Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №5**

«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план)»

**Виконав:**

студент II курсу ФІОТ

групи ІВ-91

Бурбело Сергій

**Перевірив:**

Регіда П.Г.

Київ – 2021

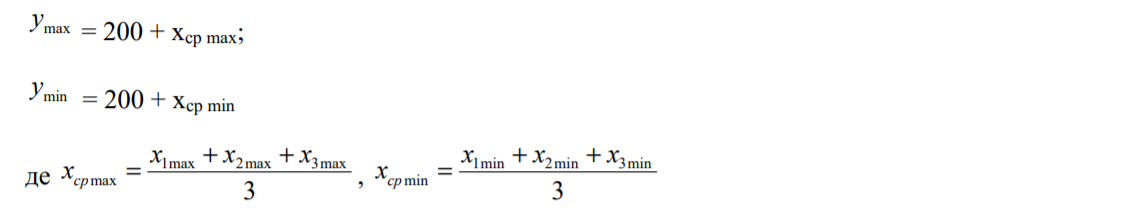
**Мета роботи:** Провести трьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

**Завдання на лабораторну роботу:**

1. Взяти рівняння з урахуванням квадратичних членів.

2. Скласти матрицю планування для ОЦКП

3. Провести експеримент у всіх точках факторного простору (знайти значення функції відгуку Y). Значення функції відгуку знайти у відповідності з варіантом діапазону, зазначеного далі.



4. Розрахувати коефіцієнти рівняння регресії і записати його.

5. Провести 3 статистичні перевірки.

**Варіант завдання:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Варіант | X1 | | X2 | | X3 | |
| min | max | min | max | min | max |
| 104 | -8 | 1 | -8 | 5 | -2 | 7 |

Довірча ймовірність дорівнює 0.95, а рівень значимості q = 0.05.

**Роздруківка тексту програми:**

from sklearn import linear\_model

from functools import partial

from scipy.stats import f, t

import math, os, sys

from pyDOE2 import \*

import numpy as np

import random

import math

class Experiment:

    def \_\_init\_\_(self, n, m):

        self.n = n

        self.m = m

        self.f1 = self.m - 1

        self.f2 = self.n

        self.f3 = self.f1\*self.f2

        self.p = 0.95

        self.q = 1 - self.p

        self.N = [i+1 for i in range(self.n+1)]

        self.ranges = ((-8, 1), (-8, 5), (-2, 7))

        self.av\_x\_max = sum([x[1] for x in self.ranges]) / 3

        self.av\_x\_min = sum([x[0] for x in self.ranges]) / 3

        self.min\_y = 200+int(self.av\_x\_min)

        self.max\_y = 200+int(self.av\_x\_max)

        self.x, self.y, self.x\_norm = self.matrix(self.min\_y, self.max\_y, self.ranges, self.n, self.m)

        self.av\_y = [round(sum(i) / len(i), 3) for i in self.y]

        self.b = self.koef(self.x, self.av\_y)

        self.Result(self.x\_norm, self.y, self.b, self.n, self.m, self.f1, self.f2, self.f3, self.q)

    def koef(self, X, Y, norm=False):

        skm = linear\_model.LinearRegression(fit\_intercept=False)

        skm.fit(X, Y)

        B = skm.coef\_

        if norm == 1:

            print('\nКоефіцієнти рівняння регресії з нормованими X:')

        else:

            print('\nКоефіцієнти рівняння регресії:')

        B = [round(i, 3) for i in B]

        print(B)

        print('\nРезультат рівняння зі знайденими коефіцієнтами:\n', np.dot(X, B))

        return B

    def s\_kv(self, y, y\_aver, n, m):

        res = []

        for i in range(n):

            s = sum([(y\_aver[i] - y[i][j]) \*\* 2 for j in range(m)]) / m

            res.append(round(s, 3))

        return res

    def regression(self, x, b):

            y = sum([x[i] \* b[i] for i in range(len(x))])

            return y

    def add\_sq\_nums(self, x):

            for i in range(len(x)):

                x[i][4] = x[i][1] \* x[i][2]

                x[i][5] = x[i][1] \* x[i][3]

                x[i][6] = x[i][2] \* x[i][3]

                x[i][7] = x[i][1] \* x[i][3] \* x[i][2]

                x[i][8] = x[i][1] \*\* 2

                x[i][9] = x[i][2] \*\* 2

                x[i][10] = x[i][3] \*\* 2

            return x

    def matrix(self, y\_min, y\_max, x\_range, n, m):

        print(f'\nМатриця планування для n = {n}, m = {m}')

        y = np.zeros(shape=(n, m))

        for i in range(n):

            for j in range(m):

                y[i][j] = random.randint(y\_min, y\_max)

        if n > 14:

            no = n - 14

        else:

            no = 1

        x\_norm = ccdesign(3, center=(0, no))

        x\_norm = np.insert(x\_norm, 0, 1, axis=1)

        for i in range(4, 11):

            x\_norm = np.insert(x\_norm, i, 0, axis=1)

        l = 1.215

        for i in range(len(x\_norm)):

            for j in range(len(x\_norm[i])):

                if x\_norm[i][j] < -1 or x\_norm[i][j] > 1:

                    if x\_norm[i][j] < 0:

                        x\_norm[i][j] = -l

                    else:

                        x\_norm[i][j] = l

        x\_norm = self.add\_sq\_nums(x\_norm)

        x = np.ones(shape=(len(x\_norm), len(x\_norm[0])), dtype=np.int64)

        for i in range(8):

            for j in range(1, 4):

                if x\_norm[i][j] == -1:

                    x[i][j] = x\_range[j - 1][0]

                else:

                    x[i][j] = x\_range[j - 1][1]

        for i in range(8, len(x)):

            for j in range(1, 3):

                x[i][j] = (x\_range[j - 1][0] + x\_range[j - 1][1]) / 2

        dx = [x\_range[i][1] - (x\_range[i][0] + x\_range[i][1]) / 2 for i in range(3)]

        x[8][1] = l \* dx[0] + x[9][1]

        x[9][1] = -l \* dx[0] + x[9][1]

        x[10][2] = l \* dx[1] + x[9][2]

        x[11][2] = -l \* dx[1] + x[9][2]

        x[12][3] = l \* dx[2] + x[9][3]

        x[13][3] = -l \* dx[2] + x[9][3]

        x = self.add\_sq\_nums(x)

        print('\nX:\n', x)

        print('\nX нормоване:\n')

        for i in x\_norm:

            print([round(x, 2) for x in i])

        print('\nY:\n', y)

        return x, y, x\_norm

    # -------------------------------------------------------

    # Перевірка однорідності дисперсії за критерієм Кохрена:

    # -------------------------------------------------------

    def kohrenCriteriy(self, y, y\_aver, n, m, f1, f2, q):

        S\_kv = self.s\_kv(y, y\_aver, n, m)

        Gp = max(S\_kv) / sum(S\_kv)

        print('\nПеревірка за критерієм Кохрена')

        return Gp

    def kohren(self, f1, f2, q=0.05):

        q1 = q / f1

        fisher\_value = f.ppf(q=1 - q1, dfn=f2, dfd=(f1 - 1) \* f2)

        return fisher\_value / (fisher\_value + f1 - 1)

    def Betas(self, x, y\_aver, n):

        res = [sum(1 \* y for y in y\_aver) / n]

        for i in range(len(x[0])):

            b = sum(j[0] \* j[1] for j in zip(x[:, i], y\_aver)) / n

            res.append(b)

        return res

    # -------------------------------------------------------

    # Перевірка однорідності дисперсії за критерієм Стьюдента:

    # -------------------------------------------------------

    def studentCriteriy(self, x, y, y\_aver, n, m):

        S\_kv = self.s\_kv(y, y\_aver, n, m)

        s\_kv\_aver = sum(S\_kv) / n

        s\_Bs = (s\_kv\_aver / n / m) \*\* 0.5

        Bs = self.Betas(x, y\_aver, n)

        ts = [round(abs(B) / s\_Bs, 3) for B in Bs]

        return ts

    # -------------------------------------------------------

    # Перевірка однорідності дисперсії за критерієм Фішера:

    # -------------------------------------------------------

    def fisherCriteriy(self, y, y\_aver, y\_new, n, m, d):

        S\_ad = m / (n - d) \* sum([(y\_new[i] - y\_aver[i]) \*\* 2 for i in range(len(y))])

        S\_kv = self.s\_kv(y, y\_aver, n, m)

        S\_kv\_aver = sum(S\_kv) / n

        return S\_ad / S\_kv\_aver

    # +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    # Вивід даних:

    # +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    def Result(self, X, Y, B, n, m, f1, f2, f3, q):

        print("\nРівняння регресії з урахуванням квадратичних членів:")

        print("ŷ = b0 + b1\*x1 + b2\*x2 + b3\*x3 + b12\*x1\*x2 + b13\*x1\*x3 + b23\*x2\*x3 + b123\*x1\*x2\*x3 + b11x1^2 + b22x2^2 + b33x3^2\n")

        student = partial(t.ppf, q=1 - q)

        t\_student = student(df=f3)

        G\_kr = self.kohren(f1, f2)

        y\_aver = [round(sum(i) / len(i), 3) for i in Y]

        print('\nСереднє значення y:', y\_aver)

        disp = self.s\_kv(Y, y\_aver, n, m)

        print('Дисперсія y:', disp)

        Gp = self.kohrenCriteriy(Y, y\_aver, n, m, f1, f2, q)

        print(f'Gp = {Gp}')

        if Gp < G\_kr:

            print(f'З ймовірністю {1 - q} дисперсії однорідні.')

        else:

            print("Необхідно збільшити кількість дослідів")

            m += 1

            Experiment(n, m)

        ts = self.studentCriteriy(X[:, 1:], Y, y\_aver, n, m)

        print('\nКритерій Стьюдента:\n', ts)

        res = [t for t in ts if t > t\_student]

        final\_k = [B[i] for i in range(len(ts)) if ts[i] in res]

        print('\nКоефіцієнти {} статистично незначущі, тому ми виключаємо їх з рівняння.'.format(

