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 при використанні рівняння регресії з урахуванням ефекту взаємодії.

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

ПЕРЕВІРИВ:
асистент
Регіда П.Г.

Київ 2021 р.

Мета:

Провести повний трьохфакторний експеримент. Знайти рівняння регресії адекватне об'єкту.

Завдання на лабораторну роботу

1. Скласти матрицю планування для повного трьохфакторного експерименту.
2. Провести експеримент, повторивши N раз досліди у всіх точках факторного простору і знайти значення відгуку Y. Знайти значення Y шляхом моделювання випадкових чисел у певному діапазоні відповідно варіанту. Варіанти вибираються за номером в списку в журналі викладача.

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

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

3. Знайти коефіцієнти рівняння регресії і записати його.
4. Провести 3 статистичні перевірки – за критеріями Кохрена, Стьюдента, Фішера.
5. Зробити висновки по адекватності регресії та значимості окремих коефіцієнтів і записати скореговане рівняння регресії.
6. Написати комп'ютерну програму, яка усе це моделює.

Варіант завдання:

N	X1		X2		X3	
	min	max	min	max	min	max
217	-10	50	25	65	-10	15

Розруківка коду програми:

```
from tkinter import *
import numpy as np
import plotly.figure_factory as ff
from scipy.stats import f, t
from random import *
from functools import reduce
from _pydecimal import Decimal
from itertools import compress
import math

class Window:
    def __init__(self):
        self.window = Tk()
        self.window.title("МОПЕ Лабораторна робота №4")
        self.x1_min, self.x1_max = -10, 50
        self.x2_min, self.x2_max = 25, 65
        self.x3_min, self.x3_max = -10, 15
        self.m, self.n = 3, 8
        self.x_min_avg = round((self.x1_min + self.x2_min + self.x3_min) / 3,
2)
        self.x_max_avg = round((self.x1_max + self.x2_max + self.x3_max) / 3,
2)
        self.y_min = round(200 + self.x_min_avg)
        self.y_max = round(200 + self.x_max_avg)
        self.norm_factor_table = [[-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, -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.var_factor_table = [[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.x1_min, self.x2_min, self.x3_max],
                                 [self.x1_min, self.x2_max, self.x3_min],
                                 [self.x1_max, self.x2_min, self.x3_min],
                                 [self.x1_max, self.x2_max, self.x3_max]]
        for i in range(len(self.var_factor_table)):
            self.var_factor_table[i].append(self.var_factor_table[i][0] *
self.var_factor_table[i][1])
            self.var_factor_table[i].append(self.var_factor_table[i][0] *
self.var_factor_table[i][2])
            self.var_factor_table[i].append(self.var_factor_table[i][1] *
self.var_factor_table[i][2])
            self.var_factor_table[i].append(self.var_factor_table[i][0] *
self.var_factor_table[i][1] * self.var_factor_table[i][2])

        self.y_arr = [[randint(self.y_min, self.y_max) for _ in
range(self.m)] for j in range(self.n)]
        self.x1 = np.array(list(zip(*self.var_factor_table))[0])
        self.x2 = np.array(list(zip(*self.var_factor_table))[1])
        self.x3 = np.array(list(zip(*self.var_factor_table))[2])
        self.yi = np.array([np.average(i) for i in self.y_arr])
        self.variant_b = Button(self.window, text='Матриця
планування\nдля\nповного трьохфакторного експерименту', width=30,
command=self.getDataForTable, bg='Gold')
```

