Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Лабораторна робота №3

з предмету «Моделювання систем»

Виконав студент 3-го курсу

Групи ІПС-31

Навка Гліб Олександрович

2021

Завдання

1. Знати означення функції чутливості і вивчити диференціальне рівняння, з якого шукається матриця чутливості. Записати рівняння чутливості для математичної моделі (3.1) - (3.3).

2. Створити програму, яка реалізує метод параметричної ідентифікації параметрів з використанням функцій чутливості для математичної моделі (3.1) - (3.3).

3. Вивести знайдені параметри.

4. Оформити в друкованій формі звіт про виконання роботи, в якому викласти результати проведених обчислень.

Теорія

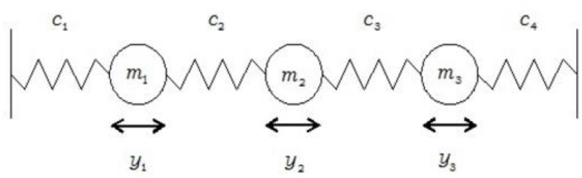

Рис. 3.1: Математична модель коливання трьох тіл

Математична модель 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 має вигляд

$$\frac{d^2y_1(t)}{dt^2} + \frac{(c_2+c_1)}{m_1}y_1(t) - \frac{c_2}{m_1}y_2(t) = f_1(t), \tag{3.1}$$

$$\frac{d^2y_2(t)}{dt^2} - \frac{c_2}{m_2}y_1(t) + \frac{(c_2+c_3)}{m_2}y_2(t) - \frac{c_3}{m_2}y_3(t) = f_2(t), \tag{3.2}$$

$$\frac{d^2y_3(t)}{dt^2} - \frac{c_3}{m_3}y_2(t) + \frac{(c_4+c_3)}{m_3}y_3(t) = f_3(t). \tag{3.3}$$

Означення 3.1. *Функцією чутливості системи* (3.4) *в точці* $p = \hat{p}$ *називається функція, яка задається співвідношенням* $U(t) = \frac{\partial x(t,\hat{p})}{\partial p}$.

Потрібно оцінити частину невідомих параметрів моделі (3.1) - (3.3) з використанням функції чутливості за відомими спостереженнями $\bar{y}(t)$ на часовому інтервалі $t \in [0, T]$. Для цього записуємо (3.1) - (3.3) у вигляді системи диференціальних рівнянь в нормальній формі розмірності 6.

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

Показник якості ідентифікації параметрів β має вигляд

$$I(\beta) = \int_{t_0}^{T} (\bar{y}(t) - y(t,\beta))^T (\bar{y}(t) - y(t,\beta))dt \to \min_{\beta}.$$

Числовий метод ітераційний і має вигляд

$$\beta_{k+1} = \beta_k + \Delta\beta.$$

Початкове наближення β_0 задається,

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

Матриці чутливості $U(t)$ визначається з матричного диференціального рівняння

$$\frac{dU(t)}{dt} = AU(t) + \frac{\partial(Ay)}{\partial\beta},\ U(t_0) = 0 \tag{3.7}$$

<u>Розв’язок</u>

Варіант №5

Файли **y5.txt**

Відомі параметри

$$c_2 = 0.3, c_3 = 0.2,\ m_1 = 12,\ m_3 = 18$$

$$\beta_0 = (0.1,\ 0.08,\ 21)^T$$

Ініціалізація відомих даних

```python
c2 = 0.3
c3 = 0.2
m1 = 12
m3 = 18

# b[0] = c1, b[1] = c4, b[2] = m2
b0 = np.array([0.1, 0.08, 21])
```

Зчитування даних з файлу

```python
measurements = []

with open('y5.txt') as file:
    for line in file.readlines():
        measurements.append(line.split())

measurements = np.array(measurements, float).T
```

Виклик функції, яка знаходить невідомі параметри

```python
b = calc_beta(b0, m1, m3, c2, c3, measurements)
```

Реалізація допоміжних функція для реалізації основних алгоритмів

Реалізація пошуку функції чутливості у вигляді диференціальних рівнянь в нормальній формі

```
def calc_sensitivity_matrix(m1, m2, m3, c1, c2, c3, c4):
    return np.array([
        [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]
    ])
```

Реалізація методу пошуку частинної похідної по c1

```
def calc_c1_derivative(m1, m2, m3, c1, c2, c3, c4):
    return np.array([
        [0,       0, 0, 0, 0, 0],
        [-1 / m1, 0, 0, 0, 0, 0],
        [0,       0, 0, 0, 0, 0],
        [0,       0, 0, 0, 0, 0],
        [0,       0, 0, 0, 0, 0],
        [0,       0, 0, 0, 0, 0]
    ])
```

Реалізація методу пошуку частинної похідної по c4

```
def calc_c4_derivative(m1, m2, m3, c1, c2, c3, c4):
    return np.array([
        [0, 0, 0, 0, 0,       0],
        [0, 0, 0, 0, 0,       0],
        [0, 0, 0, 0, 0,       0],
        [0, 0, 0, 0, 0,       0],
        [0, 0, 0, 0, 0,       0],
        [0, 0, 0, 0, -1 / m3, 0]
    ])
```

Реалізація методу пошуку частинної похідної по m2

```
def calc_m2_derivative(m1, m2, m3, c1, c2, c3, c4):
    return np.array([
        [0,                0, 0,                         0, 0,                0],
        [0,                0, 0,                         0, 0,                0],
        [0,                0, 0,                         0, 0,                0],
        [- c2 / (m2 ** 2), 0, (c2 + c3) / (m2 ** 2), 0, -c3 / (m2 ** 2), 0],
        [0,                0, 0,                         0, 0,                0],
        [0,                0, 0,                         0, 0,                0]
    ])
```