            [round(i, 3) for i in B if i not in final\_k]))

        y\_new = []

        for j in range(n):

            y\_new.append(self.regression([X[j][i] for i in range(len(ts)) if ts[i] in res], final\_k))

        print(f'\nЗначення "y" з коефіцієнтами {final\_k}')

        print(y\_new)

        d = len(res)

        if d >= n:

            print('\nF4 <= 0')

            print('')

            return

        f4 = n - d

        F\_p = self.fisherCriteriy(Y, y\_aver, y\_new, n, m, d)

        fisher = partial(f.ppf, q=0.95)

        f\_t = fisher(dfn=f4, dfd=f3)

        print('\nПеревірка адекватності за критерієм Фішера')

        print('Fp =', F\_p)

        print('F\_t =', f\_t)

        if F\_p < f\_t:

            print('Математична модель адекватна експериментальним даним')

        else:

            print('Математична модель не адекватна експериментальним даним')

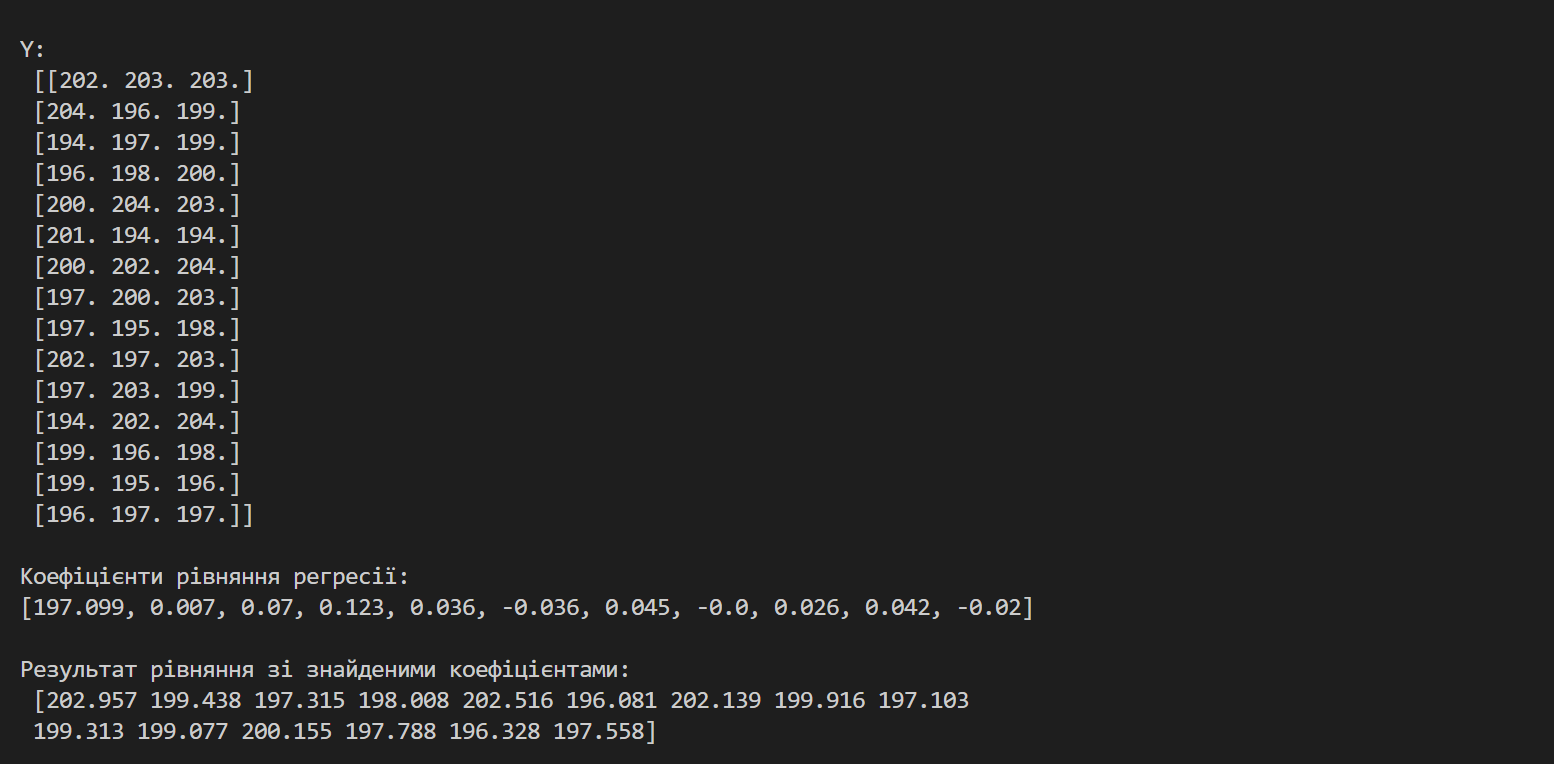
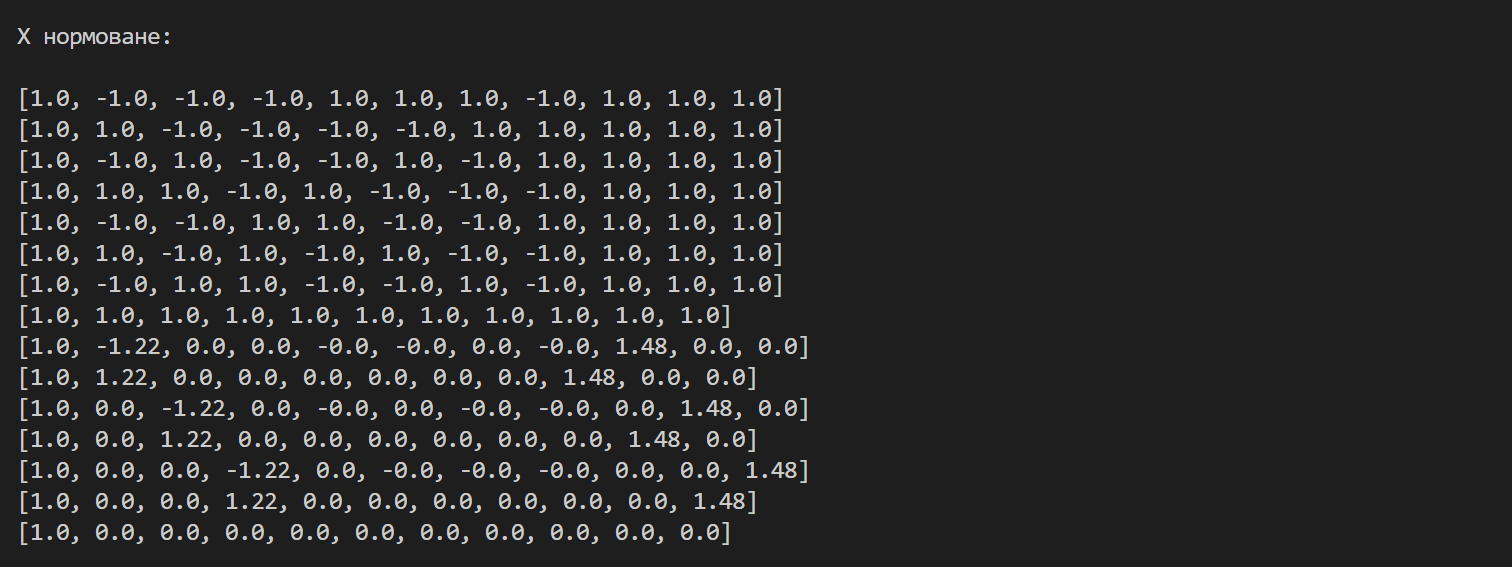
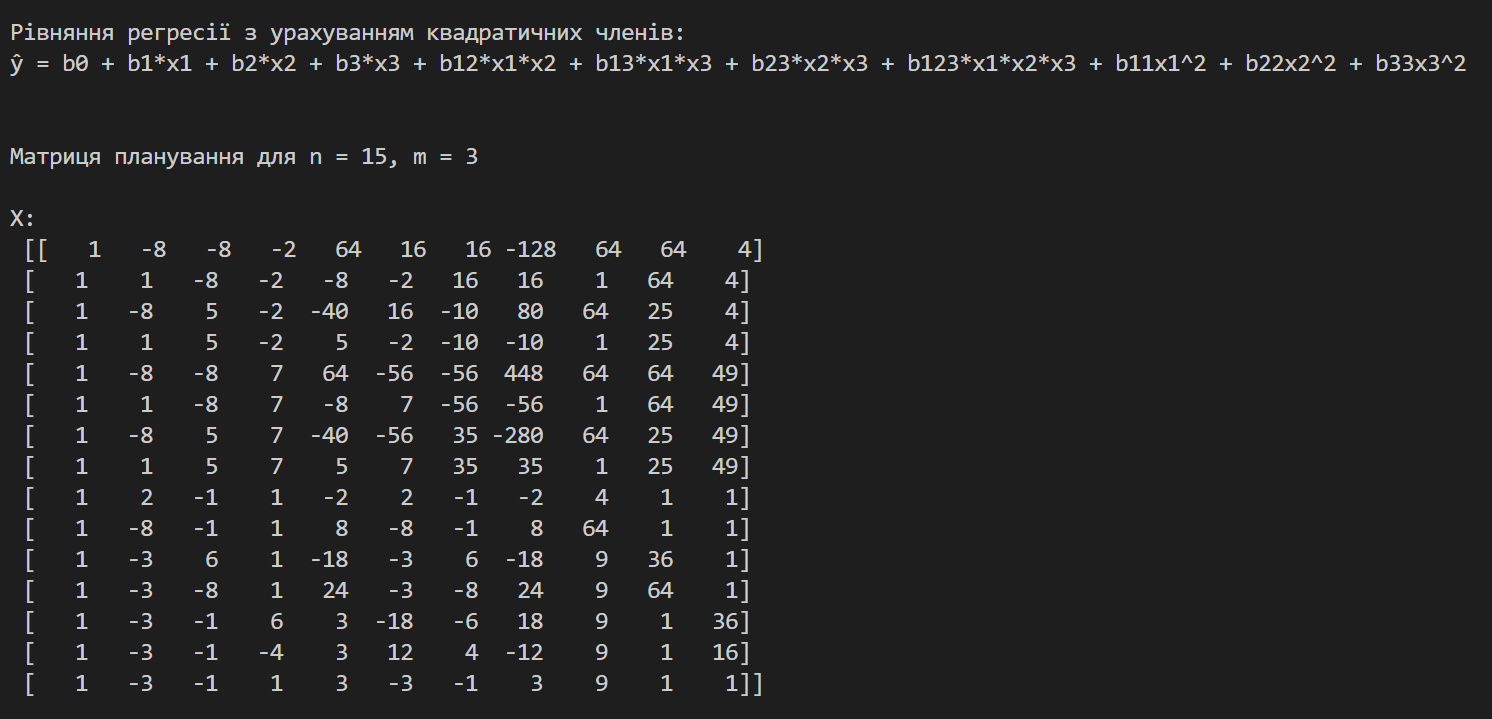
            m += 1

