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 №6**

«Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

**Виконав:**

студент II курсу ФІОТ

групи ІВ-91

Бурбело Сергій

**Перевірив:**

Регіда П.Г.

Київ – 2021

**Мета роботи:** Провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.

**Завдання на лабораторну роботу:**

1. Ознайомитися з теоретичними відомостями.

2. Вибрати з таблиці варіантів і записати в протокол інтервали значень х1, х2, х3. Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1; +*l* ; -*l* ; 0 для , , .

3. Значення функції відгуку знайти за допомогою підстановки в формулу:

yi = f(х1, х2, х3) + random(10)-5,

де f(х1, х2, х3) вибирається по номеру в списку в журналі викладача.

4. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках використовувати натуральні значення факторів.

5. Зробити висновки по виконаній роботі.

**Варіант завдання:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Варіант | X1 | | X2 | | X3 | |
| min | max | min | max | min | max |
| 104 | 15 | 25 | -25 | 10 | 45 | 50 |
| f(х1, х2, х3) | 8,7 + 4,3\*x1 + 1,2\*x2 + 2,2\*x3 + 0,4\*x1\*x1 + 1,0\*x2\*x2 + 6,4\*x3\*x3 + 1,1\*x1\*x2 + 0,1\*x1\*x3 + 9,2\*x2\*x3 + 1,2\*x1\*x2\*x3 | | | | | |

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

**Роздруківка тексту програми:**

from prettytable import PrettyTable

from math import fabs, sqrt

from random import randint

from numpy.linalg import solve

from \_pydecimal import Decimal

from scipy.stats import f, t

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.Kohren = False

        self.Fisher = False

        self.x1\_min = -20

        self.x1\_max = 30

        self.x2\_min = -25

        self.x2\_max = 10

        self.x3\_min = -25

        self.x3\_max = -20

        self.x01 = (self.x1\_max + self.x1\_min) / 2

        self.x02 = (self.x2\_max + self.x2\_min) / 2

        self.x03 = (self.x3\_max + self.x3\_min) / 2

        self.delta\_x1 = self.x1\_max - self.x01

        self.delta\_x2 = self.x2\_max - self.x02

        self.delta\_x3 = self.x3\_max - self.x03

        self.matrix\_pfe = [[-1, -1, -1, +1, +1, +1, -1, +1, +1, +1],

                            [-1, -1, +1, +1, -1, -1, +1, +1, +1, +1],

                            [-1, +1, -1, -1, +1, -1, +1, +1, +1, +1],

                            [-1, +1, +1, -1, -1, +1, -1, +1, +1, +1],

                            [+1, -1, -1, -1, -1, +1, +1, +1, +1, +1],

                            [+1, -1, +1, -1, +1, -1, -1, +1, +1, +1],

                            [+1, +1, -1, +1, -1, -1, -1, +1, +1, +1],

                            [+1, +1, +1, +1, +1, +1, +1, +1, +1, +1],

                            [-1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],

                            [+1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],

                            [0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],

                            [0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],

                            [0, 0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929],

                            [0, 0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929],

                            [0, 0,   0,   0, 0, 0, 0, 0, 0,   0   ]]

        self.matrix\_x = [[] for x in range(self.n)]

        for i in range(len(self.matrix\_x)):

            if i < 8:

                self.x\_1 = self.x1\_min if self.matrix\_pfe[i][0] == -1 else self.x1\_max

                self.x\_2 = self.x2\_min if self.matrix\_pfe[i][1] == -1 else self.x2\_max

                self.x\_3 = self.x3\_min if self.matrix\_pfe[i][2] == -1 else self.x3\_max

            else:

                self.x\_lst = self.x(self.matrix\_pfe[i][0], self.matrix\_pfe[i][1], self.matrix\_pfe[i][2])

                self.x\_1, self.x\_2, self.x\_3 = self.x\_lst

            self.matrix\_x[i] = [self.x\_1, self.x\_2, self.x\_3, self.x\_1 \* self.x\_2, self.x\_1 \* self.x\_3, self.x\_2 \* \

                 self.x\_3, self.x\_1 \* self.x\_2 \* self.x\_3, self.x\_1 \*\* 2, self.x\_2 \*\* 2, self.x\_3 \*\* 2]

