Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №3

З дисципліни «Методи оптимізації та планування»

Тема: ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ.

ВИКОНАВ:
Студент ІІ курсу ФІОТ
Групи ІО-92
Рожко М.М.

ПЕРЕВІРИВ:
асистент
Регіда П.Г.

Київ 2021 р.

Мета:

Провести дробовий трьохфакторний експеримент. Скласти матрицю планування, знайти коефіцієнти рівняння регресії, провести 3 статистичні перевірки.

Варіант завдання:

N	X1		X2		X3	
	min	max	min	max	min	max
217	20	70	5	40	20	45

Розруківка коду програми:

```
from tkinter import *
import random as r
import numpy as np
from math import *
import plotly.figure_factory as ff
from scipy.stats import f

class Window:
    def __init__(self):
        self.window = Tk()
        self.window.title("МОПЕ Лабораторна робота №3")
        self.m, self.n = 3, 4
        self.x1_min, self.x1_max = 20, 70
        self.x2_min, self.x2_max = 5, 40
        self.x3_min, self.x3_max = 20, 45
        self.y_max = int(200 + (self.x1_max + self.x2_min + self.x3_max) /
self.m)
        self.y_min = int(200 + (self.x1_min + self.x2_min + self.x3_min) /
self.m)

        self.equal_regression_l = Label(self.window, text='y = b0 + b1*x1 +
b2*x2 + b3*x3',
                                        width=30, relief='groove', bg='Light
blue', pady=10).grid(row=0, column=0, columnspan=2)
        self.variant_b = Button(self.window, text='Таблиця варіанту',
width=30, command=self.table_variant, bg='Gold')
        self.variant_b.grid(row=1, column=0, columnspan=2)

        self.xi_coord = [[1, -1, -1, -1],
                         [1, -1, 1, 1],
                         [1, 1, -1, 1],
                         [1, 1, 1, -1]]
        self.x_arr = [[self.x1_min, self.x2_min, self.x3_min],
                      [self.x1_min, self.x2_max, self.x3_max],
                      [self.x1_max, self.x2_min, self.x3_max],
                      [self.x1_max, self.x2_max, self.x3_min]]

        self.yi_columns = [[r.randint(self.y_min, self.y_max) for i in
range(self.n)] for j in range(self.m)]
```

```
        self.three_factor_b = Button(self.window, text='Матриця
планування\nдля\nдробового трьохфакторного\nексперименту',
                                     width=30,
command=self.table_three_factor_matrix, bg='Gold')
        self.three_factor_b.grid(row=2, column=0, columnspan=2)
        self.naturalized_b = Button(self.window,
                                    text='Матриця планування з
відповідними\nнатуралізованими значеннями\nфакторів',
                                    width=30,
command=self.table_naturalized_matrix, bg='Gold')
        self.naturalized_b.grid(row=3, column=0, columnspan=2)

        self.y1_avg = (self.yi_columns[0][0] + self.yi_columns[1][0] +
self.yi_columns[2][0]) / self.m
        self.y2_avg = (self.yi_columns[0][1] + self.yi_columns[1][1] +
self.yi_columns[2][1]) / self.m
        self.y3_avg = (self.yi_columns[0][2] + self.yi_columns[1][2] +
self.yi_columns[2][2]) / self.m
        self.y4_avg = (self.yi_columns[0][3] + self.yi_columns[1][3] +
self.yi_columns[2][3]) / self.m
        self.yi_avg = [self.y1_avg, self.y2_avg, self.y3_avg, self. y4_avg]
        self.mx1 = sum([i[0] for i in self.x_arr]) / self.n
        self.mx2 = sum([i[1] for i in self.x_arr]) / self.n
        self.mx3 = sum([i[2] for i in self.x_arr]) / self.n
        self.my = sum(self.yi_avg) / self.n

        self.a1 = sum([self.x_arr[i][0]*self.yi_avg[i] for i in
range(self.n)]) / self.n
        self.a2 = sum([self.x_arr[i][1]*self.yi_avg[i] for i in
range(self.n)]) / self.n
        self.a3 = sum([self.x_arr[i][2]*self.yi_avg[i] for i in
range(self.n)]) / self.n
        self.a11 = sum([self.x_arr[i][0]*self.x_arr[i][0] for i in