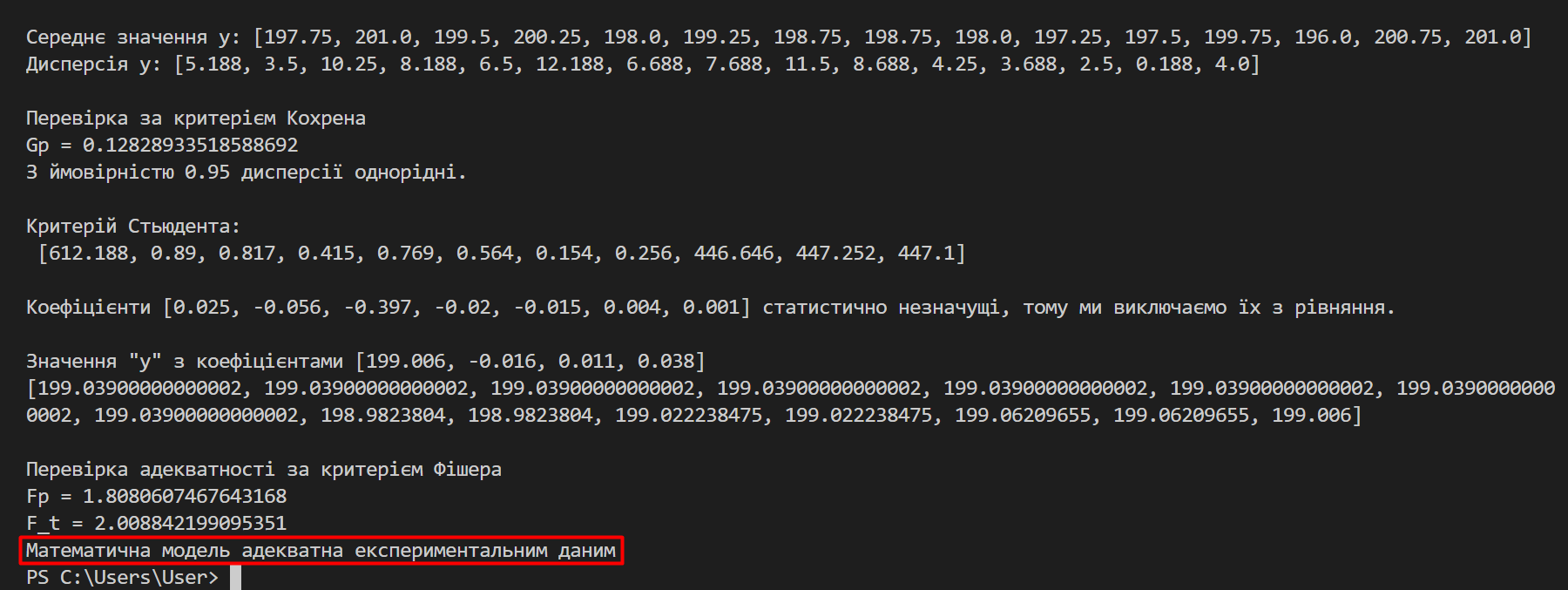
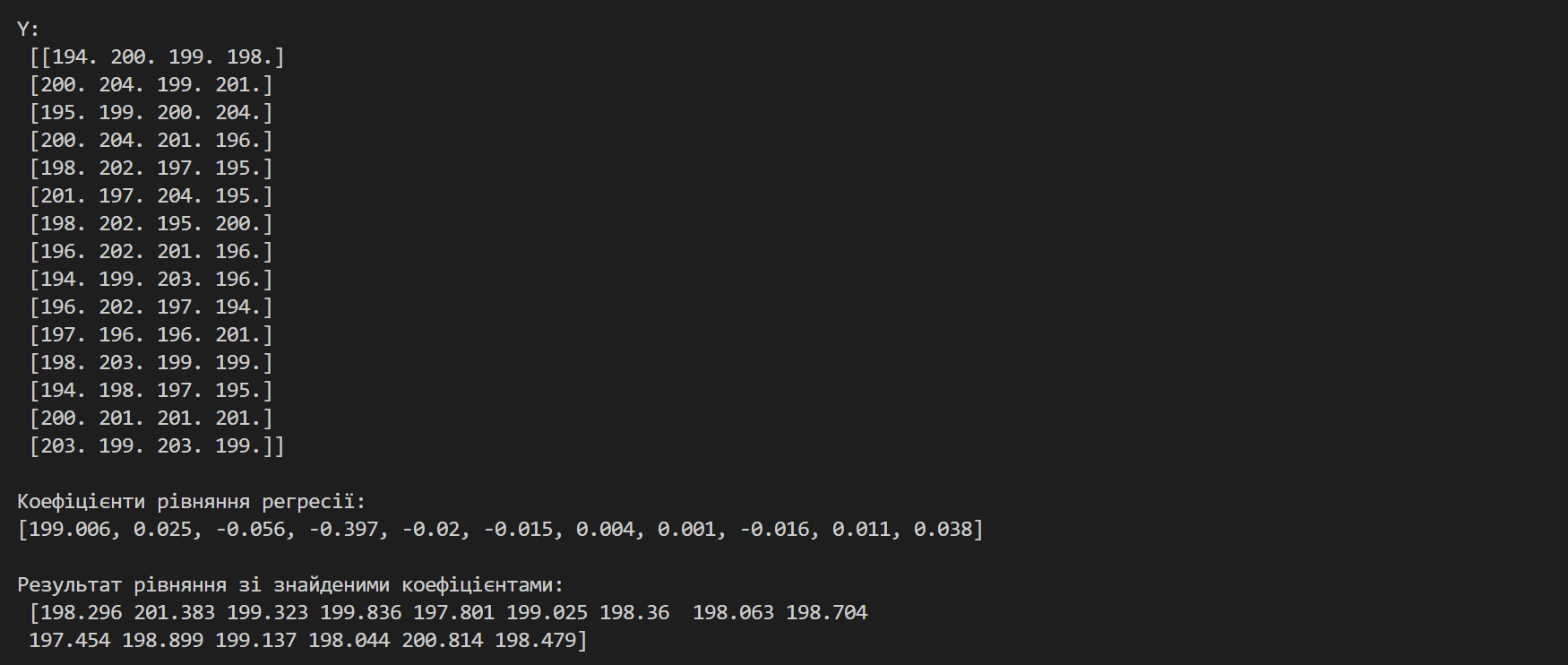
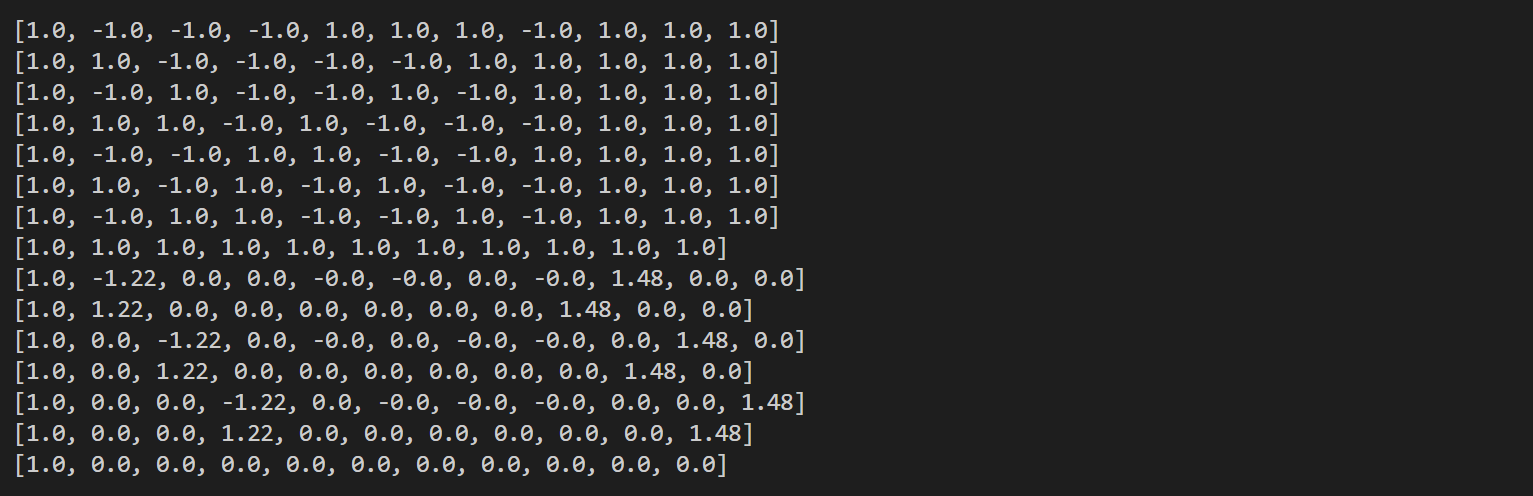
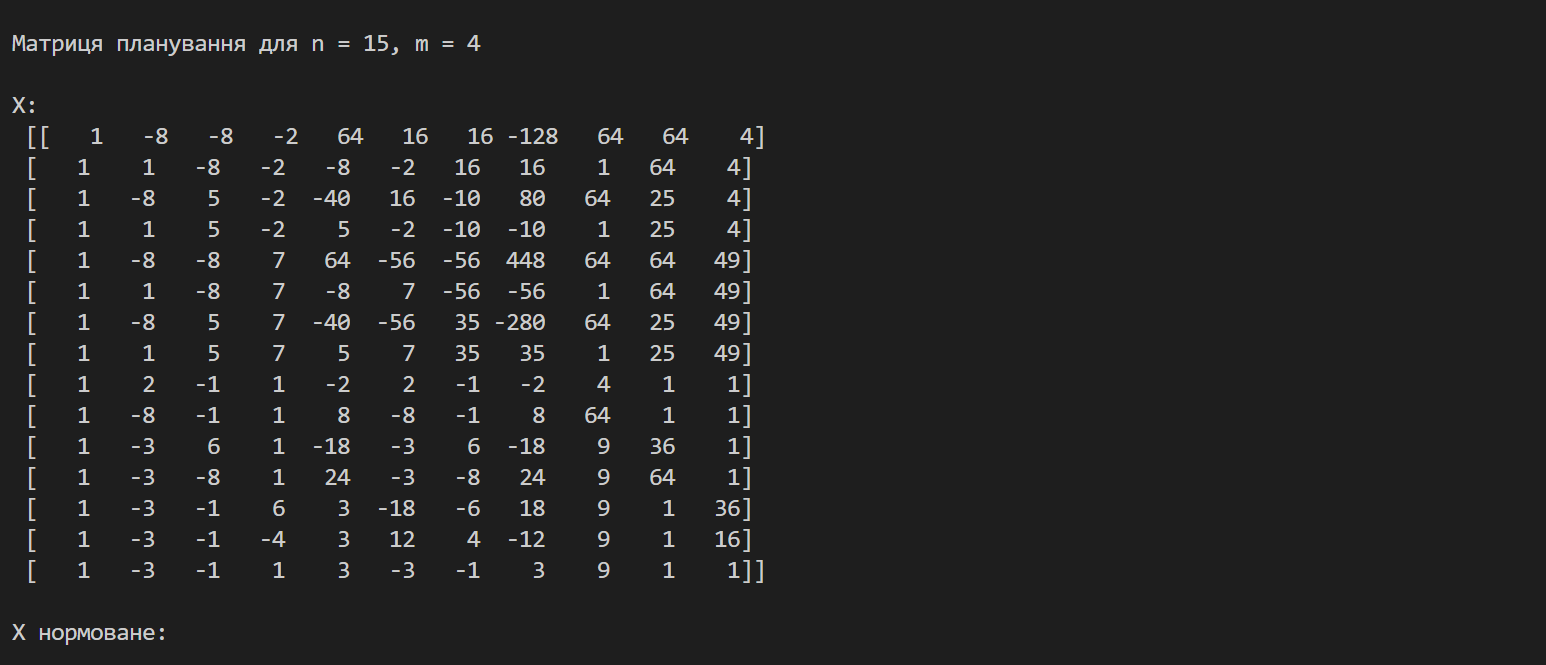