        self.Result(self.n, self.m, self.f1, self.f2, self.f3, self.q, self.matrix\_x)

    def generate\_matrix(self, matrix\_x):

        def f(X1, X2, X3):

            f = 8.7 + 4.3 \* X1 + 1.2 \* X2 + 2.2 \* X3 + 0.4 \* X1 \* X1 + 1.0 \* X2 \* X2 + 6.4 \* X3 \* X3 + 1.1 \* X1 \* X2 + \

                0.1 \* X1 \* X3 + 9.2 \* X2 \* X3 + 1.2 \* X1 \* X2 \* X3 + randint(0, 10) - 5

            return f

        matrix\_with\_y = [[f(matrix\_x[j][0], matrix\_x[j][1], matrix\_x[j][2]) for i in range(self.m)] for j in range(self.n)]

        return matrix\_with\_y

    def x(self, l1, l2, l3):

        x\_1 = l1 \* self.delta\_x1 + self.x01

        x\_2 = l2 \* self.delta\_x2 + self.x02

        x\_3 = l3 \* self.delta\_x3 + self.x03

        return [x\_1, x\_2, x\_3]

    def find\_average(self, lst, orientation):

        average = []

        if orientation == 1:

            for rows in range(len(lst)):

                average.append(sum(lst[rows]) / len(lst[rows]))

        else:

            for column in range(len(lst[0])):

                number\_lst = []

                for rows in range(len(lst)):

                    number\_lst.append(lst[rows][column])

                average.append(sum(number\_lst) / len(number\_lst))

        return average

    def find\_known(self, number):

        need\_a = 0

        for j in range(self.n):

            need\_a += self.average\_y[j] \* self.matrix\_x[j][number - 1] / 15

        return need\_a

    def solve(self, lst\_1, lst\_2):

        solver = solve(lst\_1, lst\_2)

        return solver

    def check\_result(self, b\_lst, k):

        y\_i = b\_lst[0] + b\_lst[1] \* self.matrix[k][0] + b\_lst[2] \* self.matrix[k][1] + b\_lst[3] \* self.matrix[k][2] + \

            b\_lst[4] \* self.matrix[k][3] + b\_lst[5] \* self.matrix[k][4] + b\_lst[6] \* self.matrix[k][5] + b\_lst[7] \* self.matrix[k][6] + \

            b\_lst[8] \* self.matrix[k][7] + b\_lst[9] \* self.matrix[k][8] + b\_lst[10] \* self.matrix[k][9]

        return y\_i

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

    # Перевірка однорідності дисперсії за критерієм Кохрена:

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

    def get\_cohren\_value(self, size\_of\_selections, qty\_of\_selections, significance):

        size\_of\_selections += 1

        partResult1 = significance / (size\_of\_selections - 1)

        params = [partResult1, qty\_of\_selections, (size\_of\_selections - 1 - 1) \* qty\_of\_selections]

        fisher = f.isf(\*params)

        result = fisher / (fisher + (size\_of\_selections - 1 - 1))

        return Decimal(result).quantize(Decimal('.0001')).\_\_float\_\_()

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

    # Перевірка однорідності дисперсії за критерієм Стьюдента:

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

    def student\_test(self, b\_lst, number\_x=10):

        dispersion\_b = sqrt(self.dispersion\_b2)

        for column in range(number\_x + 1):

            t\_practice = 0

            t\_theoretical = self.get\_student\_value(self.f3, self.q)

            for row in range(self.n):

                if column == 0:

                    t\_practice += self.average\_y[row] / self.n

                else:

                    t\_practice += self.average\_y[row] \* self.matrix\_pfe[row][column - 1]

            if fabs(t\_practice / dispersion\_b) < t\_theoretical:

                b\_lst[column] = 0

        return b\_lst

    def get\_student\_value(self, f3, significance):

        return Decimal(abs(t.ppf(significance / 2, f3))).quantize(Decimal('.0001')).\_\_float\_\_()

