Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Методи наукових досліджень

Лабораторна робота №5

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

Виконав:

студент групи ІВ-92

Сударєв Артем Анатолійович

Номер у списку групи – 23

Перевірив:

Регіда П. Г.

Київ 2021 р.

Мета: провести трьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Завдання:

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:

223	-10	10	-10	2	-10	4

Лістинг програми:

```
import random
import sklearn.linear_model as lm
from scipy.stats import f, t
from functools import partial
from pyDOE2 import *
from numpy import average
import pandas as pd
from tabulate import tabulate


class Criteria:
    def __init__(self, x, y, n, m):
        self.x = x
        self.y = y

        self.n = n
        self.m = m
        self.f1 = self.m - 1
        self.f2 = self.n
        self.q = 0.05
        self.q1 = self.q / self.f1

    def s_kv(self, y_aver):
        res = []
        for i in range(self.n):
            s = sum([(y_aver[i] - self.y[i][j]) ** 2 for j in range(self.m)]) /
self.m
            res.append(round(s, 3))
        return res
```

```
    def criteria_cochrana(self, y_aver):
        S_kv = self.s_kv(y_aver)
        Gp = max(S_kv) / sum(S_kv)
        print('\nПеревірка за критерієм Кохрена')
        return Gp

    def cohren(self):
        fisher_value = f.ppf(q=1 - self.q1, dfn=self.f2, dfd=(self.f1 - 1) * self.f2)
        return fisher_value / (fisher_value + self.f1 - 1)

    def bs(self, x, y_aver):
        res = [sum(y_aver) / self.n]

        for i in range(len(x[0])):
            b = sum(j[0] * j[1] for j in zip(x[:, i], y_aver)) / self.n
            res.append(b)

        return res

    def criteria_studenta(self, x, y_aver):
        S_kv = self.s_kv(y_aver)
        s_kv_aver = sum(S_kv) / self.n

        s_Bs = (s_kv_aver / self.n / self.m) ** 0.5
        Bs = self.bs(x, y_aver)
        ts = [round(abs(B) / s_Bs, 3) for B in Bs]

        return ts

    def criteria_fishera(self, y_aver, y_new, d):
        S_ad = self.m / (self.n - d) * sum([(y_new[i] - y_aver[i]) ** 2 for i in
range(len(self.y))])
        S_kv = self.s_kv(y_aver)
        S_kv_aver = sum(S_kv) / self.n

        return S_ad / S_kv_aver


class Experiment:

    def __init__(self, x1, x2, x3):

        self.n, self.m = 15, 6

        self.x, self.y, self.x_norm = None, None, None
        self.y_average = None

        self.b = None

        self.x_range = (x1, x2, x3)

        self.x_aver_max = average([x[1] for x in self.x_range])
        self.x_aver_min = average([x[0] for x in self.x_range])

        self.y_max = 200 + int(self.x_aver_max)
        self.y_min = 200 + int(self.x_aver_min)

    @staticmethod
    def add_sq_nums(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 count_y_average(self):
        self.y_average = [round(sum(i) / len(i), 3) for i in self.y]

    def regression(self, x, b):
        return sum([x[i] * b[i] for i in range(len(x))])

    def count_b_coeficient(self):
        # X5, y5_aver
        skm = lm.LinearRegression(fit_intercept=False)
        skm.fit(self.x, self.y_average)
        self.b = skm.coef_

        print('\nКоефіцієнти рівняння регресії:')
        self.b = [round(i, 3) for i in self.b]
        print("y = {} +{}*x1 +{}*x2 +{}*x3 + {}*x1*x2 + {}*x1*x3 + {}*x2*x3 +
b{}*x1*x2*x3 + {}x1^2 + {}x2^2 + {}x3^2\n".format(*self.b))
        print('\nРезультат рівняння зі знайденими коефіцієнтами:\n', np.dot(self.x,
self.b))

    def run_check(self):
        criteria = Criteria(self.x, self.y, self.n, self.m)

        print('\n\tПеревірка рівняння:')
        f1 = self.m - 1
        f2 = self.n
        f3 = f1 * f2
        q = 0.05

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

