МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний технічний університет України
«Київський Політехнічний Інститут»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №6

з дисципліни «Методи оптимізації і планування експерименту»

на тему“Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами”

Виконав:
студент 2-го курсу ФІОТ
групи ІО-82
Шевчук О. Б.
Номер у списку: 24

Перевірив:
Регіда П. Г.

Київ – 2020

Лабораторна робота №6

Варіант

224	-20	15	25	45	-20	-15	0,4+1,8*x1+6,2*x2+9,1*x3+3,7*x1*x1+0,1*x2*x2+9,0*x3*x3+1,9*x1*x2+0,8*x1*x3+9,9*x2*x3+4,7*x1*x2*x3

Лістинг коду програми

```
import numpy as np
from scipy.stats import t, f
from itertools import product, combinations
import random

np.set_printoptions(formatter={"float_kind": lambda x: '%.2f'%(x)})

def normalise(factors, def_matrx):
    X0 = np.mean(def_matrx, axis=1)
    delta = np.array([(def_matrx[i, 1] - X0[i]) for i in range(len(factors[0]))])
    X_norm = np.array(
        [[round((factors[i, j] - X0[j]) / delta[j], 3) for j in range(len(factors[i]))]
         for i in range(len(factors))])
    return X_norm


def cohran(Y_matrix, N):
    fish = fisher(0.95, N, m, 1)
    mean_Y = np.mean(Y_matrix, axis=1)
    dispersion_Y = np.mean((Y_matrix.T - mean_Y) ** 2, axis=0)
    Gp = np.max(dispersion_Y) / (np.sum(dispersion_Y))
    Gt = fish / (fish + (m - 1) - 2)
    return Gp < Gt


def student(prob, n, m):
    x_vec = [i*0.0001 for i in range(int(5/0.0001))]
    par = 0.5 + prob/0.1*0.05
    f3 = (m - 1) * n
    for i in x_vec:
        if abs(t.cdf(i, f3) - par) < 0.000005:
            return i


def students_t_test(norm_matrix, Y_matrix, N):
    mean_Y = np.mean(Y_matrix, axis=1)
    dispersion_Y = np.mean((Y_matrix.T - mean_Y) ** 2, axis=0)
    mean_dispersion = np.mean(dispersion_Y)
    sigma = np.sqrt(mean_dispersion / (N * m))
    beta = np.mean(norm_matrix.T * mean_Y, axis=1)
    t = np.abs(beta) / sigma
    if (m - 1) * N > 32:
        return np.where(t > student(0.95, N, m))
    return np.where(t > student(0.95, N, m))


def fisher(prob, n, m, d):
    x_vec = [i*0.001 for i in range(int(10/0.001))]
    f3 = (m - 1) * n
    for i in x_vec:
```

```
        if abs(f.cdf(i, n-d, f3) - prob) < 0.0001:
            return i


def fisher_criterion(Y_matrix, d, N):
    if d == N:
        return False
    sad = m / (N - d) * np.mean(check1 - mean_Y)
    mean_dispersion = np.mean(np.mean((Y_matrix.T - mean_Y) ** 2, axis=0))
    Fp = sad / mean_dispersion
    if (m - 1) * N > 32:
        if N - d > 6:
            return fisher(0.95, N, m, d)
        return Fp < fisher(0.95, N, m, d)
    if N - d > 6:
        return Fp < fisher(0.95, N, m, d)
    return Fp < fisher(0.95, N, m, d)


def make_plan_matrix_from_norm_matrix(norm_matrix):
    plan_matrix = np.empty((len(norm_matrix), len(norm_matrix[0])), dtype=np.float)
    for i in range(len(norm_matrix)):
        for j in range(len(norm_matrix[i])):
            if norm_matrix[i, j] == -1:
                plan_matrix[i, j] = def_matrx[j-1][0]
            elif norm_matrix[i, j] == 1 and j != 0:
                plan_matrix[i, j] = def_matrx[j-1][1]
            elif norm_matrix[i, j] == 1 and j == 0:
                plan_matrix[i, j] = 1
            else:
                mean = np.mean(def_matrx[j-1])
                plan_matrix[i, j] = norm_matrix[i, j] * (def_matrx[j-1][1] - mean) + mean
    return plan_matrix


def make_linear_equation():
    norm_matrix = np.array(list(product("01", repeat=3)), dtype=np.int)
    norm_matrix[norm_matrix == 0] = -1
    norm_matrix = np.insert(norm_matrix, 0, 1, axis=1)
    plan_matrix = make_plan_matrix_from_norm_matrix(norm_matrix)
    return norm_matrix, plan_matrix


def make_equation_with_interaction_effect(current_norm_matrix, current_plan_matrix):
    plan_matr = current_plan_matrix
    norm_matrix = current_norm_matrix
    combination = list(combinations(range(1, 4), 2))
    for i in combination:
        plan_matr = np.append(plan_matr, np.reshape(plan_matr[:, i[0]] * plan_matr[:, i[1]], (len(norm_matrix),
1)),axis=1)
        norm_matrix = np.append(norm_matrix, np.reshape(norm_matrix[:, i[0]] * norm_matrix[:, i[1]],
(len(norm_matrix), 1)), axis=1)
    plan_matr = np.append(plan_matr, np.reshape(plan_matr[:, 1] * plan_matr[:, 2] * plan_matr[:, 3],
(len(norm_matrix), 1)), axis=1)