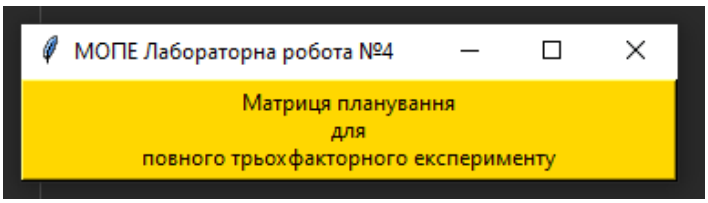
```
        self.variant_b.grid(row=0, column=0, columnspan=2)
        self.coeffs = [[self.n, self.m_ij(self.x1), self.m_ij(self.x2),
self.m_ij(self.x3),
                        self.m_ij(self.x1 * self.x2),
                        self.m_ij(self.x1 * self.x3), self.m_ij(self.x2 *
self.x3),
                        self.m_ij(self.x1 * self.x2 * self.x3)],
                        [self.m_ij(self.x1), self.m_ij(self.x1 ** 2),
self.m_ij(self.x1 * self.x2),
                         self.m_ij(self.x1 * self.x3),
                        self.m_ij(self.x1 ** 2 * self.x2), self.m_ij(self.x1
** 2 * self.x3),
                        self.m_ij(self.x1 * self.x2 * self.x3),
self.m_ij(self.x1 ** 2 * self.x2 * self.x3)],
                        [self.m_ij(self.x2), self.m_ij(self.x1 * self.x2),
self.m_ij(self.x2 ** 2),
                         self.m_ij(self.x2 * self.x3),
                        self.m_ij(self.x1 * self.x2 ** 2), self.m_ij(self.x1
* self.x2 * self.x3),
                        self.m_ij(self.x2 ** 2 * self.x3), self.m_ij(self.x1
* self.x2 ** 2 * self.x3)],
                        [self.m_ij(self.x3), self.m_ij(self.x1 * self.x3),
self.m_ij(self.x2 * self.x3),
                         self.m_ij(self.x3 ** 2),
                        self.m_ij(self.x1 * self.x2 * self.x3),
self.m_ij(self.x1 * self.x3 ** 2),
                         self.m_ij(self.x2 * self.x3 ** 2), self.m_ij(self.x1
* self.x2 * self.x3 ** 2)],
                        [self.m_ij(self.x1 * self.x2), self.m_ij(self.x1 ** 2
* self.x2),
                         self.m_ij(self.x1 * self.x2 ** 2),
                        self.m_ij(self.x1 * self.x2 * self.x3),
self.m_ij(self.x1 ** 2 * self.x2 ** 2),
                        self.m_ij(self.x1 ** 2 * self.x2 * self.x3),
self.m_ij(self.x1 * self.x2 ** 2 * self.x3),
                        self.m_ij(self.x1 ** 2 * self.x2 ** 2 * self.x3)],
                        [self.m_ij(self.x1 * self.x3), self.m_ij(self.x1 ** 2
* self.x3),
                         self.m_ij(self.x1 * self.x2 * self.x3),
                        self.m_ij(self.x1 * self.x3 ** 2), self.m_ij(self.x1
** 2 * self.x2 * self.x3),
                        self.m_ij(self.x1 ** 2 * self.x3 ** 2),
self.m_ij(self.x1 * self.x2 * self.x3 ** 2),
                        self.m_ij(self.x1 ** 2 * self.x2 * self.x3 ** 2)],
                        [self.m_ij(self.x2 * self.x3), self.m_ij(self.x1 *
self.x2 * self.x3),
                         self.m_ij(self.x2 ** 2 * self.x3),
                        self.m_ij(self.x2 * self.x3 ** 2), self.m_ij(self.x1
* self.x2 ** 2 * self.x3),
                        self.m_ij(self.x1 * self.x2 * self.x3 ** 2),
self.m_ij(self.x2 ** 2 * self.x3 ** 2),
                        self.m_ij(self.x1 * self.x2 ** 2 * self.x3 ** 2)],
                        [self.m_ij(self.x1 * self.x2 * self.x3),
self.m_ij(self.x1 ** 2 * self.x2 * self.x3),
                        self.m_ij(self.x1 * self.x2 ** 2 * self.x3),
self.m ij(self.x1 * self.x2 * self.x3 ** 2),
                        self.m_ij(self.x1 ** 2 * self.x2 ** 2 * self.x3),
self.m_ij(self.x1 ** 2 * self.x2 * self.x3 ** 2),
                        self.m_ij(self.x1 * self.x2 ** 2 * self.x3 ** 2),
                        self.m_ij(self.x1 ** 2 * self.x2 ** 2 * self.x3 **
2)]]

        self.free_vals = [self.m_ij(self.yi), self.m_ij(self.yi * self.x1),
self.m_ij(self.yi * self.x2),
```

```
                            self.m_ij(self.yi * self.x3), self.m_ij(self.yi *
self.x1 * self.x2),
                            self.m_ij(self.yi * self.x1 * self.x3),
                            self.m_ij(self.yi * self.x2 * self.x3),
self.m_ij(self.yi * self.x1 * self.x2 * self.x3)]

        self.natural_bi = np.linalg.solve(self.coeffs, self.free_vals)

        self.natural_x1 = np.array(list(zip(*self.norm_factor_table))[0])
        self.natural_x2 = np.array(list(zip(*self.norm_factor_table))[1])
        self.natural_x3 = np.array(list(zip(*self.norm_factor_table))[2])

        self.norm_bi = [self.m_ij(self.yi),
                        self.m_ij(self.yi * self.natural_x1),
                        self.m_ij(self.yi * self.natural_x2),
                        self.m_ij(self.yi * self.natural_x3),
                        self.m_ij(self.yi * self.natural_x1 * self.natural_x2),
                        self.m_ij(self.yi * self.natural_x1 * self.natural_x3),
                        self.m_ij(self.yi * self.natural_x2 * self.natural_x3),
                        self.m_ij(self.yi * self.natural_x1 * self.natural_x2 *
self.natural_x3)]

        print("\nm = {}, n = {}\n".format(self.m, self.n))
        while not self.cochran_cr(self.m, 4, self.y_arr):
            self.m += 1
            y_table = [[randint(self.y_min, self.y_max) for _ in
range(self.m)] for j in range(self.n)]
        self.norm_factors_table_zero_factor = [[+1] + i for i in
self.norm_factor_table]
        self.importance = self.student_criteria(self.m, self.n, self.y_arr,
self.norm_factors_table_zero_factor)

        self.fisher_criteria(self.m, self.n, 1, self.var_factor_table,
self.y_arr, self.natural_bi, self.importance)
        self.window.mainloop()


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

    def getDataForTable(self):
        data = [['X1', 'X2', 'X3', 'X12', 'X13', 'X23', 'X123', 'Y1', 'Y2',