        G_kr = criteria.cohren()

        y_aver = [round(sum(i) / len(i), 3) for i in self.y]
        print('\nСереднє значення y:', y_aver)

        disp = criteria.s_kv(y_aver)
        print('Дисперсія y:', disp)

        Gp = criteria.criteria_cochrana(y_aver)
        print(f'Gp = {Gp}')
        if Gp < G_kr:
            print(f'З ймовірністю {1 - q} дисперсії однорідні.')
        else:
            print("Необхідно збільшити кількість дослідів")
            self.m += 1
            new_exp = Experiment(*self.x_range)
            new_exp.run_experiment(self.n, self.m)

        ts = criteria.criteria_studenta(self.x_norm[:, 1:], y_aver)
        print('\nКритерій Стьюдента:\n', ts)
        res = [t for t in ts if t > t_student]
        final_k = [self.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 self.b if i not in final_k]))

        y_new = []
        for j in range(self.n):
            y_new.append(self.regression([self.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 >= self.n:
            print('\nF4 <= 0')

        f4 = self.n - d

        F_p = criteria.criteria_fishera(y_aver, y_new, 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')

    def fill_y(self):
        self.y = np.zeros(shape=(self.n, self.m))
        for i in range(self.n):
            for j in range(self.m):
                self.y[i][j] = random.randint(self.y_min, self.y_max)

    def make_matrix(self):
        self.fill_y()
        if self.n > 14:
            no = self.n - 14
        else:
            no = 1

        self.x_norm = ccdesign(3, center=(0, no))
        self.x_norm = np.insert(self.x_norm, 0, 1, axis=1)

        for i in range(4, 11):
            self.x_norm = np.insert(self.x_norm, i, 0, axis=1)

        l = 1.215

        for i in range(len(self.x_norm)):
            for j in range(len(self.x_norm[i])):
                if self.x_norm[i][j] < -1 or self.x_norm[i][j] > 1:
                    if self.x_norm[i][j] < 0:
                        self.x_norm[i][j] = -l
                    else:
                        self.x_norm[i][j] = l

        self.x_norm = self.add_sq_nums(self.x_norm)

        self.x = np.ones(shape=(len(self.x_norm), len(self.x_norm[0])),
dtype=np.int64)
        for i in range(8):
```

```
            for j in range(1, 4):
                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]

        for i in range(8, len(self.x)):
            for j in range(1, 3):
                self.x[i][j] = (self.x_range[j - 1][0] + self.x_range[j - 1][1]) / 2

        dx = [self.x_range[i][1] - (self.x_range[i][0] + self.x_range[i][1]) / 2 for
i in range(3)]

        self.x[8][1] = l * dx[0] + self.x[9][1]
        self.x[9][1] = -l * dx[0] + self.x[9][1]
        self.x[10][2] = l * dx[1] + self.x[9][2]
        self.x[11][2] = -l * dx[1] + self.x[9][2]
        self.x[12][3] = l * dx[2] + self.x[9][3]
        self.x[13][3] = -l * dx[2] + self.x[9][3]
        self.x = self.add_sq_nums(self.x)

        show_arr = pd.DataFrame(self.x)
        print('\nX:\n', tabulate(show_arr, headers='keys', tablefmt='psql'))

        show_arr = pd.DataFrame(self.x_norm)
        print('\nX нормоване:\n', tabulate(show_arr.round(0), headers='keys',
tablefmt='psql'))

        show_arr = pd.DataFrame(self.y)
        print('\nY:\n', tabulate(show_arr, headers='keys', tablefmt='psql'))

    def run_experiment(self, n=None, m=None):
        if n is None and m is None:
            pass
        else:
            self.n = n
            self.m = m

        self.make_matrix()
        self.count_y_average()

        self.count_b_coeficient()
        self.run_check()


if __name__ == '__main__':
    experiment = Experiment((-10, 10), (-10, 2),(-10, 4))

    experiment.run_experiment(15, 3)
```