Реалізація методу пошуку похідної

```
def calc_model_derivatives(m1, m2, m3, c1, c2, c3, c4, y):
    c1_derivative = calc_c1_derivative(m1, m2, m3, c1, c2, c3, c4) @ y
    c4_derivative = calc_c4_derivative(m1, m2, m3, c1, c2, c3, c4) @ y
    m2_derivative = calc_m2_derivative(m1, m2, m3, c1, c2, c3, c4) @ y

    return np.array([c1_derivative, c4_derivative, m2_derivative]).T
```

Реалізація функції зовнішніх сил за допомогою функції чутливості

```
def f(m1, m2, m3, c1, c2, c3, c4, y):
    return calc_sensitivity_matrix(m1, m2, m3, c1, c2, c3, c4) @ y
```

Реалізація методу пошуку y за допомогою методу Рунге-Кутти

```
def model_runge_kutta(m1, m2, m3, c1, c2, c3, c4, measurements):
    y = np.zeros_like(measurements)
    y[0] = measurements[0].copy()

    for i in range(1, len(measurements)):
        k1 = STEP * f(m1, m2, m3, c1, c2, c3, c4, y[i - 1])
        k2 = STEP * f(m1, m2, m3, c1, c2, c3, c4, y[i - 1] + k1 / 2)
        k3 = STEP * f(m1, m2, m3, c1, c2, c3, c4, y[i - 1] + k2 / 2)
        k4 = STEP * f(m1, m2, m3, c1, c2, c3, c4, y[i - 1] + k3)

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

    return y
```

Реалізація методу пошуку матриці чутливості за допомогою методу Рунге-Кутти

```
def sensitivity_function_runge_kutta(m1, m2, m3, c1, c2, c3, c4, y,
measurement_count):
    u = np.zeros([measurement_count, 6, 3])

    a = calc_sensitivity_matrix(m1, m2, m3, c1, c2, c3, c4)
    beta_derivative = calc_model_derivatives(m1, m2, m3, c1, c2, c3, c4, y.T)

    for i in range(1, measurement_count):
        k1 = STEP * (a @ u[i - 1] + beta_derivative[i - 1])
        k2 = STEP * (a @ (u[i - 1] + k1 / 2) + beta_derivative[i - 1])
        k3 = STEP * (a @ (u[i - 1] + k2 / 2) + beta_derivative[i - 1])
        k4 = STEP * (a @ (u[i - 1] + k3) + beta_derivative[i - 1])

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

    return u
```

Реалізація методу пошуку Δb

```
def calc_delta(y, u, measurements):
    lhs = []
    rhs = []

    for i in range(u.shape[0]):
        lhs.append(u[i].T @ u[i] * STEP)
        rhs.append(u[i].T @ (measurements - y)[i] * STEP)

    lhs = np.linalg.inv(np.array(lhs).sum(0))
    rhs = np.array(rhs).sum(0)

    return lhs @ rhs
```

Реалізація основного методу пошуку вектора b

```
def calc_beta(b, m1, m3, c2, c3, measurements):
    measurement_count = len(measurements)

    c1 = b[0]
    c4 = b[1]
    m2 = b[2]

    while True:
        # Step 1 (Runge-Kutta for model)
        y = model_runge_kutta(m1, m2, m3, c1, c2, c3, c4, measurements)

        # Step 2 (Runge-Kutta for sensitivity function)
        u = sensitivity_function_runge_kutta(m1, m2, m3, c1, c2, c3, c4, y,
        measurement_count)

        # Step 3 (Finding delta of beta)
        delta = calc_delta(y, u, measurements)

        # Step 4
        c1 += delta[0]
        c4 += delta[1]
        m2 += delta[2]

        max_delta = np.abs(delta).max()

        print(f'max_delta: {max_delta}')

        # Step 5
        if np.abs(delta).max() < EPS:
            break

    return np.array([c1, c4, m2])
```

Детальніше про функцію

Зчитування невідомих параметрів з вектору b

```
c1 = b[0]
c4 = b[1]
m2 = b[2]
```

На першому кроці знаходимо y за допомогою методу Рунге-Кутти

```
# Step 1 (Runge-Kutta for model)
y = model_runge_kutta(m1, m2, m3, c1, c2, c3, c4, measurements)
```

На другому кроці знаходимо матрицю чутливості за допомогою методу Рунге-Кутти

```
# Step 2 (Runge-Kutta for sensitivity function)
u = sensitivity_function_runge_kutta(m1, m2, m3, c1, c2, c3, c4, y,
measurement_count)
```

На третьому кроці обраховуємо Δb

```
delta = calc_delta(y, u, measurements)
```

На четвертому кроці додаємо відповідні елементи вектору b до невідомих параметрів

```
c1 += delta[0]
c4 += delta[1]
m2 += delta[2]
```

На п’ятому кроці перевіряємо умову зупинки алгоритму

```
if np.abs(delta).max() < EPS:
    break
```

Загальний результат роботи програми

```
max_delta: 5.227607656413347
max_delta: 1.5996345119892912
max_delta: 0.17082080230968716
max_delta: 0.0019298780319628528
max_delta: 1.3706516180304294e-05
c1: 0.1400002324154925
c4: 0.12000004180938927
m2: 28.00000655526047
```

Відповідь

c1: 0.14

c4: 0.12

m2: 28.0