Міністерство освіти і науки України
НТУУ «КПІ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Лабораторна робота №4

з дисципліни «Комп’ютерне моделювання»

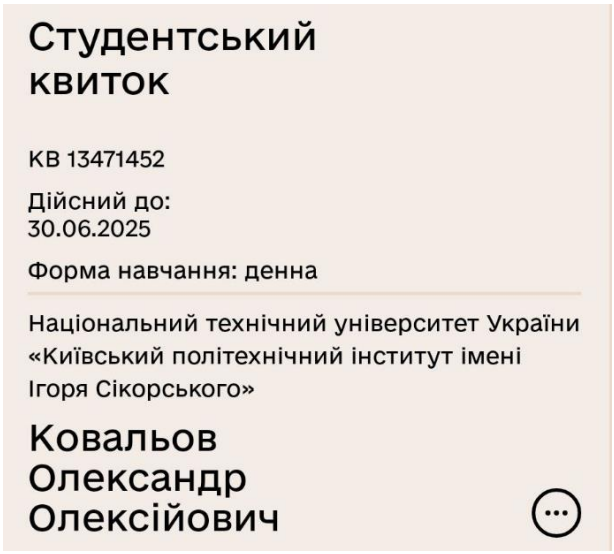
Тема «Параметрична ідентифікація моделей»

Варіант №22

Студента 3-го курсу НН ІАТЕ гр. ТР-12
Ковальова Олександра
Перевірив: д.т.н., проф. Шушура О. М.

КИЇВ 2023

Варіант. $g = 3$, $k = 6$ (де g – остання цифра у номері студентського квитка + 1, а k – передостання + 1).



Загальне завдання.

Розробити алгоритми та програмне забезпечення для розв'язку наведених задач. Алгоритми представити у вигляді блок-схем або діаграм діяльності UML. Програмне забезпечення розробити на будь-якій сучасній мові програмування.

Провести параметричну ідентифікацію моделі:

1. Сформувати індивідуальне завдання, підставивши в наведений нижче шаблон завдання параметри, визначені на основі даних номеру студентського квитка:

$$\begin{cases} \dfrac{dy_1}{dt} = -k\beta_1 y_1(t,x) - 2x\beta_2 y_2(t,x) + \cos(t) \\ \dfrac{dy_2}{dt} = -x\beta_1 y_1(t,x) - 4g\beta_2 y_2(t,x) + k * \sin(t) \end{cases}$$
$$t \in [0,1], \qquad y_1(0,x) = 0, \qquad y_2(0,x) = 0$$

   де g– остання цифра у номері студентського квитка + 1, k – передостання + 1;
   $y_1(t,x), y_2(t,x)$ – вихідні змінні моделі, як функції від часу та вхідної змінної;
   x – вхідна змінна моделі;
   β_1, β_2 – параметри моделі.
2. Виконати постановку задачі параметричної ідентифікації моделі, обравши квадратичний критерій похибки та градієнтний метод параметричної ідентифікації з застосуванням методу двох моделей.
3. Розробити алгоритм параметричної ідентифікації моделі.
4. Розробити на будь-якій сучасній мові програмування програмне забезпечення комп'ютерної моделі, що має:
   4.1. Меню з вибором варіанту дій.
   4.2. Можливість виконувати чисельний розрахунок моделі у вигляді таблично-заданих функцій для заданого значення аргументу та параметрів, візуалізовувати їх у вигляді графіків.
   4.3. Можливість здійснювати параметричну ідентифікацію моделі для заданих таблиць експериментальних даних, оцінювати точність ідентифікації.

Значення параметрів алгоритму ідентифікації має вводити користувач (точність, обмеження числа кроків й т.д.).

4.4. Можливість виводити для порівняння значення моделі та об’єкту з заданої таблиці експериментальних даних.

