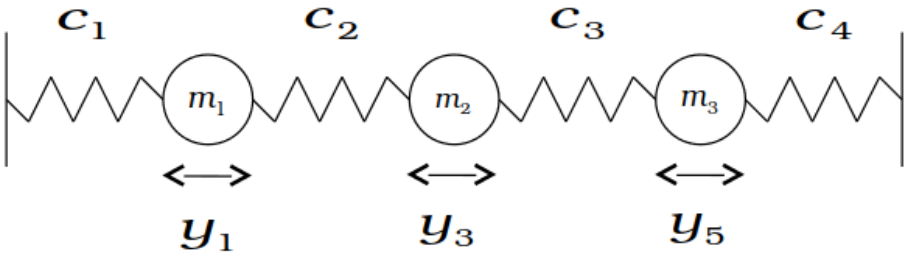
Звіт до Лабораторної роботи №2
Чиківчука Миколи, ІПС-31
Варіант 11

1. Завдання:

Параметрична ідентифікація параметрів з використанням функцій чутливості.

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.



11) Вектор оцінюваних параметрів $\beta = (c_3, m_1, m_3)^T$, початкове наближення $\beta_0 = (0.1,\ 10,\ 21)^T$, відомі параметри $c_1 = 0.14, c_2 = 0.3, c_4 = 0.12$, $m_2 = 28$, ім'я файлу з спостережуваними даними y1.txt.

2. Розв'язок:

```
import numpy as np

EPS = 0.0001
DT = 0.2

# calculates sensitivity matrix
def s_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]
        ])

# calculates model derivatives
def model_d(m1, m2, m3, c1, c2, c3, c4, y):
        c3d = np.array([
                [0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0],
```

```
            [0, 0, 0, 0, 0, 0],
            [0, 0, - 1 / m2, 0, 1 / m2, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 0, 1 / m3, 0, - 1 / m3, 0]
        ]) @ y

        m1d = np.array([
            [0, 0, 0, 0, 0, 0],
            [(c2 + c1) / m1 ** 2, 0, - c2 / m1 ** 2, 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]
        ]) @ y

        m3d = 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, - c3 / m3 ** 2, 0, (c4 + c3) / m3 ** 2, 0]
        ]) @ y

        return np.array([c3d, m1d, m3d]).T

# runge-kutta for model
def rk_model(m1, m2, m3, c1, c2, c3, c4, data):
        y = np.zeros_like(data)
        y[0] = data[0].copy()

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


# runge-kutta for sensitivity function
def rk_sfun(m1, m2, m3, c1, c2, c3, c4, y, data_size):
        u = np.zeros([data_size, 6, 3])
        a = s_matrix(m1, m2, m3, c1, c2, c3, c4)
        beta_derivative = model_d(m1, m2, m3, c1, c2, c3, c4, y.T)

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

```
            k4 = DT * (a @ (u[i - 1] + k3) + beta_derivative[i - 1])
            u[i] = u[i - 1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6

    return u


def find_delta(y, u, data):
    lhs = []
    rhs = []

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

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

    return lhs @ rhs


def find_beta(b, m2, c1, c2, c4, data):
    data_size = len(data)
    c3 = b[0]
    m1 = b[1]
    m3 = b[2]

    while True:
        y = rk_model(m1, m2, m3, c1, c2, c3, c4, data)
        u = rk_sfun(m1, m2, m3, c1, c2, c3, c4, y, data_size)

        delta = find_delta(y, u, data)
        c3 += delta[0]
        m1 += delta[1]
        m3 += delta[2]

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

    return np.array([c3, m1, m3])

# b = (c3, m1, m3)
b0 = np.array([0.1, 10, 21])

m2 = 28
c1 = 0.14
c2 = 0.3
c4 = 0.12

data = []
with open('y1.txt') as file:
for line in file.readlines():
```

```
data.append(line.split())

data = np.array(data, float).T

print(find_beta(b0, m2, c1, c2, c4, data))
```

3. Результат:

c3 = 0.1999999
m1 = 11.99999493
m3 = 17.99999187