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

    # Перевірка однорідності дисперсії за критерієм Фішера:

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

    def fisher\_test(self):

        dispersion\_ad = 0

        for row in range(len(self.average\_y)):

            dispersion\_ad += (self.m \* (self.average\_y[row] - self.check\_result(self.student\_lst, row))) / (self.n - self.d)

        self.F\_practice = dispersion\_ad / self.dispersion\_b2

        F\_theoretical = self.get\_fisher\_value(self.q)

        return self.F\_practice < F\_theoretical

    def get\_fisher\_value(self, significance):

        return Decimal(abs(f.isf(significance, self.f4, self.f3))).quantize(Decimal('.0001')).\_\_float\_\_()

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

    # Вивід даних:

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

    def Result(self, n, m, f1, f2, f3, q, matrix\_x):

        while not self.Fisher:

            matrix\_y = self.generate\_matrix(matrix\_x)

            average\_x = self.find\_average(matrix\_x, 0)

            self.average\_y = self.find\_average(matrix\_y, 1)

            self.matrix = [(matrix\_x[i] + matrix\_y[i]) for i in range(n)]

            mx\_i = average\_x

            my = sum(self.average\_y) / 15

            dispersion\_y = [0.0 for x in range(n)]

            for i in range(n):

                dispersion\_i = 0

                for j in range(m):

                    dispersion\_i += (matrix\_y[i][j] - self.average\_y[i]) \*\* 2

                dispersion\_y.append(dispersion\_i / (m - 1))

            print("\nx1min = 15.0 x1max = 25.0\nx2min = -25.0 x2max = 10.0\nx3min = 45.0 x3max = 50.0\n")

            print("\nf(x1, x2, x3) = 8,7+4,3\*x1+1,2\*x2+2,2\*x3+0,4\*x1\*x1+1,0\*x2\*x2+6,4\*x3\*x3+1,1\*x1\*x2+0,1\*x1\*x3+9,2\*x2\*x3+1,2\*x1\*x2\*x3")

            print("\nРівняння регресії: y = b0 + b1\*x'1 + b2\*x2 + b3\*x3 + b12\*x1\*x2 + b13\*x1\*x3 + b23\*x2\*x3 + b123\*x1\*x2\*x3 + b11\*x1\*x1 + b22\*x2\*x2 + b33\*x3\*x3\n")

            th = ["N", "x1", "x2", "x3", "x1\*x2", "x1\*x3", "x2\*x3", "x1\*x2\*x3", "x1^2", "x2^2", "x3^2"]

            columns = len(th)

            table = PrettyTable(th)

            table.title = "Матриця планування експерименту для РЦКП з нормованими значеннями факторів:"

            for i in range(self.n):

                td = [self.N[i], self.matrix\_pfe[i][0], self.matrix\_pfe[i][1], self.matrix\_pfe[i][2], \

                    round(self.matrix\_pfe[i][3],3), round(self.matrix\_pfe[i][4],3), round(self.matrix\_pfe[i][5],3), round(self.matrix\_pfe[i][6],3), \

                        round(self.matrix\_pfe[i][7],3), round(self.matrix\_pfe[i][8],3), round(self.matrix\_pfe[i][9],3)  ]

                td\_data = td[:]

                while td\_data:

                    table.add\_row(td\_data[:columns])

                    td\_data = td\_data[columns:]

            print(table)

            th = ["N", "x1", "x2", "x3", "x1\*x2", "x1\*x3", "x2\*x3", "x1\*x2\*x3", "x1^2", "x2^2", "x3^2"]

            th.append("<y>")

            columns = len(th)

            table = PrettyTable(th)

            table.title = "Матриця планування експерименту для РЦКП з натуральними значеннями факторів: "

            for i in range(self.n):

                td = [self.N[i], self.matrix[i][0], self.matrix[i][1], self.matrix[i][2], \

                    round(self.matrix[i][3],3), round(self.matrix[i][4],3), round(self.matrix[i][5],3), round(self.matrix[i][6],3), \