5. Пошук значень параметрів моделі здійснювати при порівнянні експериментальних даних та значень вихідних змінних моделі при $t = 1$, використовуючи наведені нижче дані:

№	x	y_1	y_2
1	$2k + g$	$k + 0.1g$	g
2	$2k + 0.9g$	$k + 0.2g$	$1.2g$
3	$2k + 1.1g$	$k - 0.1g$	$0.9g$
4	$2k + 0.95g$	$k + 0.15g$	$1.1g$

Хід роботи

Сформуємо індивідуальне завдання ($g = 3, k = 6$):

$$\begin{cases} \dfrac{dy_1}{dt} = -6\beta_1 y_1(t,x) - 2x\beta_2 y_2(t,x) + \cos(t) \\ \dfrac{dy_2}{dt} = -x\beta_1 y_1(t,x) - 12\beta_2 y_2(t,x) + 6\sin(t) \end{cases}$$

$$t \in [0,1], \qquad y_1(0,x) = 0, \qquad y_2(0,x) = 0$$

де $y_1(t,x), y_2(t,x)$ – вихідні змінні моделі, як функції від часу та вхідної змінної;
x – вхідна змінна моделі;
β_1, β_2 – параметри моделі.

Таблиця з даними:

№	x	y_1	y_2
1	15	6.3	3
2	14.7	6.6	3.6
3	15.3	5.7	2.7
4	14.85	6.45	3.3

Постановка задачі параметричної ідентифікації моделі в даному випадку включатиме в себе вибір квадратичного критерію похибки і використання градієнтного методу параметричної ідентифікації зі застосуванням методу двох моделей.

Якщо об'єкт не динамічний або порівняння проводиться в певний момент часу, то критерій похибки прийме вигляд:

$$E(\bar{\beta}) = \sum_{j=0}^{m}\sum_{i=0}^{n}\left(y_i(\bar{x}_j, \bar{\beta}) - y_{ij}^*\right)^2 \to \min_{\bar{\beta}}$$

Якщо $y_i(\bar{\beta})$ – лінійна функція, тоді маємо метод найменших квадратів.

Градієнтний метод є ітераційним методом, що базується на знаходженні градієнту критерію похибки по відношенню до параметрів моделі і зміни параметрів в

напрямку, який є протилежний градієнту. У даному випадку, будемо використовувати градієнтний метод параметричної ідентифікації з методом двух моделей. Цей метод передбачає використання двох моделей: оригінальної моделі та моделі зі зміненими параметрами, і порівняння критерію похибки для обох моделей.

Алгоритм параметричної ідентифікації наступний:

1. Знайти початковий вектор $\overline{\beta_0}$ параметрів.
2. Розрахунок нового вектору $\overline{\beta_k}$ за формулою:

$$\overline{\beta_{k+1}} = \overline{\beta_k} - \Gamma\overline{\nabla}E\left(\overline{\beta_k}\right),$$

де Г – крок градієнтного методу,

$\overline{\nabla}E\left(\overline{\beta_k}\right)$ – значення градієнту функціоналу похибки в точці $\overline{\beta_k}$.

Знак «-» – оскільки знаходимо мінімум критерію, то в якості напрямку використовуємо антиградієнт.

Розрахунок проводиться до тих пір, поки умова не буде виконана:

$$\| \overline{\beta_{k+1}} - \overline{\beta_k} \| < \varepsilon,$$

або $\| \overline{\nabla}E \| < \varepsilon$.

Розрахунок градієнта функціоналу похибки залежить від його виду. Нехай він має вигляд, як зазначено вище, для нашого випадку. Знайдемо частинні похідні:

$$\frac{\partial E}{\partial \beta_l} = 2 \cdot \sum_{j=1}^{m} \sum_{i=1}^{n} (y_i(\overline{x_j}, \overline{\beta}) - y_{ij}^*) \cdot \frac{\partial y_i}{\partial \beta_l}(\overline{\beta}, \overline{x_j})$$

