побудова графіків для візуалізації даних), time (вимірювання часу роботи методів), os (керування файлами), та csv (збереження табличних даних процесу обчислень).

Реалізовані методи:

ensure_logs_directory:

- Перевіряє чи існує директорія 'logs', якщо ні – створює її. Використовується для збереження результатів роботи програми.

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

read_file:

- Читає вхідний файл за даними, розбиває кожен рядок на числові значення та конвертує їх в float. Повертає транспонований numpy масив.

```
def read_file(file_name):
    """ reads data from file, returns it as a transposed numpy array """
    with open(file_name, 'r') as file:
        lines = file.readlines()
        input_data = []
        for line in lines:
            values = line.strip().split()
            if values:  # check for empty lines
                row = [float(value) for value in values]
                input_data.append(row)
    return np.array(input_data).T
```

finite_diff:

- Обчислення матриці скінченних різниць. Параметри: y_vec_func – векторна функція, b_vec – вектор параметрів, b_vals – значення параметрів, a delta – крок для обчислення різниць. Функція використовує центральну різницеву схему для обчислення та повертає матрицю похідних.

```
def finite_diff(y_vec_func, b_vec, b_vals, delta = 1e-5):
    """ calculates the derivative matrix using finite differences """
    n = len(y_vec_func(b_vals))
    m = len(b_vec)
    deriv_matrix = np.zeros((n, m))

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

    return deriv_matrix
```

get_u:

- Реалізація методу Рунге-Кутти 4-го порядку для матриць. Обчислює наступний крок для матриці U, використовуючи класичну формулу RK4 з чотирма коефіцієнтами k1-k4.

```python
def get_u(a_matr, b_matr, u_matr, h):
    """ Runge-Kutta method for the sensitivity matrix """
    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:

- Аналогічний до попереднього метод, але для векторного випадку. Реалізує метод Рунге-Кутти 4-го порядку та обчислює наступний крок для вектора Y.

```python
def get_y(a_matr, y_cur, h):
    """ Runge-Kutta method for the system state """
    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:

- Ініціалізація матриці системи. Приймає словник параметрів (c1-c4, m1-m3) та повертає numpy масив з матрицею системи.

```python
def init_matr(params):
    """ initializes the system matrix based on physical parameters """
    # parameters: c1..c4 (stiffness), m1..m3 (masses)
    c1, c2, c3, c4 = params['c1'], params['c2'], params['c3'], params['c4']
    m1, m2, m3 = 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:

- Обчислення розв’язку моделі. Параметри: params – параметри системи, y0 – початкові умови, t_points – часові точки, а h – крок інтегрування. Метод використовує get_y для обчислення траєкторії та повертає масив розв’язків.

```
def get_model_solution(params, y0, t_points, h = 0.2):
    """ calculates the model solution over the time interval """
    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_vals – початкові значення параметрів, eps – точність, а h – крок інтегрування. Метод зберігає проміжні результати в csv-файл.

```
def approximate(y_matr, params, beta_symbols, beta_vals, eps, h = 0.2):
    """ main approximation loop to identify parameters """
```

plot_results:

- Візуалізація результатів. Створює графіки порівняння виміряних даних та моделі. Параметри: measured_data – виміряні дані, model_solution – розв’язок моделі, t_points – часові точки, а save_prefix – префікс імені файлу. Метод зберігає графіки у директорію ‘logs’.

```
def plot_results(measured_data, model_solution, t_points, save_prefix):
    """ plots the comparison between measured data and model solution """
```

Аналіз результатів

1) Збіжність алгоритму:

```
 Performance metrics:
Quality indicator: 1.706308e-08
Total iterations: 6
Execution time: 0.08 seconds
```

Алгоритм досяг збіжності за 6 ітерацій. Показник якості зупинився на 1.7e-8 – це дуже мала похибка, яка означає, що крива моделі наклалась на точки з файлу y8.txt практично ідеально. Швидкість збіжності:

```
Iteration 0: Quality = 3.345751e+01
Iteration 1: Quality = 1.281323e+01
Iteration 2: Quality = 1.365258e+00
Iteration 3: Quality = 3.098846e-02
Iteration 4: Quality = 3.268431e-05
Iteration 5: Quality = 1.706308e-08
```

