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

Лабораторна робота №3

з Моделювання складних систем

"Параметрична ідентифікація динамічної системи з використанням функцій чутливості"

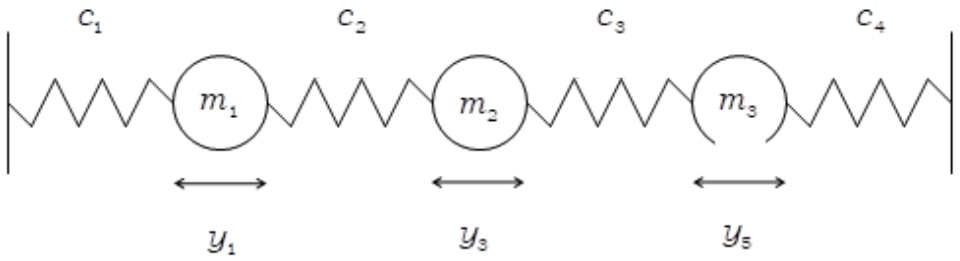
Варіант №12

Виконав студент групи ІПС-31
Тесленко Назар Олександрович

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



Постановка задачі та теоретичні відомості:

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 невідомих параметрів β має вигляд

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

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

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

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 $t_0 = 0,\ t_k = 50,\ \Delta t = 0.2$.

Метод Рунге-Кутта 4-го порядку:

У даній роботі ми маємо справу з диференціальними рівняннями, які описують рух мас та зміну функцій чутливості. Замість аналітичного розв'язку, ми використовуємо чисельний метод, щоб знайти значення координат та матриці чутливості у кожний момент часу t.

Суть методу полягає в тому, щоб визначити значення функції в наступній точці y_{n+1}, знаючи значення в попередній (y_n), використовуючи усереднений нахил графіка на цьому кроці.

Фінальне значення обчислюється як середнє зважене нахилів:

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

де

$k_1 = hf(y_n, t_n)$ - нахил (швидкість) на початку інтервалу

$k_2 = hf(y_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h)$ - нахил посередині інтервалу, обчислений на основі k_1 (пробний крок на пів інтервалу).

$k_3 = hf(y_n + \frac{1}{2}k_2, t_n + \frac{1}{2}h)$ - ще один нахил посередині інтервалу, але вже уточнене за допомогою k_2

$k_4 = hf(y_n + k_3, t_n + h)$ - нахил у кінці інтервалу

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

Варіант 12:

Вектор оцінюваних параметрів $\beta = (m_1, m_2, m_3)^T$ **, початкове наближення** $\beta_0 = (10,\ 18,\ 15)^T$ **, відомі параметри** $c_1 = 0.14, c_2 = 0.3, c_3 = 0.2, c_4 = 0.12$ **, ім'я файлу з спостережуваними даними y2.txt.**

Алгоритм виконання роботи:

Програмна реалізація задачі ідентифікації виконується за наступним алгоритмом:

1. Зчитування даних: Завантажуємо експериментальні дані з файлу (y2.txt) та формуємо матрицю спостережень $\overline{y}(t)$.
2. Ініціалізація параметрів: Встановлюємо відомі коефіцієнти жорсткості ($c_1 - c_4$) та задаємо початкове наближення для невідомих мас $\beta_0 = (10, 18, 15)^T$
3. Ітераційний процес (Головний цикл): Запускаємо цикл уточнення параметрів, який працює до досягнення заданої точності ε:

   3a. Формування матриць: На кожній ітерації будуємо матрицю системи A та матрицю часткових похідних $\frac{\partial A}{\partial \beta}$ для поточних значень мас

   3b. Чисельне інтегрування: Методом Рунге-Кутта 4-го порядку паралельно розв'язуємо рівняння руху для отримання $y(t)$ та рівняння чутливості для отримання $U(t)$ на інтервалі $t \in [0; 50]$

   3c. Обчислення інтегралів: Методом трапецій (або сумуванням) обчислюємо визначені інтеграли, що входять до системи нормальних рівнянь

   3d. Корекція: Знаходимо вектор правок Δβ за вище згаданою формулою (наскільки треба змінити поачтковий вектор шуканих даних, щоб графік моделі став ближчим до реальних даних) та оновлюємо параметри: $\beta_{new} = \beta_{old} + \Delta\beta$.
4. Перевірка критерію зупинки: Якщо інтегральна похибка або величина поправки менша за ε, процес зупиняється.