Для розрахунку значення $\frac{\partial y_i}{\partial \beta_l}$ часто використовується метод двох моделей. Він полягає в тому, що вихідні змінні обчислюються для значень параметра β_l і $\beta_l + \Delta\beta_l$, де $\Delta\beta_l$ – деякий невеликий приріст цього параметра. Відповідно, за визначенням частинної похідної:

$$\frac{\partial y_i}{\partial \beta_l} \approx \frac{y_i(\beta_l + \Delta\beta_l, \overline{x_j}) - y_i(\beta_l, \overline{x_j})}{\Delta\beta_l}$$

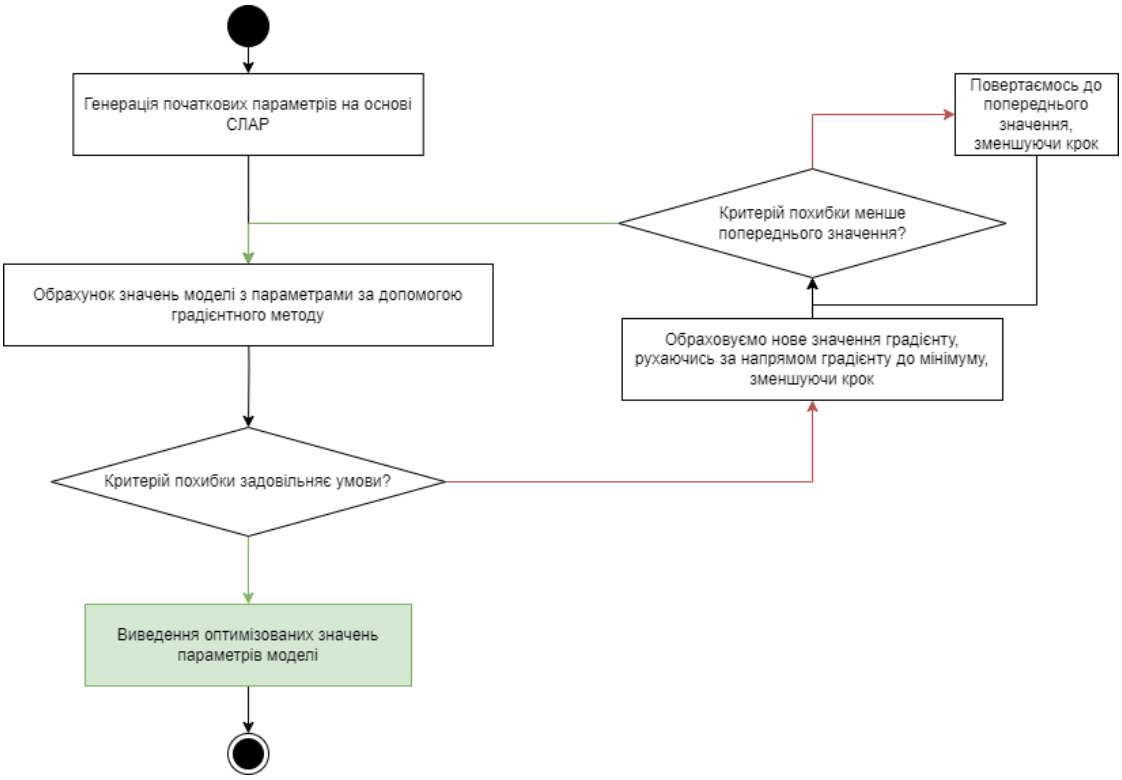
Вибір величини параметра кроку в градієнтному алгоритмі досить важливий. Відомі модифікації градієнтного алгоритму, у яких параметр кроку Г змінюється автоматично в процесі пошуку.

Найпростіший варіант – градієнтний метод з діленням кроку навпіл. Він полягає в тому, що якщо при пошуку мінімуму в новій точці виконується умова, то необхідно зменшити крок у двічі і знову спробувати знайти наступну точку. Так робити до тих пір, поки буде знайдена підходяща точка або виконається умова зупинки алгоритму.