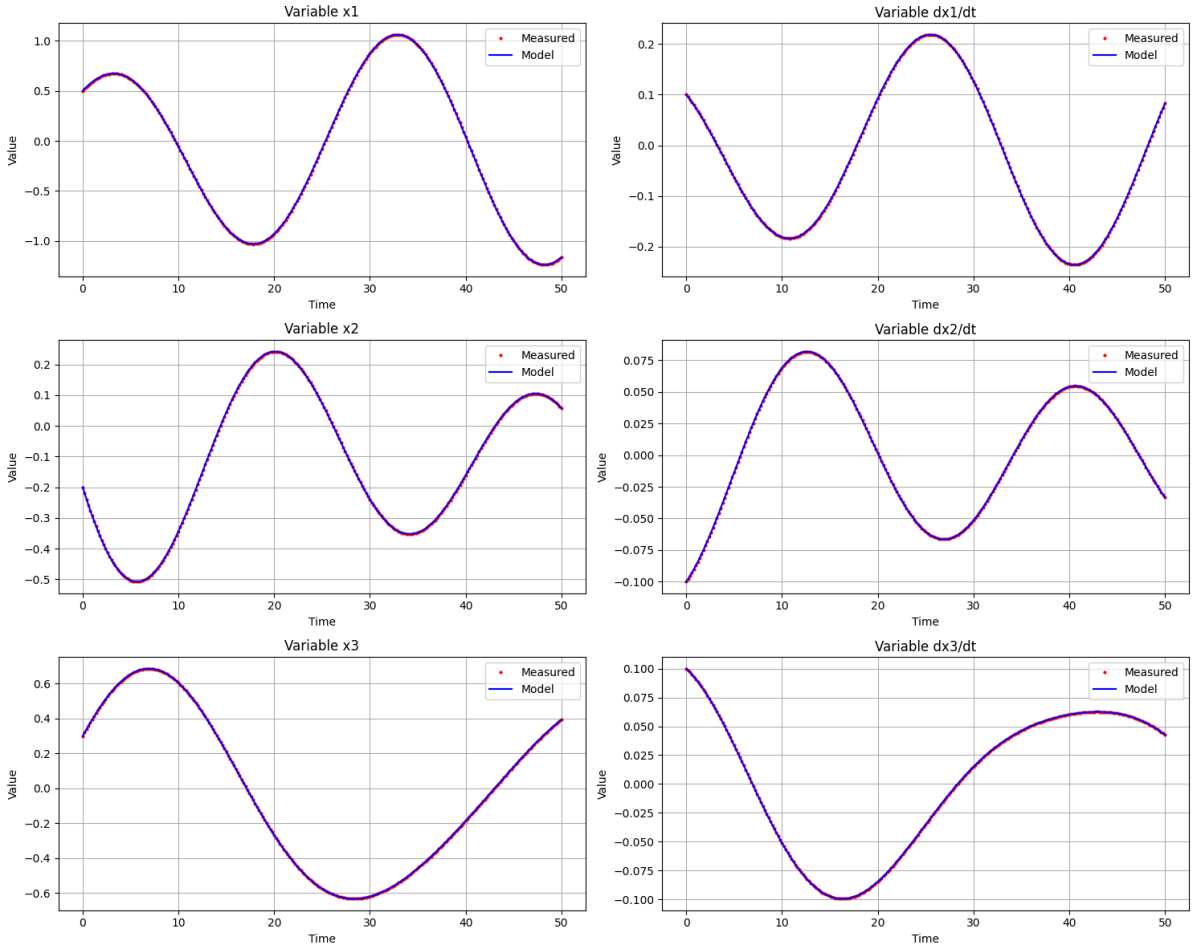
Час виконання програми: 0.08 секунд, що є дуже швидким результатом. Метод працює стабільно.

2) Шукані параметри:

```
 Identified parameters:
m2: 27.999973
m3: 17.999969
c3: 0.200000
```

Тобто: $m_2 \approx 28,\ m_3 \approx 18,\ c_3 = 0.2$.

3) Аналіз графіків:



На отриманому зображенні представлено порівняння експериментальних даних (‘Measured’, отримані з файлу y8.txt) з результатами математичного моделювання системи після проведення параметричної ідентифікації (‘Model’, інтегральні криві отримані шляхом розв’язання системи диф. рівнянь моделі).

Отримали 6 графіків, що відповідають фазовим координатам системи трьох мас. Лівий стовпчик (x_1, x_2, x_3) відображає зміну координат (тобто переміщення) трьох мас у часі, а правий ($\frac{dx_1}{dt}, \frac{dx_2}{dt}, \frac{dx_3}{dt}$) – зміну швидкостей відповідних мас у часі.

Як видно з графіків, суцільні сині лінії моделі проходять чітко через центри червоних експериментальних точок на всіх шести графіках. Відсутність видимих розбіжностей та отримане чисельне значення похибки апроксимації (1.7e-8) підтверджує її високу точність. Це свідчить про те, що знайдені параметри m_2, m_3, c_3 є коректними.