range(self.n)]) / self.n
        self.a22 = sum([self.x_arr[i][1] * self.x_arr[i][1] for i in
range(self.n)]) / self.n
        self.a33 = sum([self.x_arr[i][2] * self.x_arr[i][2] for i in
range(self.n)]) / self.n
        self.a12 = self.a21 = sum([self.x_arr[i][0] * self.x_arr[i][1] for i
in range(self.n)]) / self.n
        self.a13 = self.a31 = sum([self.x_arr[i][0] * self.x_arr[i][2] for i
in range(self.n)]) / self.n
        self.a23 = self.a32 = sum([self.x_arr[i][1] * self.x_arr[i][2] for i
in range(self.n)]) / self.n

        self.b0 = ((np.linalg.det(np.array([[self.my, self.mx1, self.mx2,
self.mx3], [self.a1, self.a11, self.a12, self.a13],
                                            [self.a2, self.a12, self.a22,
self.a32], [self.a3, self.a13, self.a23, self.a33]]))) /
                   ((np.linalg.det(np.array([[1, self.mx1, self.mx2,
self.mx3], [self.mx1, self.a11, self.a12, self.a13],
                                             [self.mx2, self.a12, self.a22,
self.a32], [self.mx3, self.a13, self.a23, self.a33]]))))
        self.b1 = ((np.linalg.det(np.array([[1, self.my, self.mx2, self.mx3],
[self.mx1, self.a1, self.a12, self.a13],
                                            [self.mx2, self.a2, self.a22,
self.a32], [self.mx3, self.a3, self.a23, self.a33]]))) /
                   ((np.linalg.det(np.array( [[1, self.mx1, self.mx2,
self.mx3], [self.mx1, self.a11, self.a12, self.a13],
                                             [self.mx2, self.a12, self.a22,
self.a32], [self.mx3, self.a13, self.a23, self.a33]]))))
        self.b2 = ((np.linalg.det(np.array([[1, self.mx1, self.my, self.mx3],
[self.mx1, self.a11, self.a1, self.a13],
```

```
                                                [self.mx2, self.a12, self.a2,
self.a32], [self.mx3, self.a13, self.a3, self.a33]])) /
                    ((np.linalg.det(np.array( [[1, self.mx1, self.mx2,
self.mx3], [self.mx1, self.a11, self.a12, self.a13],
                                                [self.mx2, self.a12, self.a22,
self.a32], [self.mx3, self.a13, self.a23, self.a33]]))))
        self.b3 = ((np.linalg.det(np.array([[1, self.mx1, self.mx2, self.my],
[self.mx1, self.a11, self.a12, self.a1],
                                                [self.mx2, self.a12, self.a22,
self.a2], [self.mx3, self.a13, self.a23, self.a3]])) /
                    ((np.linalg.det(np.array( [[1, self.mx1, self.mx2,
self.mx3], [self.mx1, self.a11, self.a12, self.a13],
                                                [self.mx2, self.a12, self.a22,
self.a32], [self.mx3, self.a13, self.a23, self.a33]]))))
        self.print_regression_equation()
        self.criterionKohren()
        self.window.mainloop()


    def make_table(self, data):
        fig = ff.create_table(data)
        fig.show()

    def table_variant(self):
        data = [['X1min', 'X1max', 'X2min', 'X2max', 'X3min', 'X3max',
'Ymin', 'Ymax'],
            [self.x1_min, self.x1_max, self.x2_min, self.x2_max,
             self.x3_min, self.x3_max, self.y_min, self.y_max]]
        self.make_table(data)

    def table_three_factor_matrix(self):
        data = [['N', 'X0', 'X1', 'X2', 'X3', 'Y1', 'Y2', 'Y3'],
                ['1'] + self.xi_coord[0] + [self.yi_columns[i][0] for i in
range(self.m)],
                ['2'] + self.xi_coord[1] + [self.yi_columns[i][1] for i in
range(self.m)],
                ['3'] + self.xi_coord[2] + [self.yi_columns[i][2] for i in
range(self.m)],
                ['4'] + self.xi_coord[3] + [self.yi_columns[i][3] for i in
range(self.m)]]
        self.make_table(data)

    def table_naturalized_matrix(self):
        data = [['X1', 'X2', 'X3', 'Y1', 'Y2', 'Y3'],
                self.x_arr[0] + [self.yi_columns[i][0] for i in
