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 експерименту» на тему

«ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ.»

ВИКОНАВ:
студент ІІ курсу ФІОТ
групи ІО-93
Корякін Єгор
Варіант: 315

ПЕРЕВІРИВ:
Регіда П. Г.

Київ – 2020

315	10	50	25	65	50	65

Лістинг програми

```
from random import *
import numpy as np
from numpy.linalg import solve
from scipy.stats import f, t
from functools import partial


class FractionalExperiment:


    def __init__(self, n, m):
        self.n = n
        self.m = m
        self.x_min = (10 + 25 + 50) / 3
        self.x_max = (50 + 65 + 65) / 3
        self.y_max = round(200 + self.x_max)
        self.y_min = round(200 + self.x_min)
        self.x_norm = [[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]]
        self.x_range = [(-40, 20), (-35, 15), (20, 25)]
        self.y = np.zeros(shape=(self.n, self.m))
        self.y_new = []
        for i in range(self.n):
            for j in range(self.m):
                self.y[i][j] = randint(self.y_min, self.y_max)
        self.y_av = [round(sum(i) / len(i), 2) for i in self.y]
        self.x_norm = self.x_norm[:len(self.y)]
        self.x = np.ones(shape=(len(self.x_norm), len(self.x_norm[0])))
        for i in range(len(self.x_norm)):
            for j in range(1, len(self.x_norm[i])):
                if self.x_norm[i][j] == -1:
                    self.x[i][j] = self.x_range[j - 1][0]
                else:
                    self.x[i][j] = self.x_range[j - 1][1]
        self.f1 = m - 1
        self.f2 = n
        self.f3 = self.f1 * self.f2
        self.q = 0.05

    def regression(self, x, b):

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

    def count_koefs(self):

        mx1 = sum(self.x[:, 1]) / self.n
        mx2 = sum(self.x[:, 2]) / self.n
        mx3 = sum(self.x[:, 3]) / self.n
        my = sum(self.y_av) / self.n
        a12 = sum([self.x[i][1] * self.x[i][2] for i in range(len(self.x))]) / self.n
```

```
        a13 = sum([self.x[i][1] * self.x[i][3] for i in range(len(self.x))]) / self.n
        a23 = sum([self.x[i][2] * self.x[i][3] for i in range(len(self.x))]) / self.n
        a11 = sum([i ** 2 for i in self.x[:, 1]]) / self.n
        a22 = sum([i ** 2 for i in self.x[:, 2]]) / self.n
        a33 = sum([i ** 2 for i in self.x[:, 3]]) / self.n
        a1 = sum([self.y_av[i] * self.x[i][1] for i in range(len(self.x))]) / self.n
        a2 = sum([self.y_av[i] * self.x[i][2] for i in range(len(self.x))]) / self.n
        a3 = sum([self.y_av[i] * self.x[i][3] for i in range(len(self.x))]) / self.n

        X = [[1, mx1, mx2, mx3], [mx1, a11, a12, a13], [mx2, a12, a22, a23], [mx3,
a13, a23, a33]]
        Y = [my, a1, a2, a3]
        B = [round(i, 2) for i in solve(X, Y)]
        print('\nРівняння регресії')
        print(f'y = {B[0]} + {B[1]}*x1 + {B[2]}*x2 + {B[3]}*x3')

        return B

    def dispersion(self):

        res = []
        for i in range(self.n):
            s = sum([(self.y_av[i] - self.y[i][j]) ** 2 for j in range(self.m)]) /
self.m
            res.append(s)
        return res

    def kohren(self):

        q1 = self.q / self.f1
        fisher_value = f.ppf(q=1 - q1, dfn=self.f2, dfd=(self.f1 - 1) * self.f2)
        G_cr = fisher_value / (fisher_value + self.f1 - 1)
        s = self.dispersion()
        Gp = max(s) / sum(s)
        return Gp, G_cr

    def student(self):


        def bs():
            res = [sum(1 * y for y in self.y_av) / self.n]
            for i in range(3):  # 4 - кість факторів
                b = sum(j[0] * j[1] for j in zip(self.x[:, i], self.y_av)) / self.n
                res.append(b)
            return res

        S_kv = self.dispersion()
        s_kv_aver = sum(S_kv) / self.n

        # статистична оцінка дисперсії
        s_Bs = (s_kv_aver / self.n / self.m) ** 0.5
        Bs = bs()
        ts = [abs(B) / s_Bs for B in Bs]
        return ts

    def fisher(self, d):

        S_ad = self.m / (self.n - d) * sum([(self.y_new[i] - self.y_av[i]) ** 2 for i
in range(len(self.y))])
        S_kv = self.dispersion()
        S_kv_aver = sum(S_kv) / self.n
        F_p = S_ad / S_kv_aver
```

```
        return F_p

    def check(self):

        student = partial(t.ppf, q=1 - 0.025)
        t_student = student(df=self.f3)

        print('\nПеревірка за критерієм Кохрена')
        Gp, G_kr = self.kohren()
        print(f'Gp = {Gp}')
        if Gp < G_kr:
            print(f'З ймовірністю {1-self.q} дисперсії однорідні.')
        else:
            print("Необхідно збільшити кількість дослідів")
            self.m += 1
            FractionalExperiment(self.n, self.m)

        ts = self.student()
        print('\nПеревірка значущості коефіцієнтів за критерієм Стьюдента')
        print('Критерій Стьюдента:\n', ts)
        res = [t for t in ts if t > t_student]
        B = self.count_koefs()
        final_k = [B[ts.index(i)] for i in ts if i in res]
        print('Коефіцієнти {} статистично незначущі, тому ми виключаємо їх з
рівняння.'.format(
            [i for i in B if i not in final_k]))

        for j in range(self.n):
            self.y_new.append(self.regression([self.x[j][ts.index(i)] for i in ts if
i in res], final_k))

        print(f'\nЗначення "y" з коефіцієнтами {final_k}')
        print(self.y_new)

        d = len(res)
        f4 = self.n - d
        F_p = self.fisher(d)

        fisher = partial(f.ppf, q=1 - 0.05)
        f_t = fisher(dfn=f4, dfd=self.f3)  # табличне знач
        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')


experiment = FractionalExperiment(7, 8)
experiment.check()
```

Результати виконання програми:

```
Перевірка за критерієм Кохрена
Gp = 0.24079872277151962
З ймовірністю 0.95 дисперсії однорідні.

Перевірка значущості коефіцієнтів за критерієм Стьюдента
Критерій Стьюдента:
 [221.37919208666938, 221.37919208666938, 3245.622286511892, 3010.0454584541526]

Рівняння регресії
y = 257.94 + -0.12*x1 + -0.04*x2 + -0.65*x3
Коефіцієнти [-0.12] статистично незначущі, тому ми виключаємо їх з рівняння.

Значення "y" з коефіцієнтами [257.94, 257.94, -0.04, -0.65]
[504.28, 499.03, 501.03, 502.28, 501.03, 502.28, 504.28]

Перевірка адекватності за критерієм Фішера
Fp = 17734.802629734022
F_t = 2.7939488515842408
Математична модель не адекватна експериментальним даним
```