Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №6

з дисципліни МОПЕ

на тему:

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

Виконав:

студент групи ІВ-82

Стрілецький Вадим

Залікова книжка № 8225

Варіант: 224

Перевірив:

Регіда П.Г.

Київ – 2020

****

**Код програми:**

import math

import numpy as np

from scipy.stats import t,f

import random as r

import prettytable as p

class LabSix():

    def \_\_init\_\_(self, m, p):

        self.m = m

        self.P = p

        self.k = 3

        self.X1\_range = [-5, 5]

        self.X2\_range = [-5, 6]

        self.X3\_range = [-4, 8]

        self.X\_ranges = [self.X1\_range, self.X2\_range, self.X3\_range]

        self.X0i = [self.get\_x0(i) for i in self.X\_ranges]

        self.detXi = [self.get\_detX(self.X0i[0], self.X1\_range[1]), self.get\_detX(self.X0i[1], self.X2\_range[1]), self.get\_detX(self.X0i[2], self.X3\_range[1])]

        self.Xcp\_min = self.get\_average([self.X1\_range[0]+ self.X2\_range[0]+self.X3\_range[0]])

        self.Xcp\_max = self.get\_average([self.X1\_range[1]+ self.X2\_range[1]+self.X3\_range[1]])

        self.Y\_min = 200 + self.Xcp\_min

        self.Y\_max = 200 + self.Xcp\_max

        self.l = 1.215

        self.X1\_norm = [-1, -1, -1, -1, 1, 1, 1, 1, -self.l, self.l, 0, 0, 0, 0, 0]

        self.X2\_norm = [-1, -1, 1, 1, -1, -1, 1, 1, 0, 0, -self.l, self.l, 0, 0, 0]

        self.X3\_norm = [-1, 1, -1, 1, -1, 1, -1, 1, 0, 0, 0, 0, -self.l, self.l, 0]

        self.X\_norm = [self.X1\_norm, self.X2\_norm, self.X3\_norm]

        self.N = len(self.X1\_norm)

        self.X1\_abs = self.get\_abs\_values(self.X1\_norm, 1)

        self.X2\_abs = self.get\_abs\_values(self.X2\_norm, 2)

        self.X3\_abs = self.get\_abs\_values(self.X3\_norm, 3)

        self.X\_abs = [self.X1\_abs, self.X2\_abs, self.X3\_abs]

        self.add\_efects(self.X\_norm)

        self.add\_efects(self.X\_abs)

        self.X\_types = X\_types = ["X1", "X2", "X3", "X12", "X13", "X23", "X123", "X1^2", "X2^2", "X3^2"]

    def get\_abs\_values(self, X\_norm, n):

        n = n - 1

        X\_abs = []

        for i in X\_norm:

            if(i == 1):

                X\_abs.append(self.X\_ranges[n][1])

            elif(i == -1):

                X\_abs.append(self.X\_ranges[n][0])

            elif(i == 0):

                X\_abs.append(self.X0i[n])

            elif(i == self.l):

                X\_abs.append(round(self.l\*self.detXi[n] + self.X0i[n], 3))

            else:

                X\_abs.append(round(-self.l\*self.detXi[n] + self.X0i[n], 3))

        return X\_abs

    def get\_x0(self, X\_range):

        return (X\_range[0]+X\_range[1])/2

    def get\_detX(self, x\_max, x0):

        return x\_max - x0

    def get\_average(self, y):

        return sum(y)/len(y)

    def make\_experiment(self):

        self.Y\_exp = [[self.varint\_func(self.X1\_abs[i], self.X2\_abs[i], self.X3\_abs[i]) for \_ in range(self.m)] for i in range(self.N)]

    def create\_and\_print\_table(self, X\_list, print\_that):

        table = p.PrettyTable()

        for i in range(len(X\_list)):

            table.add\_column(self.X\_types[i], X\_list[i])

        for i in range(self.m):