range(self.m)],
                self.x_arr[1] + [self.yi_columns[i][1] for i in
range(self.m)],
                self.x_arr[2] + [self.yi_columns[i][2] for i in
range(self.m)],
                self.x_arr[3] + [self.yi_columns[i][3] for i in
range(self.m)]]
        self.make_table(data)

    def print_regression_equation(self):
        print("Рівняння регресії: y =
{0:.2f}+({1:.2f})*X1+({2:.2f})*X2+({3:.2f})*X3".format(self.b0, self.b1,
self.b2, self.b3))
        print("Перевірка:")

print("{0:.2f}+({1:.2f})*({2:.2f})+({3:.2f})*({4:.2f})+({5:.2f})*({6:.2f}) =
{7:.2f} = Y1cep = {8:.2f}".format(
            self.b0, self.b1, self.x1_min, self.b2, self.x2_min, self.b3,
self.x3_min,
```

```
            (self.b0 + (self.b1 * self.x1_min) + (self.b2 * self.x2_min) +
(self.b3 * self.x3_min)), self.y1_avg))

print("{0:.2f}+({1:.2f})*({2:.2f})+({3:.2f})*({4:.2f})+({5:.2f})*({6:.2f}) =
{7:.2f} = Y2cep = {8:.2f}".format(
            self.b0, self.b1, self.x1_min, self.b2, self.x2_max, self.b3,
self.x3_max,
            (self.b0 + (self.b1 * self.x1_min) + (self.b2 * self.x2_max) +
(self.b3 * self.x3_max)), self.y2_avg))

print("{0:.2f}+({1:.2f})*({2:.2f})+({3:.2f})*({4:.2f})+({5:.2f})*({6:.2f}) =
{7:.2f} = Y3cep = {8:.2f}".format(
            self.b0, self.b1, self.x1_max, self.b2, self.x2_min, self.b3,
self.x3_max,
            (self.b0 + (self.b1 * self.x1_max) + (self.b2 * self.x2_min) +
(self.b3 * self.x3_max)), self.y3_avg))

print("{0:.2f}+({1:.2f})*({2:.2f})+({3:.2f})*({4:.2f})+({5:.2f})*({6:.2f}) =
{7:.2f} = Y4cep = {8:.2f}".format(
            self.b0, self.b1, self.x1_max, self.b2, self.x2_max, self.b3,
self.x3_min,
            (self.b0 + (self.b1 * self.x1_max) + (self.b2 * self.x2_max) +
(self.b3 * self.x3_min)), self.y4_avg))


    def criterionKohren(self):
        print("\nПеревірка рівномірності дисперсій за критерієм Кохрена (M =
{0}, N = {1}):".format(self.m, self.n))
        self.cochraneTable = {1: 9065, 2: 7679, 3: 6841, 4: 6287, 5: 5892, 6:
5598, 7: 5365, 8: 5175, 9: 5017, 10: 4884}
        self.Ydisp = [np.var(i) for i in self.yi_columns]
        self.GP = max(self.Ydisp) / sum(self.Ydisp)
        self.F1 = self.m - 1
        self.F2 = self.n
        self.q = 0.05
        print("F1 = M - 1 = {0} - 1 = {1} \nF2 = N = {2} \nq =
{3}".format(self.m, self.F1, self.F2, self.q))
        self.cochraneValues, self.cochraneKeys =
list(self.cochraneTable.values()), list(self.cochraneTable.keys())
        for keys in range(len(self.cochraneKeys)):
            if (self.cochraneKeys[keys] == self.F1):
                self.GT = self.cochraneValues[keys] / pow(10, 4)
        if (self.GP < self.GT):
            print("GP = {0:.4f} < GT = {1:.4f} - Дисперсія
однорідна!".format(self.GP, self.GT))
            self.criterionStudent()
        else:
            print("GP = {0:.4f} > GT = {1} - Дисперсія неоднорідна! Змінимо M
на M=M+1".format(self.GP, self.GT))
            self.M = self.M + 1
            self.__init__()

    def criterionStudent(self):
        print("\nПеревірка значимості коефіцієнтів регресії згідно критерію
Стьюдента (M = {0}, N = {1}):".format(self.m, self.n))
        self.studentTable = {1: 12.71, 2: 4.303, 3: 3.182, 4: 2.776, 5:
2.571, 6: 2.447, 7: 2.365, 8: 2.306, 9: 2.262, 10: 2.228,
                             11: 2.201, 12: 2.179, 13: 2.160, 14: 2.145, 15:
2.131, 16: 2.120, 17: 2.110, 18: 2.101, 19: 2.093, 20: 2.086,
                             21: 2.080, 22: 2.074, 23: 2.069, 24: 2.064, 25:
2.060, 26: 2.056, 27: 2.052, 28: 2.048, 29: 2.045, 30: 2.042}
        self.Sb = (float(sum(self.Ydisp)) / self.n)
        self.Sbs = sqrt(((self.Sb) / (self.n * self.m)))
        self.xis = np.array([[x[i] for x in self.xi_coord] for i in
```