'Y3']]
        for i in range(len(self.var_factor_table)):
            data.append(self.var_factor_table[i])
            data[i+1] += self.y_arr[i]
        self.make_table(data)


    def m_ij(self, *arrays):
        return np.average(reduce(lambda accum, el: accum * el, arrays))

    def cochran_cr(self, m, N, y_table):
        print("Перевірка рівномірності дисперсій за критерієм Кохрена: ")
        y variations = [np.var(i) for i in y table]
        max_y_variation = max(y_variations)
        gp = max_y_variation / sum(y_variations)
        f1 = m - 1
        f2 = N
        p = 0.95
        q = 1 - p
        gt = self.Cochran_val(f1, f2, q)
        print("Gp = {:.3f}; Gt = {:.3f}".format(gp, gt))
```

```
        if gp < gt:
            print("Gp < Gt => дисперсії рівномірні")
            return True
        else:
            print("Gp > Gt => дисперсії нерівномірні - Потрібно додати
експерименти")
            return False

    def student_criteria(self, m, N, y_table, normalized_x_table: "with zero
factor!"):
        print("\nПеревірка значимості коефіцієнтів регресії за критерієм
Стьюдента: ")
        average_variation = np.average(list(map(np.var, y_table)))

        y_averages = np.array(list(map(np.average, y_table)))
        variation_beta_s = average_variation / N / m
        standard_deviation_beta_s = math.sqrt(variation_beta_s)
        x_i = np.array([[el[i] for el in normalized_x_table] for i in
range(len(normalized_x_table))])
        coefficients_beta_s = np.array([round(np.average(y_averages *
x_i[i]), 3) for i in range(len(x_i))])
        print("Оцінки коефіцієнтів βs: " + ", ".join(list(map(str,
coefficients_beta_s))))
        t_i = np.array(
            [abs(coefficients_beta_s[i]) / standard_deviation_beta_s for i in
range(len(coefficients_beta_s))])
        print("Коефіцієнти ts: " + ", ".join(list(map(lambda i:
"{:.2f}".format(i), t_i))))
        f3 = (m - 1) * N
        q = 0.05

        t = self.Student_val(f3, q)
        importance = [True if el > t else False for el in list(t_i)]

        print("Табличне значення критерія Стьюдента: {}".format(t))
        beta_i = ["β0", "β1", "β2", "β3", "β12", "β13", "β23", "β123"]
        importance_to_print = ["важливий" if i else "неважливий" for i in
importance]
        to_print = map(lambda x: x[0] + " " + x[1], zip(beta_i,
importance_to_print))
        x_i_names = list(compress(["", "X1", "X2", "X3", "X12", "X13", "X23",
"X123"], importance))
        betas_to_print = list(compress(coefficients_beta_s, importance))
        print(*to_print, sep="; ")
        equation = " ".join(
            ["".join(i) for i in zip(list(map(lambda x: "{:+.2f}".format(x),
betas_to_print)), x_i_names)])
        print("Рівняння регресії без незначимих членів: y = " + equation)
        return importance

    def calculate_theoretical_y(self, x_table, b_coefficients, importance):
        x_table = [list(compress(row, importance)) for row in x_table]
        b_coefficients = list(compress(b_coefficients, importance))
        y_vals = np.array([sum(map(lambda x, b: x * b, row, b_coefficients))
for row in x_table])
        return y_vals

    def fisher_criteria(self, m, N, d, naturalized_x_table, y_table,
b_coefficients, importance):
        print("\nПеревірка адекватності моделі за критерієм Фішера:")
        f3 = (m - 1) * N
        f4 = N - d
        q = 0.05
        theoretical_y = self.calculate_theoretical_y(naturalized_x_table,
```

```
b_coefficients, importance)
        theoretical_values_to_print = list(
            zip(map(lambda x: "x1 = {0[1]}, x2 = {0[2]}, x3 =
{0[3]}".format(x), naturalized_x_table), theoretical_y))
        print("Теоретичні значення y для різних комбінацій факторів:")
        print("\n".join(["{arr[0]}: y = {arr[1]:.2f}".format(arr=el) for el
in theoretical_values_to_print]))
        y_averages = np.array(list(map(np.average, y_table)))
        s_ad = m / (N - d) * (sum((theoretical_y - y_averages) ** 2))
        y_variations = np.array(list(map(np.var, y_table)))
        s_v = np.average(y_variations)
        f_p = float(s_ad / s_v)
        f_t = self.Fisher_val(f3, f4, q)
        print("\nFp = {:.3f}, Ft = {:.3f}".format(f_p, f_t))
        print("Fp < Ft => МАТЕМАТИЧНА МОДЕЛЬ АДЕКВАТНА" if f_p < f_t else "Fp
> Ft => МАТЕМАТИЧНА МОДЕЛЬ НЕАДЕКВАТНА")
        return True if f_p < f_t else False


    def Cochran_val(self, f1, f2, q):
        partResult1 = q / f2
        params = [partResult1, f1, (f2 - 1) * f1]
        fisher = f.isf(*params)
        result = fisher / (fisher + (f2 - 1))
        return Decimal(result).quantize(Decimal('.0001')).__float__()

    def Student_val(self, f3, q):
        return Decimal(abs(t.ppf(q / 2,
f3))).quantize(Decimal('.0001')).__float__()

    def Fisher_val(self, f3, f4, q):
        return Decimal(abs(f.isf(q, f4,
f3))).quantize(Decimal('.0001')).__float__()


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

Результати роботи программи:



X1	X2	X3	X12	X13	X23	X123	Y1	Y2	Y3
-10	25	-10	-250	100	-250	2500	208	221	243
-10	65	15	-650	-150	975	-9750	215	216	203
50	25	15	1250	750	375	18750	229	226	231
50	65	-10	3250	-500	-650	-32500	230	211	241
-10	25	15	-250	-150	375	-3750	236	205	233
-10	65	-10	-650	100	-650	6500	236	220	213
50	25	-10	1250	-500	-250	-12500	209	241	224
50	65	15	3250	750	975	48750	202	231	209

