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 № 5

з дисципліни «Методи оптимізації та планування експерименту» на тему

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

Виконав:
студент ІІ курсу ФІОТ
групи ІО-93
Бриль Владислав
Залікова – 9303
Номер у списку: 2

ПЕРЕВІРИВ:
Асистент 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. Варіанти вибираються по номеру в списку в журналі викладача.

$$y_{i\max} = 200 + x_{cp\max}$$
$$y_{i\min} = 200 + x_{cp\min}$$

где $x_{cp\max} = \frac{x_{1\max} + x_{2\max} + x_{3\max}}{3}$, $x_{cp\min} = \frac{x_{1\min} + x_{2\min} + x_{3\min}}{3}$

4. Розрахувати коефіцієнти рівняння регресії і записати його.

5. Провести 3 статистичні перевірки.

Варіант:

602	-9	1	-7	10	-1	2

Програмний код:

```
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 = ((-9, 1), (-7, 10), (-1, 2))

        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("\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")

        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] = -1 * 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):

        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 \nНеобхідно збільшити кількість дослідів')

if __name__ == '__main__':
    Experiment(15, 3)
```

Вивід програми:

```
C:\Users\Владислав\AppData\Local\Programs\Python\Python38\python.exe C:/Users/Владислав/PycharmProjects/MOPE_LAB_5/MOPE_LAB_5.py

Рівняння регресії з урахуванням квадратичних членів:
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


Матриця планування для n = 15, m = 3

X:
 [[   1   -9   -7   -1   63    9    7  -63   81   49    1]
 [   1    1   -7   -1   -7   -1    7    7    1   49    1]
 [   1   -9   10   -1  -90    9  -10   90   81  100    1]
 [   1    1   10   -1   10   -1  -10  -10    1  100    1]
 [   1   -9   -7    2   63  -18  -14  126   81   49    4]
 [   1    1   -7    2   -7    2  -14  -14    1   49    4]
 [   1   -9   10    2  -90  -18   20 -180   81  100    4]
 [   1    1   10    2   10    2   20   20    1  100    4]
 [   1    2    1    1    2    2    1    2    4    1    1]
 [   1  -10    1    1  -10  -10    1  -10  100    1    1]
 [   1   -4   11    1  -44   -4   11  -44   16  121    1]
 [   1   -4   -9    1   36   -4   -9   36   16   81    1]
 [   1   -4    1    2   -4   -8    2   -8   16    1    4]
 [   1   -4    1    0   -4    0    0    0   16    1    0]
 [   1   -4    1    1   -4   -4    1   -4   16    1    1]]

X нормоване:

[1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, -1.22, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 1.48, 0.0, 0.0]
[1.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48, 0.0, 0.0]
[1.0, 0.0, -1.22, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0, 1.48, 0.0]
[1.0, 0.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48, 0.0]
[1.0, 0.0, 0.0, -1.22, 0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 1.48]
[1.0, 0.0, 0.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48]
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
[1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, -1.22, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 1.48, 0.0, 0.0]
[1.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48, 0.0, 0.0]
[1.0, 0.0, -1.22, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0, 1.48, 0.0]
[1.0, 0.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48, 0.0]
[1.0, 0.0, 0.0, -1.22, 0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 1.48]
[1.0, 0.0, 0.0, 1.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.48]
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Y:
 [[202. 200. 200.]
 [200. 204. 197.]
 [200. 202. 198.]
 [198. 200. 195.]
 [197. 198. 203.]
 [203. 198. 199.]
 [197. 203. 200.]
 [200. 202. 201.]
 [197. 201. 202.]
 [200. 198. 196.]
 [197. 198. 197.]
 [201. 200. 200.]
 [198. 197. 196.]
 [198. 200. 201.]
 [202. 195. 198.]]

Коефіцієнти рівняння регресії:
[198.778, 0.254, -0.144, 0.24, -0.007, 0.074, 0.06, 0.005, 0.034, 0.012, -0.035]

Результат рівняння зі знайденими коефіцієнтами:
 [200.897 200.817 199.877 197.757 199.199 200.289 198.944 200.544 199.699
 199.051 198.831 199.871 198.03  198.202 198.151]

Середнє значення y: [200.667, 200.333, 200.0, 197.667, 199.333, 200.0, 200.0, 201.0, 200.0, 198.0, 197.333, 200.333, 197.0, 199.667, 198.333]
Дисперсія y: [0.889, 8.222, 2.667, 4.222, 6.889, 4.667, 6.0, 0.667, 4.667, 2.667, 0.222, 0.222, 0.667, 1.556, 8.222]

Перевірка за критерієм Кохрена
Gp = 0.15677077374823625
З ймовірністю 0.95 дисперсії однорідні.

Критерій Стьюдента:
```

```
Критерій Стьюдента:
 [715.035, 0.82, 0.473, 1.173, 0.398, 1.037, 1.196, 0.558, 522.951, 522.834, 522.481]

Коефіцієнти [0.254, -0.144, 0.24, -0.007, 0.074, 0.06, 0.005] статистично незначущі, тому ми виключаємо їх з рівняння.

Значення "y" з коефіцієнтами [198.778, 0.034, 0.012, -0.035]
[198.789, 198.789, 198.789, 198.789, 198.789, 198.789, 198.789, 198.789, 198.82819164999998, 198.82819164999998, 198.7957147, 198.7957147, 198.726332125, 198.726332125, 198.778]

Перевірка адекватності за критерієм Фішера
Fp = 2.1355974661456365
F_t = 2.125558760875511
Математична модель не адекватна експериментальним даним
Необхідно збільшити кількість дослідів

Process finished with exit code 0
```