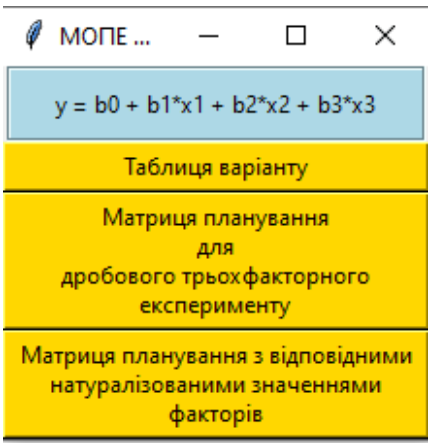
```
range(len(self.xi_coord))])
        self.Beta = np.array([np.average(self.yi_avg * self.xis[i]) for i in
range(len(self.xis))])
        print("Оцінки коефіцієнтів Bs: B1={0:.2f}, B2={1:.2f}, B3={2:.2f},
B4={3:.2f}".format(self.Beta[0], self.Beta[1], self.Beta[2], self.Beta[3]))
        self.t = np.array([((fabs(self.Beta[i])) / self.Sbs) for i in
range(self.n)])
        print("Коефіцієнти ts: t1={0:.2f}, t2={1:.2f}, t3={2:.2f},
t4={3:.2f}".format(self.t[0], self.t[1], self.t[2], self.t[3]))
        self.F3 = self.F1 * self.F2
        print("F3 = F1*F2 = {0}*{1} = {2} \nq = {3}".format(self.F1, self.F2,
self.F3, self.q))
        self.studentValues, self.studentKeys =
list(self.studentTable.values()), list(self.studentTable.keys())
        for keys in range(len(self.studentKeys)):
            if (self.studentKeys[keys] == self.F3):
                self.Ttab = self.studentValues[keys]
        print("t табличне = {0}".format(self.Ttab))
        self.ZO = {}
        for i in range(len(self.t)):
            if ((self.t[i]) > self.Ttab):
                self.ZO[i] = 1
            if ((self.t[i]) < self.Ttab):
                self.ZO[i] = 0
        print("Рівняння регресії: y =
{0:.2f}*({1})+({2:.2f})*({3})*X1+({4:.2f})*({5})*X2+({6:.2f})*({7})*X3".forma
t(
            self.b0, self.ZO[0], self.b1, self.ZO[1], self.b2, self.ZO[2],
self.b3, self.ZO[3]))
        self.Y1v = self.b0 * (self.ZO[0]) + self.b1 * (self.ZO[1]) *
self.x_arr[0][0] + self.b2 * (self.ZO[2]) * self.x_arr[0][
            1] + self.b3 * (self.ZO[3]) * self.x_arr[0][2]
        self.Y2v = self.b0 * (self.ZO[0]) + self.b1 * (self.ZO[1]) *
self.x_arr[1][0] + self.b2 * (self.ZO[2]) * self.x_arr[1][
            1] + self.b3 * (self.ZO[3]) * self.x_arr[1][2]
        self.Y3v = self.b0 * (self.ZO[0]) + self.b1 * (self.ZO[1]) *
self.x_arr[2][0] + self.b2 * (self.ZO[2]) * self.x_arr[2][
            1] + self.b3 * (self.ZO[3]) * self.x_arr[2][2]
        self.Y4v = self.b0 * (self.ZO[0]) + self.b1 * (self.ZO[1]) *
self.x_arr[3][0] + self.b2 * (self.ZO[2]) * self.x_arr[3][
            1] + self.b3 * (self.ZO[3]) * self.x_arr[3][2]
        self.Yv = [self.Y1v, self.Y2v, self.Y3v, self.Y4v]
        self.criterionFisher()


    def criterionFisher(self):
        print("\nПеревірка адекватності за критерієм Фішера (M = {0}, N =
{1}):".format(self.m, self.n))
        self.d = 0
        for i in range(len(self.ZO)):
            if (self.ZO[i] == 1):
                self.d += 1
        print("Кількість значимих коефіцієнтів d = {0}".format(self.d))
        self.Yrazn = 0
        for i in range(self.n):
            self.Yrazn += pow((self.Yv[i] - self.yi_avg[i]), 2)
        self.Sad = ((self.m / (self.n - self.d)) * self.Yrazn)
        print("Sad = {0:.2f}".format(self.Sad))
        self.Fp = (self.Sad / self.Sb)
        print("FP = {0:.2f}".format(self.Fp))
        self.F4 = self.n - self.d
        print("F4 = N - d = {0} - {1} = {2} \nq = {3}".format(self.n, self.d,
self.F4, self.q))
        self.Ft = f.ppf(q=1 - 0.05, dfn=self.F4, dfd=self.F3)
```

```
        print("FT = {0}".format(self.Ft))
        if (self.Ft > self.Fp):
            print("FT = {0:.2f} > FP = {1:.2f} - рівняння регресії адекватно
оригіналу".format(self.Ft, self.Fp))
        if (self.Fp > self.Ft):
            print("FP = {0:.2f} > FT = {1:.2f} - рівняння регресії
неадекватно оригіналу".format(self.Fp, self.Ft))


if __name__ == '__main__':
    Window()
```