```
m = 3, n = 8

Перевірка рівномірності дисперсій за критерієм Кохрена:
Gp = 0.206; Gt = 0.768
Gp < Gt => дисперсії рівномірні

Перевірка значимості коефіцієнтів регресії за критерієм Стьюдента:
Оцінки коефіцієнтів βs: 222.208, 1.458, -3.292, -2.542, 0.292, 0.208, -3.708, -0.625
Коефіцієнти ts: 96.77, 0.63, 1.43, 1.11, 0.13, 0.09, 1.61, 0.27
Табличне значення критерія Стьюдента: 2.1199
β0 важливий; β1 неважливий; β2 неважливий; β3 неважливий; β12 неважливий; β13 неважливий; β23 неважливий; β123 неважливий
Рівняння регресії без незначимих членів: y = +222.21

Перевірка адекватності моделі за критерієм Фішера:
Теоретичні значення y для різних комбінацій факторів:
x1 = 25, x2 = -10, x3 = -250: y = -35.59
x1 = 65, x2 = 15, x3 = -650: y = -35.59
x1 = 25, x2 = 15, x3 = 1250: y = 177.94
x1 = 65, x2 = -10, x3 = 3250: y = 177.94
x1 = 25, x2 = 15, x3 = -250: y = -35.59
x1 = 65, x2 = -10, x3 = -650: y = -35.59
x1 = 25, x2 = -10, x3 = 1250: y = 177.94
x1 = 65, x2 = 15, x3 = 3250: y = 177.94

Fp = 919.256, Ft = 2.657
Fp > Ft => МАТЕМАТИЧНА МОДЕЛЬ НЕАДЕКВАТНА
```

Висновок:

Успішно проведено трьохфакторний експеримент. Складено матрицю планування. Результати виконання лабораторної роботи надані у звіті.