                        round(self.matrix[i][7],3), round(self.matrix[i][8],3), round(self.matrix[i][9],3)  ]

                td.append(round(self.average\_y[i],2))

                td\_data = td[:]

                while td\_data:

                    table.add\_row(td\_data[:columns])

                    td\_data = td\_data[columns:]

            print(table)

            def a(first, second):

                need\_a = 0

                for j in range(n):

                    need\_a += matrix\_x[j][first - 1] \* matrix\_x[j][second - 1] / n

                return need\_a

            koef = [

                [1, mx\_i[0], mx\_i[1], mx\_i[2], mx\_i[3], mx\_i[4], mx\_i[5], mx\_i[6], mx\_i[7], mx\_i[8], mx\_i[9]],

                [mx\_i[0], a(1, 1), a(1, 2), a(1, 3), a(1, 4), a(1, 5), a(1, 6), a(1, 7), a(1, 8), a(1, 9), a(1, 10)],

                [mx\_i[1], a(2, 1), a(2, 2), a(2, 3), a(2, 4), a(2, 5), a(2, 6), a(2, 7), a(2, 8), a(2, 9), a(2, 10)],

                [mx\_i[2], a(3, 1), a(3, 2), a(3, 3), a(3, 4), a(3, 5), a(3, 6), a(3, 7), a(3, 8), a(3, 9), a(3, 10)],

                [mx\_i[3], a(4, 1), a(4, 2), a(4, 3), a(4, 4), a(4, 5), a(4, 6), a(4, 7), a(4, 8), a(4, 9), a(4, 10)],

                [mx\_i[4], a(5, 1), a(5, 2), a(5, 3), a(5, 4), a(5, 5), a(5, 6), a(5, 7), a(5, 8), a(5, 9), a(5, 10)],

                [mx\_i[5], a(6, 1), a(6, 2), a(6, 3), a(6, 4), a(6, 5), a(6, 6), a(6, 7), a(6, 8), a(6, 9), a(6, 10)],

                [mx\_i[6], a(7, 1), a(7, 2), a(7, 3), a(7, 4), a(7, 5), a(7, 6), a(7, 7), a(7, 8), a(7, 9), a(7, 10)],

                [mx\_i[7], a(8, 1), a(8, 2), a(8, 3), a(8, 4), a(8, 5), a(8, 6), a(8, 7), a(8, 8), a(8, 9), a(8, 10)],

                [mx\_i[8], a(9, 1), a(9, 2), a(9, 3), a(9, 4), a(9, 5), a(9, 6), a(9, 7), a(9, 8), a(9, 9), a(9, 10)],

                [mx\_i[9], a(10, 1), a(10, 2), a(10, 3), a(10, 4), a(10, 5), a(10, 6), a(10, 7), a(10, 8), a(10, 9), a(10, 10)]

            ]

            known = [my, self.find\_known(1), self.find\_known(2), self.find\_known(3), self.find\_known(4), self.find\_known(5), \

                 self.find\_known(6), self.find\_known(7), self.find\_known(8), self.find\_known(9), self.find\_known(10)]

            beta = self.solve(koef, known)

            print("\nОтримане рівняння регресії: y = {:.3f} + {:.3f} \* x1 + {:.3f} \* x2 + {:.3f} \* x3 + {:.3f} \* x1x2 + {:.3f} \* x1x3 + {:.3f} \* x2x3"

                "+ {:.3f} \* x1x2x3 + {:.3f} \* x1^2 + {:.3f} \* x2^2 + {:.3f} \* x3^2 \nПеревірка"

                .format(beta[0], beta[1], beta[2], beta[3], beta[4], beta[5], beta[6], beta[7], beta[8], beta[9], beta[10]))

            for i in range(n):

                print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), self.check\_result(beta, i), self.average\_y[i]))

            while not self.Kohren:

                Gp = max(dispersion\_y) / sum(dispersion\_y)

                print("\nКритерій Кохрена: Gp = {}".format(Gp))

                Gt = self.get\_cohren\_value(f2, f1, q)

                if Gt > Gp:

                    print("Дисперсія однорідна при рівні значимості {:.2f}.\n".format(q))

                    self.Kohren = True

                else:

                    print("Дисперсія не однорідна при рівні значимості {:.2f}! Збільшуємо m.\n".format(q))

                    self.m += 1

            self.dispersion\_b2 = sum(dispersion\_y) / (n \* n \* m)

            self.student\_lst = list(self.student\_test(beta))

            print("Отримане рівняння регресії з урахуванням критерія Стьюдента: y = {:.3f} + {:.3f} \* x1 + {:.3f} \* x2 + {:.3f} \* x3 + {:.3f} \* x1x2 + {:.3f} \* x1x3 + {:.3f} \* x2x3"