Результати роботи программи:



X1min	X1max	X2min	X2max	X3min	X3max	Ymin	Ymax
20	70	5	40	20	45	215	240

N	X0	X1	X2	X3	Y1	Y2	Y3
1	1	-1	-1	-1	221	228	236
2	1	-1	1	1	239	238	233
3	1	1	-1	1	236	217	240
4	1	1	1	-1	233	225	218

X1	X2	X3	Y1	Y2	Y3
20	5	20	221	228	236
20	40	45	239	238	233
70	5	45	236	217	240
70	40	20	233	225	218

```
C:\Users\YmkA\PycharmProjects\mope_labs\venv\Scripts\python.exe C:/Users/YmkA/PycharmProjects/mope_labs/lab3.py
Рівняння регресії: y = 224.28+(-0.09)*X1+(0.04)*X2+(0.28)*X3
Перевірка:
224.28+(-0.09)*(20.00)+(0.04)*(5.00)+(0.28)*(20.00) = 228.33 = Y1cep = 228.33
224.28+(-0.09)*(20.00)+(0.04)*(40.00)+(0.28)*(45.00) = 236.67 = Y2cep = 236.67
224.28+(-0.09)*(70.00)+(0.04)*(5.00)+(0.28)*(45.00) = 231.00 = Y3cep = 231.00
224.28+(-0.09)*(70.00)+(0.04)*(40.00)+(0.28)*(20.00) = 225.33 = Y4cep = 225.33

Перевірка рівномірності дисперсій за критерієм Кохрена (M = 3, N = 4):
F1 = M - 1 = 3 - 1 = 2
F2 = N = 4
q = 0.05
GP = 0.4014 < GT = 0.7679 - Дисперсія однорідна!

Перевірка значимості коефіцієнтів регресії згідно критерію Стьюдента (M = 3, N = 4):
Оцінки коефіцієнтів Bs: B1=230.33, B2=-2.17, B3=0.67, B4=3.50
Коефіцієнти ts: t1=121.55, t2=1.14, t3=0.35, t4=1.85
F3 = F1*F2 = 2*4 = 8
q = 0.05
t табличне = 2.306
Рівняння регресії: y = 224.28*(1)+(-0.09)*(0)*X1+(0.04)*(0)*X2+(0.28)*(0)*X3

Перевірка адекватності за критерієм Фішера (M = 3, N = 4):
Кількість значимих коефіцієнтів d = 1
Sad = 216.31
FP = 5.02
F4 = N - d = 4 - 1 = 3
q = 0.05
FT = 4.06618055135116
FP = 5.02 > FT = 4.07 - рівняння регресії неадекватно оригіналу
```

Висновок:

Проведено дробовий трьохфакторний експеримент. Складено матрицю планування, успішно знайдено коефіцієнти рівняння регресії, а також проведено 3 статистичні перевірки. Результати правильності роботи програми надані у звіті.

Контрольні запитання:

1. Що називається дробовим факторним експериментом?

Дробовий факторний експеримент – це скорочений експеримент від повного факторного експерименту.

2. Для чого потрібно розрахункове значення Кохрена?

Для перевірки дисперсії.

3. Для чого перевіряється критерій Стьюдента?

Для перевірки значущості коефіцієнтів рівняння регресії.

4. Чим визначається критерій Фішера і як його застосовувати?

Критерій Фішера застосовується для перевірки адекватності моделі оригіналу. Для цієї мети необхідно оцінити, наскільки відрізняються середні значення у вихідної величини, отриманої в точках факторного простору, і значення у, отриманого з рівняння регресії в тих самих точках факторного простору. Для цього використовують дисперсію адекватності.