На основі обраного методу створюється програмне забезпечення. Вона включає методи (процедуру) чисельного рішення моделі, блок початкової ідентифікації, рекурентну процедуру розрахунку параметрів, функцію обчислення градієнта. В результаті розрахунку будуть отримані значення невідомих коефіцієнтів. Ці значення повинні пройти перевірку на адекватність.

Спочатку оцінюється близькість значень вихідних змінних моделі до вихідних значень об'єкта, за яким проводилася ідентифікація. Якщо результати задовільні (досягається виконання фізичної постановки задачі ідентифікації), приступають до перевірки адекватності моделі на новому наборі статистичних даних відповідно до загальної методології математичного моделювання.

Блок-схема до програмного забезпечення:



Програмна реалізація:

Головне меню:

```
MENU | Lab 4 | Kovalov Alex
1. Calculate the initial parameters
2. Parametric identification | Gradient Descent method
3. Calculate model | Fixed x, parameters | Plot Output
4. Comparison | Model vs Experimental Data

9. Exit

INPUT:
```

Розрахування початкових параметрів на основі СЛАР:

```
INPUT: 1
Pair 1: [0.030320614184792163, -0.004039254271580375]
Pair 2: [0.02969037838159787, -0.0036581498181499 28]
Pair 3: [0.0326819322904137, -0.004082338704145822]
Pair 4: [0.0299927109062761, -0.003824151710140595]
Result: [0.030671408940769956, -0.0039009736260041798]
```

Процесс вирішення задачі параметричної ідентифікації:

```
INPUT: 2
B1: 0.030671408940769956
B2: -0.0039009736260041798
Max amount of steps: 10000
Precision: 0.001
1 | Error: 9999999.000000000 | B1: 0.03067 | B2: -0.003900974
2 | Error: 7.001253427 | B1: 0.03060 | B2: -0.004472628
3 | Error: 6.979150423 | B1: 0.03052 | B2: -0.005044357
4 | Error: 6.956964237 | B1: 0.03044 | B2: -0.005616167
5 | Error: 6.934693532 | B1: 0.03036 | B2: -0.006188038
6 | Error: 6.912339686 | B1: 0.03028 | B2: -0.006759966
7 | Error: 6.889902305 | B1: 0.03020 | B2: -0.007331945
8 | Error: 6.867381046 | B1: 0.03012 | B2: -0.007903971
9 | Error: 6.844775604 | B1: 0.03003 | B2: -0.008476037
10 | Error: 6.822085711 | B1: 0.02995 | B2: -0.009048138
11 | Error: 6.799311130 | B1: 0.02986 | B2: -0.009620268
12 | Error: 6.776451658 | B1: 0.02978 | B2: -0.010192421
```

Результат:

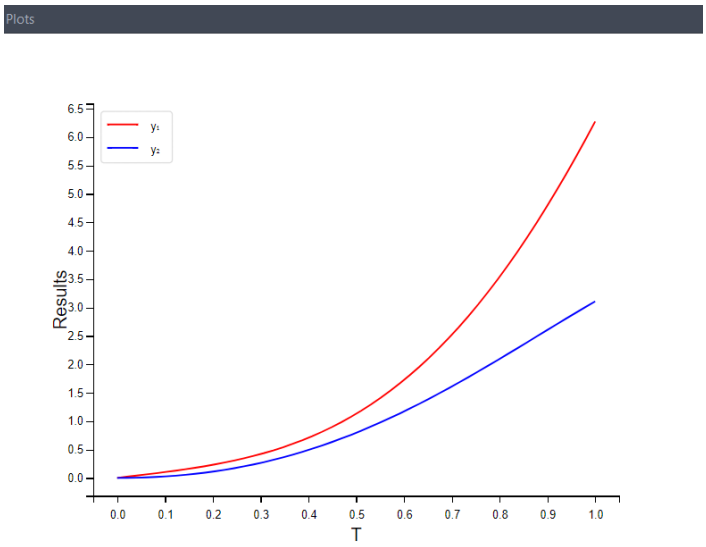
```
7983 | Error: 0.689787654 | B1: 0.07962 | B2: -0.196698844
7984 | Error: 0.689787654 | B1: 0.07962 | B2: -0.196699036
7985 | Error: 0.689787654 | B1: 0.07962 | B2: -0.196699228
7986 | Error: 0.689787654 | B1: 0.07962 | B2: -0.196699420
7986 | Error: 0.689787654 | B1: 0.07962 | B2: -0.196699228
Result: [0.07962114027889326, -0.1966992282515572]

MENU | Lab 4 | Kovalov Alex
1. Calculate the initial parameters
```