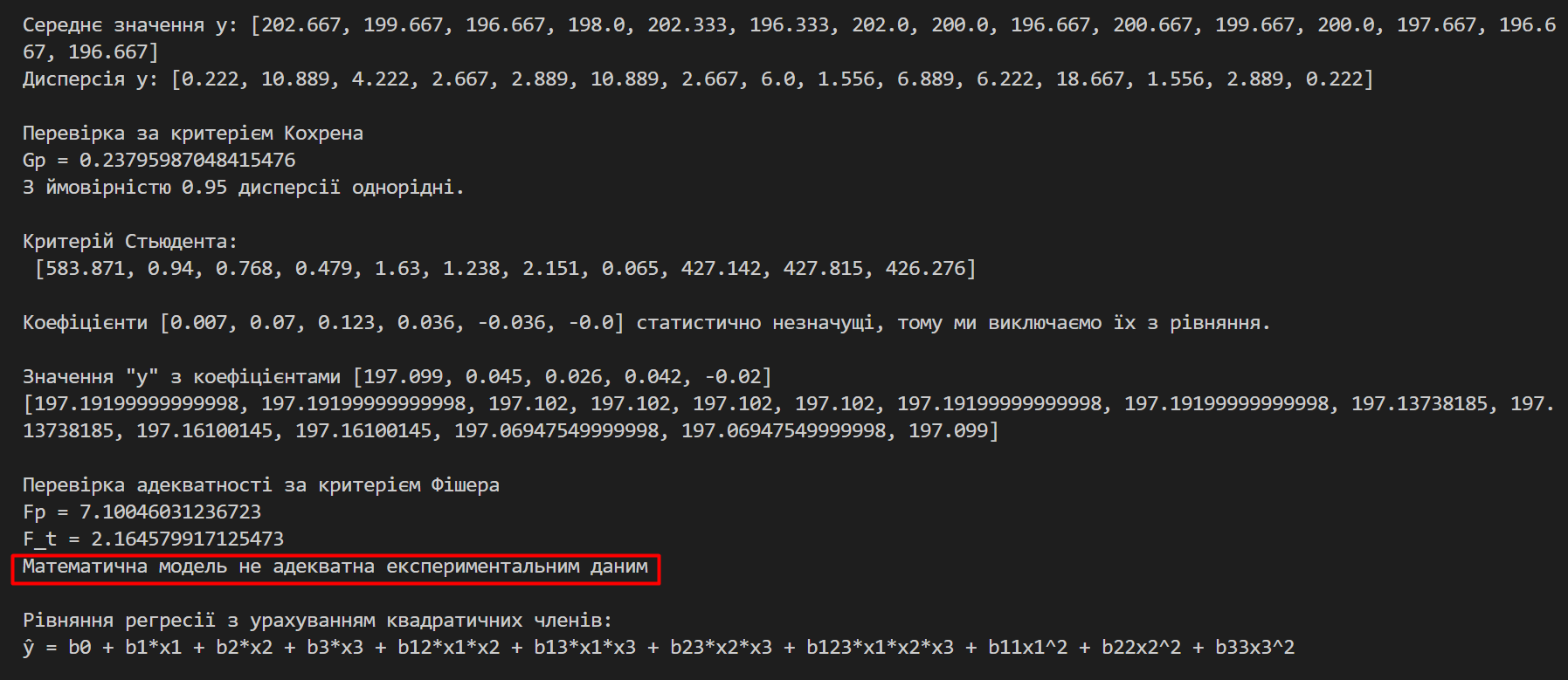
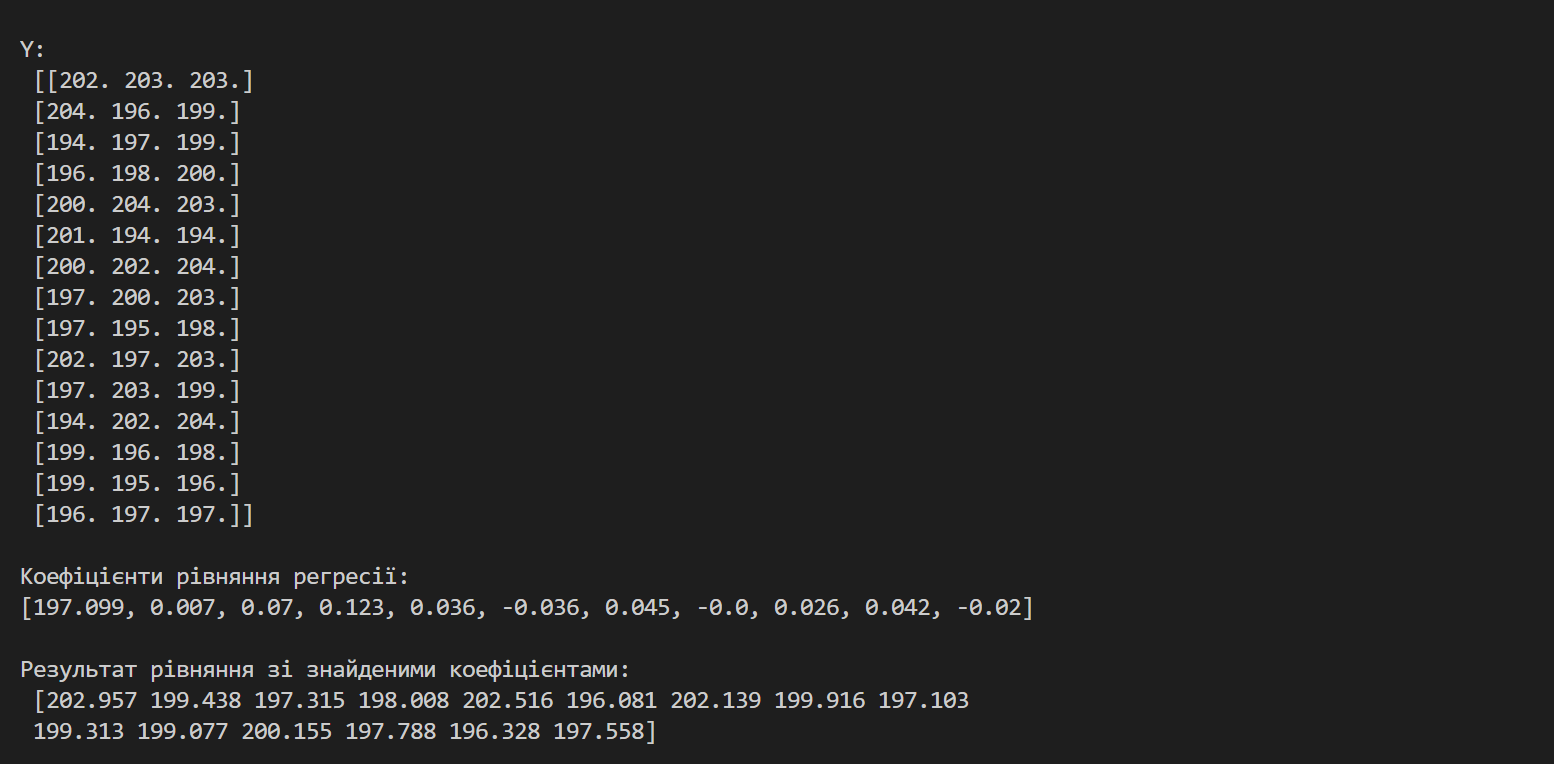
            Experiment(n, m)

if \_\_name\_\_ == '\_\_main\_\_':

    Experiment(15, 3)

**Результати роботи програми:**





**Висновок:**

У ході лабораторної роботи я змоделював ттрьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральний ортогональний композиційний план та знайшов рівняння регресії, яке адекватне для опису об'єкту.