                "+ {:.3f} \* x1x2x3 + {:.3f} \* x1^2 + {:.3f} \* x2^2 + {:.3f} \* x3^2 \nПеревірка"

                .format(self.student\_lst[0], self.student\_lst[1], self.student\_lst[2], self.student\_lst[3], self.student\_lst[4], self.student\_lst[5],

                        self.student\_lst[6], self.student\_lst[7], self.student\_lst[8], self.student\_lst[9], self.student\_lst[10]))

            for i in range(n):

                print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), self.check\_result(self.student\_lst, i), self.average\_y[i]))

            self.d = 11 - self.student\_lst.count(0)

            self.f4 = self.n - self.d

            self.fisher\_test()

            print("\nКритерій Фішера: Fp = {}".format(self.F\_practice))

            if self.fisher\_test():

                print("Рівняння регресії адекватне  оригіналу")

                self.Fisher = True

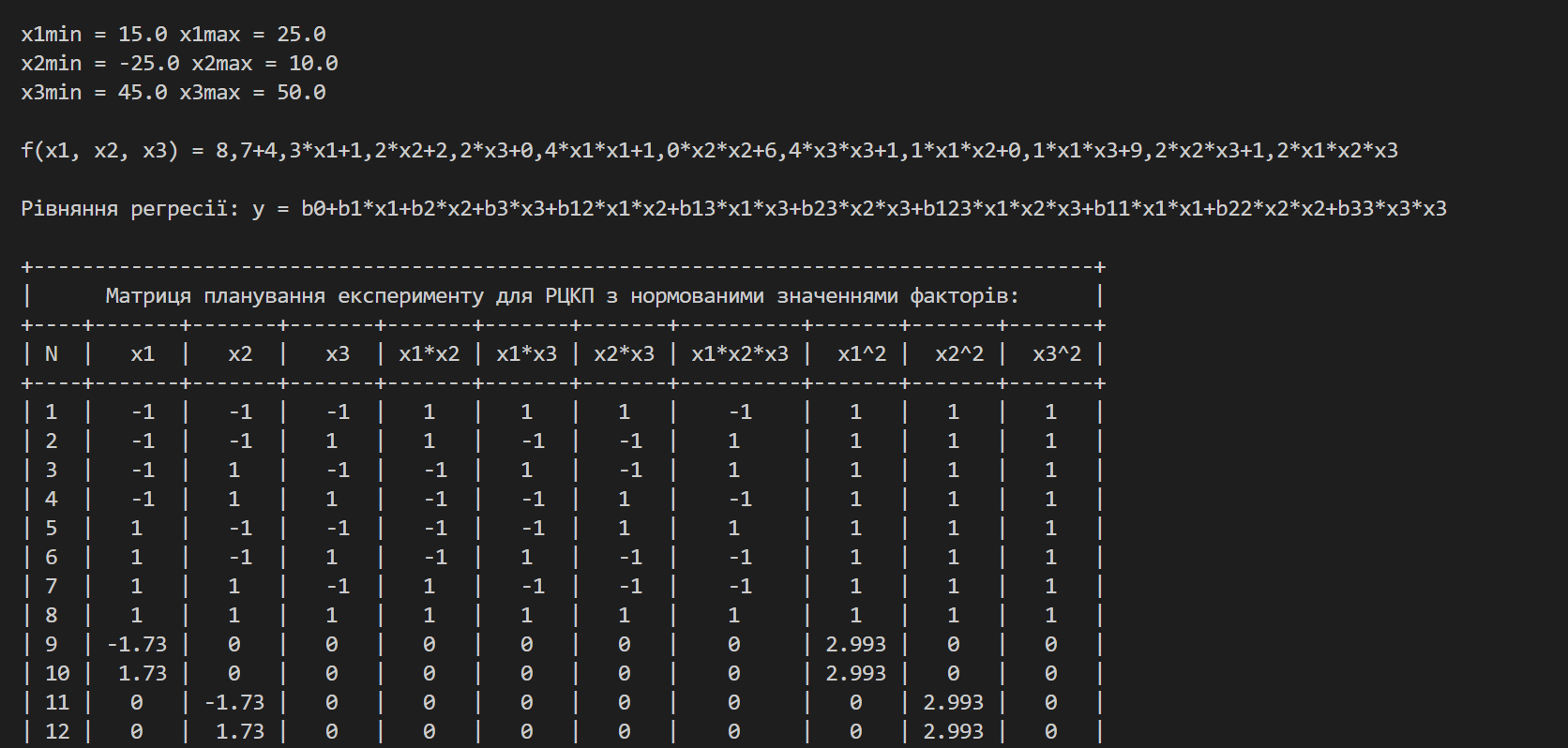
            else:

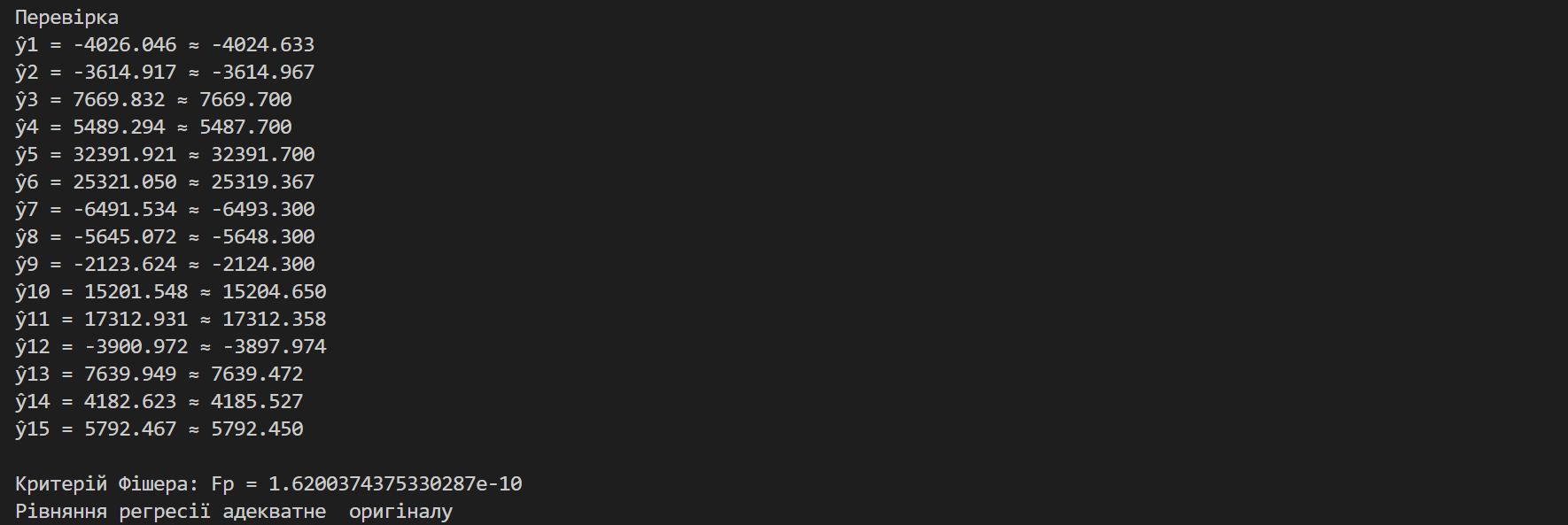
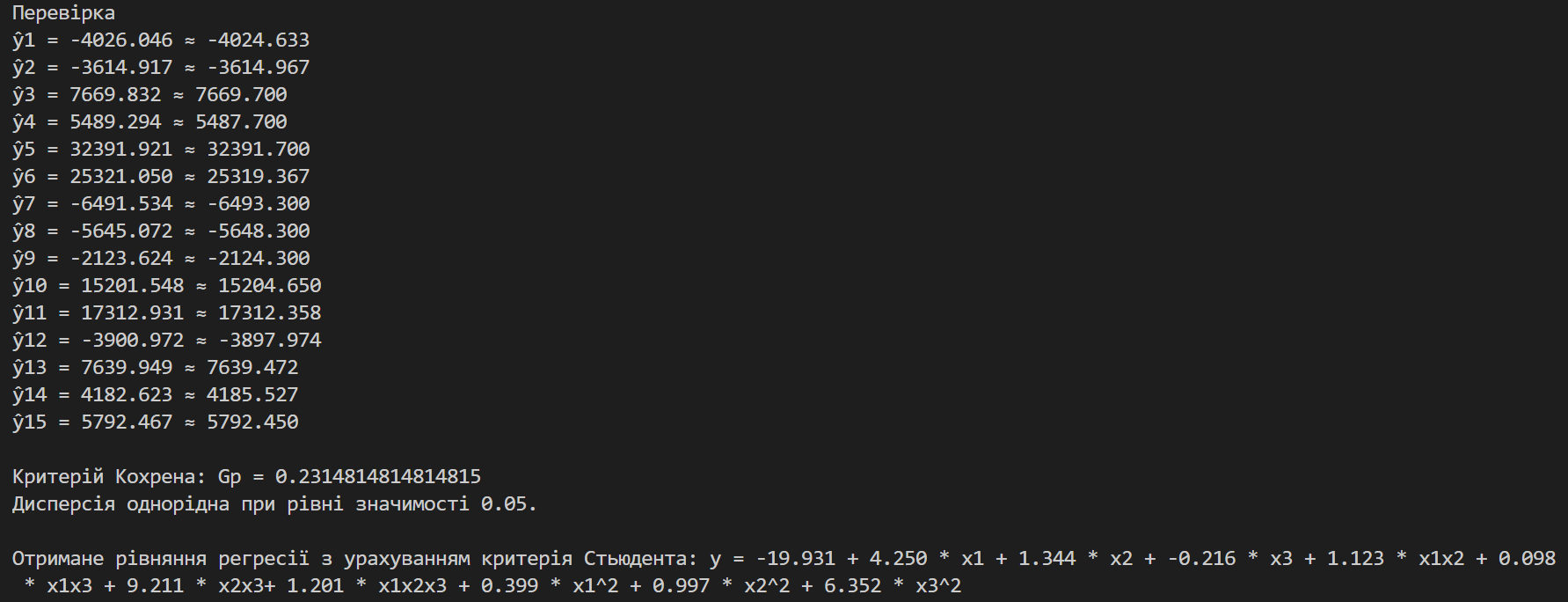
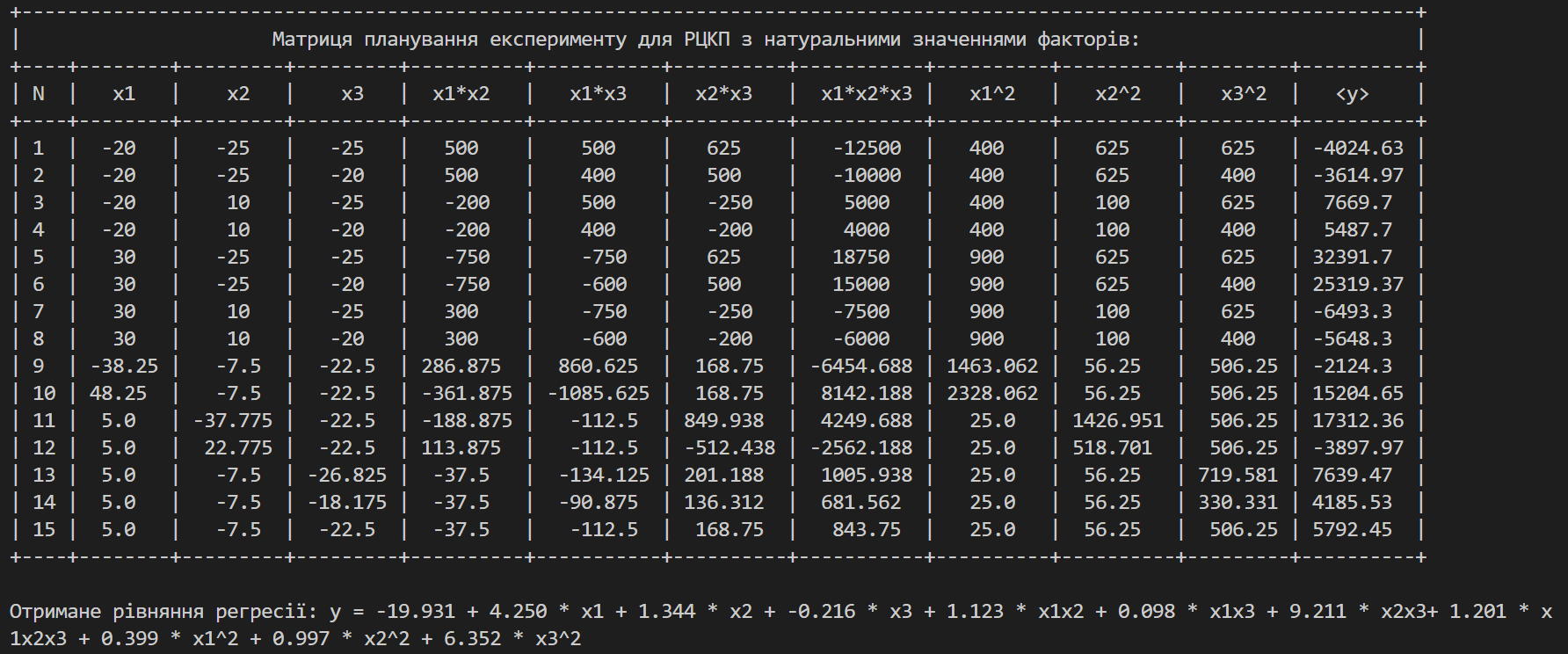
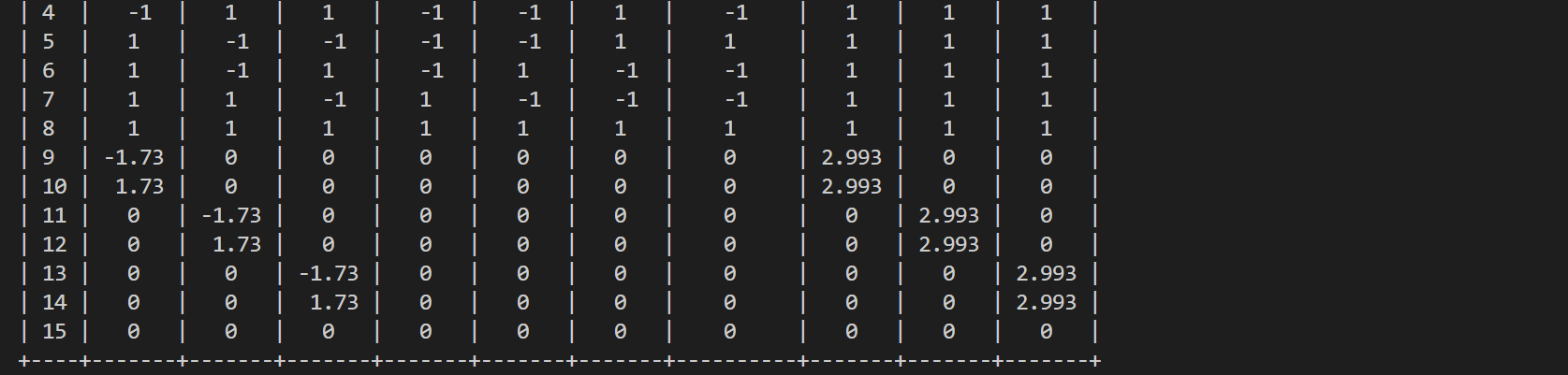
                print("Рівняння регресії неадекватне  оригіналу\n\t Проводимо експеремент повторно")

if \_\_name\_\_ == '\_\_main\_\_':

    Experiment(15, 3)

**Результати роботи програми:**





**Висновок:** У ході лабораторної роботи був проведений трьохфакторний експеримент, в резудьтаті якого я отримав адекватну модель – рівняння регресії, за допомогою рототабельного композиційного плану.