Побудова моделі за значеннями параметрів та обраним X:

```
INPUT: 3
X: 15
B1: 0.07962114027889326
B2: -0.1966992282515572
+---------+---------+---------+
|    T    |    Y1   |    Y2   |
+---------+---------+---------+
| 0.00000 | 0.00000 | 0.00000 |
| 0.02041 | 0.02035 | 0.00102 |
| 0.04082 | 0.04074 | 0.00414 |
| 0.06122 | 0.06142 | 0.00946 |
| 0.08163 | 0.08216 | 0.01707 |
```

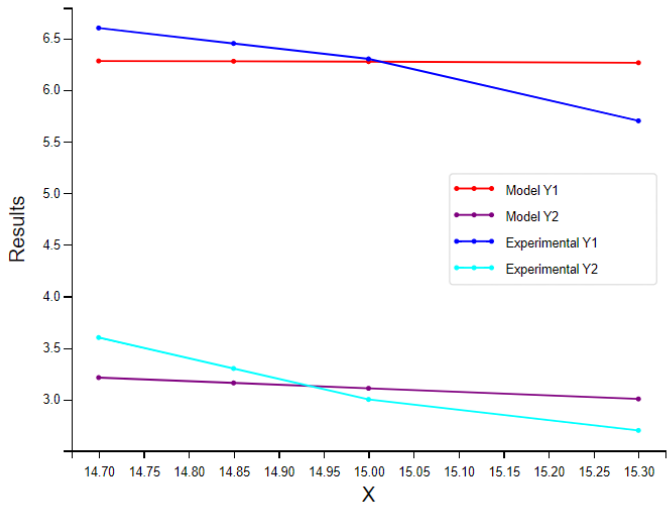
Графік:



Порівняння експериментальних значень та побудованої моделі:

```
INPUT: 4
B1: 0.07962114027889326
B2: -0.1966992282515572
+------------------------------+  +------------------------------+
|          Model Data          |  |      Experimental data       |
+----------+---------+---------+  +----------+---------+---------+
|    X     |    Y1   |    Y2   |  |    X     |    Y1   |    Y2   |
+----------+---------+---------+  +----------+---------+---------+
|    15    | 6.27387 | 3.10795 |  |    15    | 6.30000 |    3    |
| 14.70000 | 6.28022 | 3.21202 |  | 14.70000 | 6.60000 | 3.60000 |
| 15.30000 | 6.26308 | 3.00480 |  | 15.30000 | 5.70000 | 2.70000 |
| 14.85000 | 6.27761 | 3.15988 |  | 14.85000 | 6.45000 | 3.30000 |
+----------+---------+---------+  +----------+---------+---------+
```

Порівняльний графік:



Висновок: Під час виконання лабораторної роботи були набуті практичні навички для роботи з задачами параметричної ідентифікації. Був опрацьований метод градієнтного спуску, метод двох моделей. Було розроблено програмне забезпечення для моделювання системи за допомогою вищезазначених методів.