            table.add\_column("Y{0}".format(i+1), [j[i] for j in self.Y\_exp])

        print(print\_that)

        print(table)

        return table

    def add\_efects(self, X\_list):

        x1 = X\_list[0]

        x2 = X\_list[1]

        x3 = X\_list[2]

        x12 = [round(x1[i]\*x2[i], 3) for i in range(self.N)]

        x13 = [round(x1[i]\*x3[i], 3) for i in range(self.N)]

        x23 = [round(x2[i]\*x3[i], 3) for i in range(self.N)]

        x123 = [round(x1[i]\*x2[i]\*x3[i], 3) for i in range(self.N)]

        x1pow = [round(i\*\*2, 3) for i in x1]

        x2pow = [round(i\*\*2, 3) for i in x2]

        x3pow = [round(i\*\*2, 3) for i in x3]

        X\_list.append(x12)

        X\_list.append(x13)

        X\_list.append(x23)

        X\_list.append(x123)

        X\_list.append(x1pow)

        X\_list.append(x2pow)

        X\_list.append(x3pow)

    def set\_y\_aver(self):

        self.y\_aver = [self.get\_average(i) for i in self.Y\_exp]

    def find\_coefs(self):

        x0 = [1]\*15

        x1 = self.X\_abs[0]

        x2 = self.X\_abs[1]

        x3 = self.X\_abs[2]

        x12 = self.X\_abs[3]

        x13 = self.X\_abs[4]

        x23 = self.X\_abs[5]

        x123 = self.X\_abs[6]

        x1pow = self.X\_abs[7]

        x2pow = self.X\_abs[8]

        x3pow = self.X\_abs[9]

        self.lines\_of\_matr = list(zip(x0, x1, x2, x3, x12, x13, x23, x123, x1pow, x2pow, x3pow))

        self.b = np.linalg.lstsq(self.lines\_of\_matr, self.y\_aver, rcond=None)[0]

        self.b = [round(i, 3) for i in self.b]

        print("Апроксимуюча функція:")

        print("y = {0} + {1}\*x1 + {2}\*x2 + {3}\*x3 + {4}\*x1\*x2 + {5}\*x1\*x3 + {6}\*x2\*x3 + {7}\*x1\*x2\*x3 + {8}\*x1\*x1 + {9}\*x2\*x2 + {10}\*x3\*x3".format(\*self.b))

        print("Задана функція:")

        print("y = 3.5 + 6.6\*x1 + 3.9\*x2+ 1.8\*x3 + 6.0\*x1\*x2+ 0.8\*x1\*x3+ 9.4\*x2\*x3 + 3.0\*x1\*x2\*x3 + 5.3\*x1\*x1 + 0.5\*x2\*x2 + 4.3\*x3\*x3")

    def varint\_func(self, x1, x2, x3):

        return 3.5+6.6\*x1+3.9\*x2+1.8\*x3+5.3\*x1\*x1+0.5\*x2\*x2+4.3\*x3\*x3+6.0\*x1\*x2+0.8\*x1\*x3+9.4\*x2\*x3+3.0\*x1\*x2\*x3 + r.randrange(0, 10) - 5

    def get\_func\_value(self, position):

        return sum([self.b[i]\*self.lines\_of\_matr[position][i] for i in range(len(self.b))])

    def check\_func\_values(self):

        func\_values = [round(self.get\_func\_value(i), 3) for i in range(15)]

        for i in range(self.N):

            print("y{0} = {1} ≈ {2}".format(i+1, round(func\_values[i], 3), round(self.y\_aver[i], 3)))

        return func\_values

    def get\_cohren\_critical(self, prob, f1, f2):

        f\_crit = f.isf((1 - prob) / f2, f1, (f2 - 1) \* f1)

        return f\_crit / (f\_crit + f2 - 1)

    def cohren\_check(self):

        self.f1 = self.m - 1

        self.f2 = self.N

        self.y\_disp = [self.get\_dispersion(self.y\_aver[i], self.Y\_exp[i]) for i in range(self.N)]