Хід роботи

Мова реалізації: Python

Реалізовані методи

`read_file():`

- отримує назву файлу
- построчно зчитує дані
- перетворює кожне значення у float
- формує числову матрицю
- повертає її транспонований варіант

```
def read_file(file_name):
    with open(file_name, 'r') as file:
        lines = file.readlines()
    data = []
    for line in lines:
        parts = line.strip().split()
        numbers = [float(x) for x in parts]
        data.append(numbers)
    matrix = np.array(data)
    return matrix.T
```

`ensure_logdir():`

- створює директорію logs/ для збереження логів роботи програми
- перевіряє наявність директорії
- якщо її немає — створює

```
def ensure_logdir():
    if not os.path.exists("logs"):
        os.makedirs("logs")
```

`model_matr():`

- Формує матрицю системи для моделі коливального руху
- Аргументи:

    - p - словник параметрів моделі (m1-m3, k1-k4)
- отримує параметри мас та коефіцієнтів пружності
- підставляє їх у матричний запис системи
- повертає матрицю A розмірності 6×6

```
def model_matr(p):
    #print(params)
    c1, c2, c3, c4 = p["c1"], p["c2"], p["c3"], p["c4"]
    m1, m2, m3 = p["m1"], p["m2"], p["m3"]

    matrix = [[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(matrix)
```

finite_diff():

- чисельне обчислення часткових похідних функції y_func за параметрами моделі методом центральних скінченних різниць, для побудови матриці Якобі, яка використовується у методі ідентифікації параметрів.
- Аргументи:
    - y_func - функція моделі, що для заданого набору параметрів β будуєм матрицю $A(\beta)$ та повертає результативний вектор
    - betas - параметри апроксимації
    - delta - точність для обчислення методу
- повертає матрицю похідних

```
def finite_diff(y_func, betas, delta=1e-5):
    keys=np.array(list(betas.keys()))
    # print("keys: ", keys)
    n=len(y_func(betas))
    m=len(keys)
    deriv_matrix=np.zeros((n,m))
```

```
    for k in range(m):
        orig_value=betas[keys[k]]
        betas[keys[k]]= orig_value + delta
        y_plus=y_func(betas)
        betas[keys[k]]= orig_value - delta
        y_minus=y_func(betas)
        betas[keys[k]]= orig_value
        deriv_matrix[:,k]=(y_plus-y_minus)/(2*delta)
    return deriv_matrix
```

get_U():

- реалізує метод Рунге-Кутта 4-го порядку для знаходження наближеного розв’язку диференціального рівняння матриці U
- Аргументи:
    - A - поточна модель системи
    - partial_der - матриця часткових похідних
    - U - поточна матриця U
    - h - крок інтегрування (=0,2)

```
def get_U(A, partial_der,U, h):
    pd = np.array(partial_der)
    k1 = h * (np.dot(A, U) + pd)
    k2 = h * (np.dot(A, U + k1 / 2) + pd)
    k3 = h * (np.dot(A, U + k2 / 2) + pd)
    k4 = h * (np.dot(A, U + k3) + pd)
    return U + (k1 + 2 * k2 + 2 * k3 + k4) / 6
```

get_y():

- Аналогічно до попереднього методу, метод Рунге-Кутта для знаходження наближеного розв’язку наступного значення стану системи y(t)

```
def get_y(A, y_aprox, h):
    k1 = h * np.dot(A, y_aprox)
    k2 = h * np.dot(A, y_aprox + k1 / 2)
    k3 = h * np.dot(A, y_aprox + k2 / 2)
    k4 = h * np.dot(A, y_aprox + k3)
```

```
    return y_aprox + (k1 + 2 * k2 + 2 * k3 + k4) / 6
```

identify_parameters():