Програмний код

gradient_descent.py:

```
import model
import mse


def calculate_parameters(data, coefficients, init_parameters, max_steps,
precision):
    error = 9999999.0
    previous_error = error
    iteration = 1

    step = 0.000001

    gradient_params = []

    b1, b2 = init_parameters

    while True:
        if error < precision:
            print(f"*** Error < Precision ({error:.9f} < {precision:.9f}) ***)")
            break
        elif iteration > max_steps:
            print(f"*** Iteration limit is over! ({iteration} | {max_steps})
***")
            break

        print(f"{iteration} | Error: {error:0.9f} | B1: {b1:0.5f} | B2:
{b2:0.9f}")

        temp_previous_error = previous_error

        previous_error = error
        error = calc_error(coefficients, data, [b1, b2])

        if previous_error < error:
            error = previous_error
            previous_error = temp_previous_error

            b1 += step * gradient_params[0]
            b2 += step * gradient_params[1]

            step /= 2
            continue

        if previous_error == error:
            break

        gradient_params = solve_gradient(coefficients, data, [b1, b2])
        b1 -= step * gradient_params[0]
        b2 -= step * gradient_params[1]

        iteration += 1

    print(f"Result: {[b1, b2]}")
    return b1, b2
```

```
        def calc_error(coefficients, data, parameters):
            # Fixed value. Got from task
            t_span_fixed = [0, 1]

            # Arrays
            x_array, y1_observed, y2_observed = data

            y1_predicted = []
            y2_predicted = []

            for x in x_array:
                y1, y2 = model.runge_kutta_precised(x, t_span_fixed, parameters,
coefficients)
                y1_predicted.append(y1)
                y2_predicted.append(y2)

            error = mse.calculate(y1_predicted, y1_observed) +
mse.calculate(y2_predicted, y2_observed)
            return error


        def solve_gradient(coefficients, data, params):
            return [solve_gradient_1(coefficients, data, params),
                    solve_gradient_2(coefficients, data, params)]


        def solve_gradient_1(coefficients, data, params):
            e = 0
            iteration = 0

            # Fixed value. Got from task
            t_span_fixed = [0, 1]

            x, y1_exp, y2_exp = data
            for i in range(len(x)):
                y1, y2 = model.runge_kutta_precised(x[i], t_span_fixed, params,
coefficients)

                delta1 = calc_delta_1(coefficients, data, params, iteration)
                iteration += 1

                e += (y1 - y1_exp[i]) * delta1[0] + (y2 - y2_exp[i]) * delta1[1]

            return e


        def solve_gradient_2(coefficients, data, params):
            e = 0
            iteration = 0

            # Fixed value. Got from task
            t_span_fixed = [0, 1]

            x, y1_exp, y2_exp = data
            for i in range(len(x)):
                y1, y2 = model.runge_kutta_precised(x[i], t_span_fixed, params,
coefficients)

                delta2 = calc_delta_2(coefficients, data, params, iteration)
                iteration += 1

                e += (y1 - y1_exp[i]) * delta2[0] + (y2 - y2_exp[i]) * delta2[1]
```

```
            return e


        def calc_delta_1(coefficients, data, params, iteration):
            db = 0.000001

            # Fixed value. Got from task
            t_span_fixed = [0, 1]

            b1, b2 = params
            x, y1_exp, y2_exp = data
            y1, y2 = model.runge_kutta_precised(x[iteration], t_span_fixed, [b1 + db,
b2], coefficients)
            y11, y21 = model.runge_kutta_precised(x[iteration], t_span_fixed, [b1, b2],
coefficients)

            result = [(y1 - y11) / db, (y2 - y21) / db]

            return result


        def calc_delta_2(coefficients, data, params, iteration):
            db = 0.000001

            # Fixed value. Got from task
            t_span_fixed = [0, 1]

            b1, b2 = params
            x, y1_exp, y2_exp = data
            y1, y2 = model.runge_kutta_precised(x[iteration], t_span_fixed, [b1, b2 +
db], coefficients)
            y11, y21 = model.runge_kutta_precised(x[iteration], t_span_fixed, [b1, b2],
coefficients)

            result = [(y1 - y11) / db, (y2 - y21) / db]

            return result
```