        Gp = max(self.y\_disp)/sum(self.y\_disp)

        Gt = self.get\_cohren\_critical(self.P, self.f1, self.f2)

        self.separator()

        if(Gp < Gt):

            print("Дисперсії однорідні")

            return True

        else:

            print("Дисперсії не однорідні, m+=1")

            self.m += 1

            self.Y\_exp[0].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[1].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[2].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[3].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[4].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[5].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[6].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[7].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[8].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[9].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[10].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[11].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[12].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[13].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            self.Y\_exp[14].append(r.randint(math.floor(self.Y\_min), math.floor(self.Y\_max)))

            return False

    def get\_dispersion(self, y\_aver, y):

        return sum([(i-y\_aver)\*\*2 for i in y])/len(y)

    def get\_fisher\_teor(self, q, f3, f4):

       return f.isf(q, f4, f3)

    def get\_student\_teor(self, f3, q):

        return t.ppf(q / 2, f3)

    def before\_chochren\_steps(self):

        self.norm\_table = self.create\_and\_print\_table(self.X\_norm, "Нормалізована матриця")

        self.abs\_table = self.create\_and\_print\_table(self.X\_abs, "Абсолютна матриця")

        self.set\_y\_aver()

        return self.cohren\_check()

    def student\_crit(self):

        self.d = 0

        S2B = sum(self.y\_disp)/self.N

        S2b = S2B/(self.N\*self.m)

        Sb = math.sqrt(S2b)

        betai = np.zeros(len(self.lines\_of\_matr[0]))

        betai[0] = sum(self.y\_aver)

        for i in range(len(self.X\_norm)):

            betai[i+1] = sum([self.X\_norm[i][j]\*self.y\_aver[j] for j in range(self.N)])

        ts = []

        for b in betai:

            ts.append(b/Sb)

        self.f3 = self.f1\*self.f2

        Stud\_teor = self.get\_student\_teor(self.f3, 1 - self.P)

        for i in range(len(ts)):

            if ts[i] < Stud\_teor:

                self.b[i] = 0

            else:

                self.d += 1

        print("Кількість значимих коефіцієнтів:", self.d)

        print("Апроксимуюча функція після перевірки значимості коефіцієнтів:")

        print("y = {0} + {1}\*x1 + {2}\*x2 + {3}\*x3 + {4}\*x1\*x2 + {5}\*x1\*x3 + {6}\*x2\*x3 + {7}\*x1\*x2\*x3 + {8}\*x1\*x1 + {9}\*x2\*x2 + {10}\*x3\*x3".format(\*self.b))

        print("Після перевірки значимості коефіцієнтів")

        self.y\_new = self.check\_func\_values()

    def fisher\_critical(self):

        self.f4 = self.N - self.d

        S2ad = abs((self.m/self.f4)\*sum([(self.y\_new[i] - self.y\_aver[i])\*\*2 for i in range(self.N)]))

        Sa = sum(self.y\_disp)/self.N

        Fp = S2ad/Sa

        Fkr = self.get\_fisher\_teor(1 - self.P, self.f3, self.f4)

        if(Fkr > Fp):

            print("Рівняння адекватне оригіналу")

            print("Fkr =", Fkr, "> Fp =", Fp)

            self.is\_adekvat = True

        else:

            print("Рівняння неадекватне оригіналу")

            print("Fkr =", Fkr, "< Fp =", Fp, "\nПочинаємо спочатку")

            self.separator()

    def separator(self):

        print("\n"+"-"\*188)

    def run(self):

        self.is\_adekvat = False

        while(not self.is\_adekvat):

            self.make\_experiment()

            while not self.before\_chochren\_steps():

                pass

            self.find\_coefs()

            self.check\_func\_values()

            self.separator()

            self.student\_crit()

            self.fisher\_critical()

labSix = LabSix(3, 0.95)

labSix.run()