    norm_matrix = np.append(norm_matrix, np.reshape(norm_matrix[:, 1] * norm_matrix[:, 2] * norm_matrix[:, 3],
(len(norm_matrix), 1)), axis=1)
    return norm_matrix, plan_matr
```

```
def make_equation_with_quadratic_terms(current_norm_matrix):
    norm_matrix_second_part = np.empty((3, 7))
    key = 0
    for i in range(3):
        j = 0
        while j < 7:
            if j == key:
                norm_matrix_second_part[i][key] = -1.215
                norm_matrix_second_part[i][key + 1] = 1.215
                j += 1
            else:
                norm_matrix_second_part[i][j] = 0
            j += 1
        key += 2
    norm_matrix_second_part = np.insert(norm_matrix_second_part, 0, 1, axis=0)
    norm_matrix = np.append(current_norm_matrix, norm_matrix_second_part.T, axis=0)
    plan_matrix = make_plan_matrix_from_norm_matrix(norm_matrix)
    plan_matrix = make_equation_with_interaction_effect(norm_matrix, plan_matrix)[1]
    plan_matrix = np.append(plan_matrix, plan_matrix[:, 1:4] ** 2, axis=1)
    norm_matrix = make_equation_with_interaction_effect(norm_matrix, plan_matrix)[0]
    norm_matrix = np.append(norm_matrix, norm_matrix[:, 1:4] ** 2, axis=1)
    return norm_matrix, plan_matrix



gt = {12: {1: 0.5410, 2: 0.3924, 3: 0.3264, 4: 0.2880, 5: 0.2624, 6: 0.2439, 7: 0.2299, 8: 0.2187, 9: 0.2098, 10:
0.2020},
      15: {1: 0.4709, 2: 0.3346, 3: 0.2758, 4: 0.2419, 5: 0.2159, 6: 0.2034, 7: 0.1911, 8: 0.1815, 9: 0.1736, 10: 0.1671}}
tt = {24: 2.064, 30: 2.042, 32: 1.96}  # m = [3, 6]
ft = {1: 4.2, 2: 3.3, 3: 2.9, 4: 2.7, 5: 2.5, 6: 2.4}
def_matrx = np.array([[-20, 15], [25, 45], [-20, -15]])
m = 3
count = 0
flag_of_model = False

while not flag_of_model:
    norm_matrix = make_linear_equation()[0]
    plan_matr = make_linear_equation()[1]
    if count == 1:
        norm_matrix = make_equation_with_interaction_effect(norm_matrix, plan_matr)[0]
        plan_matr = make_equation_with_interaction_effect(norm_matrix, plan_matr)[1]
    elif count > 1:
        plan_matr = make_equation_with_quadratic_terms(norm_matrix)[1]
        norm_matrix = make_equation_with_quadratic_terms(norm_matrix)[0]
    plan_matr_for_calc_Y = plan_matr
    N = len(plan_matr)
    Y_matrix = []
    mean_Y = []
    indexes = []
    flag_of_dispersion = False
    while flag_of_dispersion is False:
        Y_matrix = np.array(
            [0.4 + 1.8 * plan_matr_for_calc_Y[:, 1] + 6.2 * plan_matr_for_calc_Y[:, 2] + 9.1 * plan_matr_for_calc_Y[:, 3]
+
             3.7 * plan_matr_for_calc_Y[:, 1] * plan_matr_for_calc_Y[:, 1] + 0.1 * plan_matr_for_calc_Y[:, 2] *
```

```
            plan_matr_for_calc_Y[:, 2] + 9.0 * plan_matr_for_calc_Y[:, 3] * plan_matr_for_calc_Y[:, 3] + 1.9 *
            plan_matr_for_calc_Y[:, 1] * plan_matr_for_calc_Y[:, 2] + 0.8 * plan_matr_for_calc_Y[:, 1] *
            plan_matr_for_calc_Y[:, 3] + 9.9 * plan_matr_for_calc_Y[:, 2] * plan_matr_for_calc_Y[:, 3] + 4.7 *
            plan_matr_for_calc_Y[:, 1] * plan_matr_for_calc_Y[:, 2] * plan_matr_for_calc_Y[:, 3] +
            random.randint(0, 100) - 50 for i in range(m)]).T
        mean_Y = np.mean(Y_matrix, axis=1)
        if cohran(Y_matrix, N):
            flag_of_dispersion = True
            b_natural = np.linalg.lstsq(plan_matr, mean_Y, rcond=None)[0]
            b_norm = np.linalg.lstsq(norm_matrix, mean_Y, rcond=None)[0]
            check1 = np.sum(b_natural * plan_matr, axis=1)
            indexes = students_t_test(norm_matrix, Y_matrix, N)
            check2 = np.sum(b_natural[indexes] * np.reshape(plan_matr[:, indexes], (N, np.size(indexes))), axis=1)
            print("Фактори: \n", plan_matr)
            print("Нормована матриця факторів: \n", norm_matrix)
            print("Функції відгуку: \n", Y_matrix)
            print("Середні значення У: ", mean_Y)
            print("Натуралізовані коефіцієнти: ", b_natural)
            print("Перевірка 1: ", check1)
            print("Індекси коефіцієнтів, які задовольняють критерію Стьюдента: ", np.array(indexes)[0])
            print("Критерій Стьюдента: ", check2)
        else:
            m += 1
            print("Дисперсія неоднорідна!")
    if fisher_criterion(Y_matrix, np.size(indexes), N):
        flag_of_model = True
        print("Рівняння регресії є адекватним оригіналу.")
    else:
        count += 1
        print("Рівняння регресії не є адекватним оригіналу.")
```