model.py:

```
        import math

        import numpy as np
        from scipy.integrate import solve_ivp


        # Parameters: beta1, beta2, ...
        # Coefficients: G, K (Fixed constants for task. Given in task specification)
        # T Span - range of Ts (for example - [0, 1]
        def runge_kutta(x, t_span, parameters, coefficients):
            # y1, y2 = 0
            start_conditions = [0, 0]

            # Solve IVP - library method for solving system by Runge-Kutta method
            solution = solve_ivp(model, t_span, start_conditions, args=(x, parameters,
coefficients),
                                 method='RK45', t_eval=np.linspace(0, 1))

            # Extract solutions
            t = solution.t
            y1 = solution.y[0]
            y2 = solution.y[1]

            # Return arrays
```

```
            return [t, y1, y2]


        # Made for optimization purposes.
        # Returns last Y1 and Y2 values.
        def runge_kutta_precised(x, t_span, parameters, coefficients):
            # y1, y2 = 0
            start_conditions = [0, 0]

            # Solve IVP - library method for solving system by Runge-Kutta method
            solution = solve_ivp(model, t_span, start_conditions, args=(x, parameters,
coefficients),
                                 method='RK45', t_eval=np.linspace(0, 1))

            y1 = solution.y[0]
            y2 = solution.y[1]

            # Return arrays
            return [y1[-1], y2[-1]]


        # Model function calculates variables from system. So, it's system defining
function.
        # Variables - y1, y2
        # Parameters - beta1, beta 2
        # Coefficients (G, K) - Fixed constants for task. Given in task specification
        def model(t, variables, x, parameters, coefficients):
            y1, y2 = variables
            b1, b2 = parameters
            g, k = coefficients

            y1_derivative = -k * b1 * y1 - 2 * x * b2 * y2 + math.cos(t)
            y2_derivative = -x * b1 * y1 - 4 * g * b2 * y2 + k * math.sin(t)
            return [y1_derivative, y2_derivative]
```

initial_parameters.py:

```
        import numpy as np
        from scipy.integrate import odeint
        from scipy.optimize import fsolve


        # Initial system. Returns values of derivatives.
        # Used in "objective" method for solving this system.
        def model(y, t, b, x, coefficients):
            b1, b2 = b
            g, k = coefficients
            y1, y2 = y

            dy1dt = -k * b1 * y1 - 2 * x * b2 * y2 + np.cos(t)
            dy2dt = -x * b1 * y1 - 4 * g * b2 * y2 + k * np.sin(t)
            return [dy1dt, dy2dt]


        # Calculates difference between solved Y values and Y values from experiment.
        # Used in "calculate" method.
        def objective(b, t, variables, coefficients):
            t_span = np.array([0, t])  # Time span from 0 to 1
            x, y1, y2 = variables

            y_pred = odeint(model, [y1, y2], t_span, args=(b, x, coefficients))
            y_actual = np.array([y1, y2])  # Given values of y1 and y2
            return y_pred[-1] - y_actual
```

```
        # T - fixed number.
        # Data - arrays of experimental data from task (X, Y1, Y2).
        # fsolve function minimize differences between Y. Seeks for Bs that do it.
        def calculate(t, arrays, coefficients):
            x, y1, y2 = arrays

            # Arrays lengths must be equal
            rows = len(arrays[0])
            for array in arrays:
                if len(array) != rows:
                    raise ValueError("There are different amount of rows in arrays")

            # Initial guess for Beta1 and Beta2
            initial_guess = np.array([1, 1])
            result = [0, 0]

            # Finding Betas from every experiment
            for i in range(0, rows):
                row = [x[i], y1[i], y2[i]]

                # Getting betas
                b_solution = fsolve(objective, initial_guess, args=(t, row,
coefficients))

                # Result - array of average betas
                for j in range(len(b_solution)):
                    result[j] += b_solution[j] / rows

                # Printing pairs
                print(f"Pair {i + 1}: [{b_solution[0]}, {b_solution[1]}]")

            # Printing results
            print(f"Result: {result}")
            return result
```