- головна функція алгоритму
- Аргументи:
  - y_matrix - виміряні дані
  - params - відомі фіксовані параметри
  - betas - параметри для апроксимації з їх початковими даними
- ініціалізуємо потрібні дані: матриця A, матриця U, акумулятивні змінні інтегралів та ін.
- на кожній ітерації будуєься нова матриця A на основі поточни параметрів
- для оцінки параметрів розв'язується система за формулою

  $\Delta\beta = (\int U^T U)^{-1} \int U^T (y - y_{aprox})$ і оновлюються параметри

- якщо інтегральна похибка $<$ ε - алгоритм зупиняється і повертаються результати

```
def identify_parameters(y_matrix, params, betas,
eps=1e-5,h=0.2):
     all_params={**params, **betas}
    A_matr=model_matr(all_params)
    #print(A_matr)

    beta_results = np.array([b for b in
betas.values()],dtype=float)

 while True:
        all_params.update(betas)
        A_iter=model_matr(all_params)

        U = np.zeros((6, 3))
        accuracy = 0
        inv_integral = np.zeros((len(betas), len(betas)))
        integral_aprox= np.zeros(len(betas))
```

```
        y_approximation = y_matrix[0]

        for i in range (len(y_matrix)):
            y_vector = lambda b_values:
model_matr({**all_params, **b_values}) @ y_approximation
            partial_der_matrix=finite_diff(y_vector, betas)
            inv_integral += U.T @ U
            integral_aprox +=
U.T@(y_matrix[i]-y_approximation)
            accuracy += (y_matrix[i]-y_approximation).T @
(y_matrix[i]-y_approximation)

            U = get_U(A_iter, partial_der_matrix,U, h)
            y_approximation = get_y(A_iter,y_approximation,h)

        inv_integral*=h
        integral_aprox*=h
        accuracy*=h


        delta_beta = np.linalg.inv(inv_integral) @
integral_aprox.flatten()
        beta_results += delta_beta
        for idx, key in enumerate(betas.keys()):
            betas[key] = beta_results[idx]
    if accuracy < eps:
            return betas, eps, iteration+1, execution_time
        iteration+=1
```

Аналіз результатів

```
============================
IDENTIFIED PARAMETERS
============================
Initialized parametrs:
m1: 10
m2: 18
m3: 15

Identified parameters:
m1: 11.999995551097813
m2: 27.999993269702323
m3: 17.999999830402427

Performance metrics:
Quality indicator: 1.000000e-05
Total iterations: 4
Execution time: 0.12 seconds
============================
```

Шукані параметри:

- $m_1 \approx 12$
- $m_2 \approx 28$
- $m_3 \approx 18$

З виводу у консолі можемо зробити висновки:

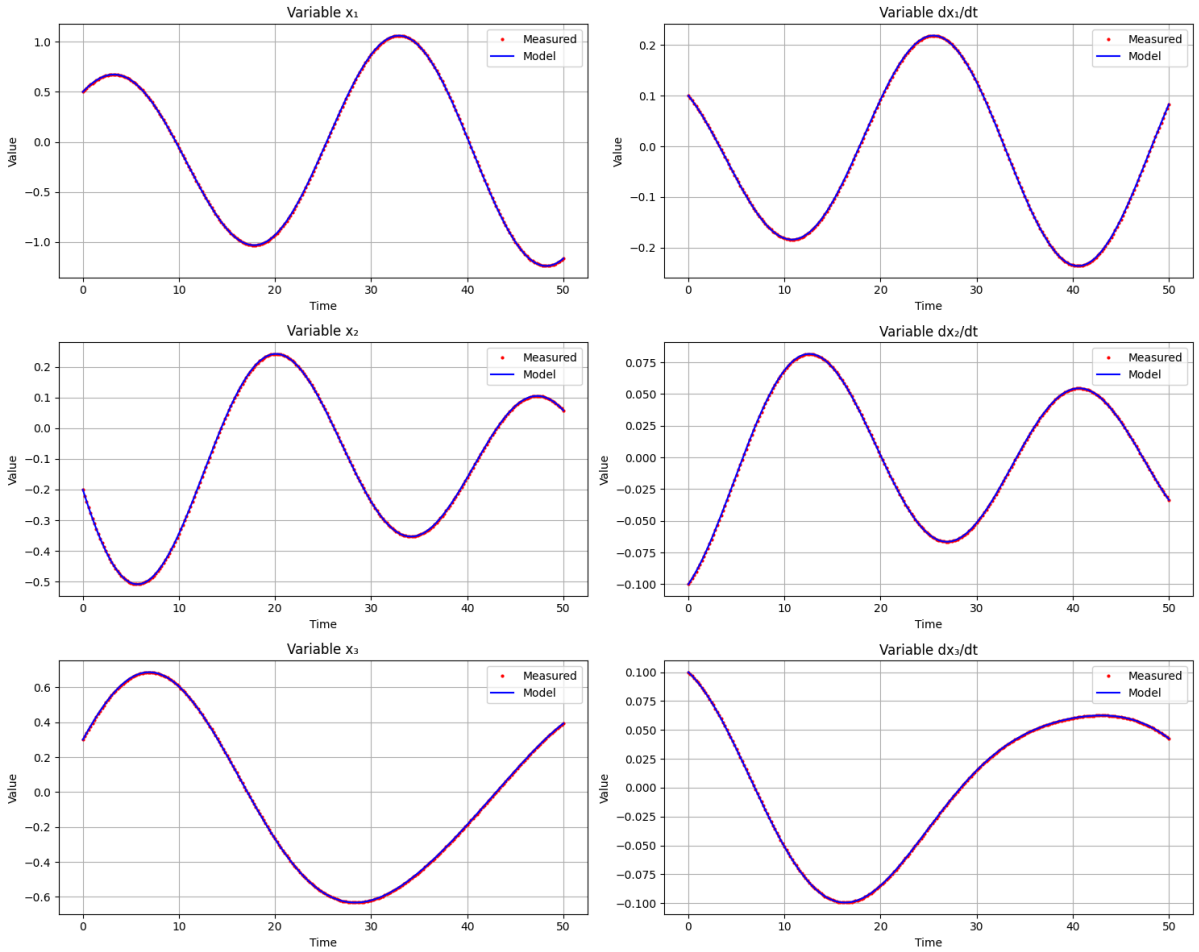
- алгоритму знадобилось 4 ітерації для знаходження правильних параметрів
- виконання алгоритму займає лише 0,12 секунд

Помилка апроксимації:

```
Accuracy at 0 iteration:  23.420611487905884
Accuracy at 1 iteration:  2.6269017676132513
Accuracy at 2 iteration:  0.01394421146177104
Accuracy at 3 iteration:  5.549129404122092e-07
```

Похибка моделі зменшувалась від 23.42 до $5.5 \cdot 10^{-7}$ протягом 4 ітерацій, що показує зростання точності апроксимації.

Аналіз графіків



На наведених графіках зображено порівняння системи для спостережуваних даних (Measured) та отриманих результатів (Model) з використанням ідентифікованих параметрів

- Зліва: графіки зміни координат $x_1,\ x_2,\ x_3$ у часі для відповідних мас $m_1,\ m_2,\ m_3$
- Справа: графіки зміни швидкостей мас $\frac{dx_1}{dt},\ \frac{dx_2}{dt},\ \frac{dx_3}{dt}$

Аналіз показує, що траєкторії моделі (Model) практично ідеально накладаються на точки вимірювань (Measured) на всьому інтервалі часу $t \in [0; 50]$. Відсутність видимих розбіжностей між кривими підтверджує високу точність апроксимації, що збігається з отриманим чисельним значенням помилки апроксимації ($5.5 \cdot 10^{-7}$). Це свідчить про те, що знайдені параметри m1, m2, m3 є коректними.