Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №5

«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів»

Виконав:
студент групи ІВ-81
Зікратий Д. О.
Залікова книжка № 8114
Перевірив:
Регіда П. Г.

Київ – 2020

Варіант завдання

№варіанта	x_1		x_2		x_3	
	min	max	min	max	min	max
113	-6	2	0	2	-6	8

Код програми

```
import numpy as np
from scipy.stats import f,t
from random import randrange

#const
x1_min= -6
x1_max= 2
x2_min= 0
x2_max= 2
x3_min= -6
x3_max= 8
x_average_max = (x1_max + x2_max + x3_max)/3
x_average_min = (x1_min + x2_min + x3_min)/3
y_max = 200 + x_average_max
y_min = 200 + x_average_min


def main(x1_min, x1_max, x2_min, x2_max, x3_min, x3_max,matrix_y,m=3,N=4,p=0.95):
    matrix_x_natural = np.array([[x1_min, x2_min, x3_min],
                                 [x1_min, x2_max, x3_max],
                                 [x1_max, x2_min, x3_max],
                                 [x1_max, x2_max, x3_min]])
    matrix_x = [[1, -1, -1, -1],
                [1, -1, 1, 1],
                [1, 1, -1, 1],
                [1, 1, 1, -1]]

    #Slice only for 4 function y
    matrix_y = matrix_y[:4,:]
    print("Matrix of Y")
    print(matrix_y)
    print("Matrix of normalized X")
    print(matrix_x_natural)
    y_average_list = [sum(y)/len(y) for y in matrix_y ]
    mx_list=[_/N for _ in np.sum(matrix_x_natural, axis=0)]
    mx1,mx2,mx3=mx_list
    my=sum(y_average_list)/len(y_average_list)

    a1 = sum([matrix_x_natural[_][0] * y_average_list[_] for _ in range(N)]) / N
    a2 = sum([matrix_x_natural[_][1] * y_average_list[_] for _ in range(N)]) / N
    a3 = sum([matrix_x_natural[_][2] * y_average_list[_] for _ in range(N)]) / N

    a11 = sum([matrix_x_natural[_][0] ** 2 for _ in range(N)]) / N
    a22 = sum([matrix_x_natural[_][1] ** 2 for _ in range(N)]) / N
    a33 = sum([matrix_x_natural[_][2] ** 2 for _ in range(N)]) / N

    a12 = sum([matrix_x_natural[_][0] * matrix_x_natural[_][1] for _ in range(N)]) /
N
    a13 = sum([matrix_x_natural[_][0] * matrix_x_natural[_][2] for _ in range(N)]) /
N
    a32 = sum([matrix_x_natural[_][2] * matrix_x_natural[_][1] for _ in range(N)]) /
N
```

```
    a21=a12
    a23=a32
    a31=a13
    det_denominator =
np.linalg.det([[1,mx1,mx2,mx3],[mx1,a11,a12,a13],[mx2,a12,a22,a32],[mx3,a13,a23,a33]]
)
    b0 =
np.linalg.det([[my,mx1,mx2,mx3],[a1,a11,a12,a13],[a2,a12,a22,a32],[a3,a13,a23,a33]])
/ det_denominator
    b1 =
np.linalg.det([[1,my,mx2,mx3],[mx1,a1,a12,a13],[mx2,a2,a22,a32],[mx3,a3,a23,a33]]) /
det_denominator
    b2 =
np.linalg.det([[1,mx1,my,mx3],[mx1,a11,a1,a13],[mx2,a12,a2,a32],[mx3,a13,a3,a33]]) /
det_denominator
    b3 =
np.linalg.det([[1,mx1,mx2,my],[mx1,a11,a12,a1],[mx2,a12,a22,a2],[mx3,a13,a23,a3]]) /
det_denominator

    print(f"Regression y = {round(b0,2)} + {round(b1,2)}*X1 + {round(b2,2)}*X2 +
{round(b3,2)}*X3")
    #checking
    y = [b0 + matrix_x_natural[_][0]*b1 + matrix_x_natural[_][1]*b2 +
matrix_x_natural[_][2]*b3 for _ in range(N)]


    dispersion1 = sum([(_ - y_average_list[0])**2 for _ in matrix_y[0]]) / m
    dispersion2 = sum([(_ - y_average_list[1])**2 for _ in matrix_y[1]]) / m
    dispersion3 = sum([(_ - y_average_list[2])**2 for _ in matrix_y[2]]) / m
    dispersion4 = sum([(_ - y_average_list[3])**2 for _ in matrix_y[3]]) / m
    dispersion_list = [dispersion1,dispersion2,dispersion3,dispersion4]

    Gp = max(dispersion_list)/sum(dispersion_list)
    f1 = m-1,
    f2 = N

    Gt=(1 / (1 + (f2 - 1) / f.ppf(1 - (1 - p) / f2, f1, (f2 - 1) * f1)))[0]

    if Gp < Gt:
        print(f"Homogeneous dispersion with {p} probability:\t{round(Gp,3)} < {Gt}")
    else:
        print(f"Inhomogeneous dispersion with {p} probability:\t{round(Gp,3)} >
{Gt}")
        return False


    s_b = sum(dispersion_list) / N
    s2_b_s = s_b / (N * m)
    s_b_s = s2_b_s**(0.5)

    beta0 = sum([matrix_x[_][0] * y_average_list[_] for _ in range(len(matrix_x))]) /
N
    beta1 = sum([matrix_x[_][1] * y_average_list[_] for _ in range(len(matrix_x))]) /
N
    beta2 = sum([matrix_x[_][2] * y_average_list[_] for _ in range(len(matrix_x))]) /
N
    beta3 = sum([matrix_x[_][3] * y_average_list[_] for _ in range(len(matrix_x))]) /
N

    t0 = abs(beta0) / s_b_s
    t1 = abs(beta1) / s_b_s
```

```
    t2 = abs(beta2) / s_b_s
    t3 = abs(beta3) / s_b_s
    f3 = f1 * f2
    t_table = t.ppf((1 + p) / 2, f3)[0]
    b00 = b0 if t0 > t_table else 0
    b11 = b1 if t1 > t_table else 0
    b22 = b2 if t2 > t_table else 0
    b33 = b3 if t3 > t_table else 0
    b_list = np.array([b00, b11, b22, b33])
    print(f"Regression y = {round(b00, 2)} + {round(b11, 2)}*X1 + {round(b22, 2)}*X2
+ {round(b33, 2)}*X3")


    ch11 = b00 + b11 * matrix_x_natural[0][0] + b22 * matrix_x_natural[0][1] + b33 *
matrix_x_natural[0][2]
    ch22 = b00 + b11 * matrix_x_natural[1][0] + b22 * matrix_x_natural[1][1] + b33 *
matrix_x_natural[1][2]
    ch33 = b00 + b11 * matrix_x_natural[2][0] + b22 * matrix_x_natural[2][1] + b33 *
matrix_x_natural[2][2]
    ch44 = b00 + b11 * matrix_x_natural[3][0] + b22 * matrix_x_natural[3][1] + b33 *
matrix_x_natural[3][2]
    ch_list = [ch11, ch22, ch33, ch44]


    d = len(b_list[np.array(b_list) != 0])
    f4 = N - d

    s2_ad = m / f4 * sum([(ch_list[_] - y_average_list[_]) ** 2 for _ in
range(len(y_average_list))])
    Fp = s2_ad / s2_b_s
    Ft = f.ppf(p, f4, f3)[0]

    if Fp > Ft:
        print(f"Equation is not adequate to the original with probability -
{p}:\t{round(Fp,3)} > {round(Ft,3)}")
        return False
    else:
        print(f"Equation is adequate to the original with probability -
{p}:\t{round(Fp,3)} < {round(Ft,3)}")
    return True

def after_main(x1_min, x1_max, x2_min, x2_max, x3_min,
x3_max,matrix_y,m=3,N=8,p=0.95):
    #Slicing matrix for only 8 functions of y
    matrix_y = matrix_y[:8, :]
    result_main = main(x1_min, x1_max, x2_min, x2_max, x3_min, x3_max,matrix_y,m)
    print(result_main)
    if not result_main:
        y_average_list = [sum(y) / len(y) for y in matrix_y]
        matrix_x_normal = np.array([[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]])

        x1x2_normal = np.array([[x1 * x2 for x1, x2 in zip(matrix_x_normal[:, 1],
matrix_x_normal[:, 2])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x1x2_normal, axis=1)
        x1x3_normal = np.array([[x1 * x3 for x1, x3 in zip(matrix_x_normal[:, 1],
```

```
matrix_x_normal[:, 3])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x1x3_normal, axis=1)
        x2x3_normal = np.array([[x2 * x3 for x2, x3 in zip(matrix_x_normal[:, 2],
matrix_x_normal[:, 3])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x2x3_normal, axis=1)
        x1x2x3_normal = np.array([[x1 * x2 * x3 for x1, x2, x3 in
zip(matrix_x_normal[:, 1], matrix_x_normal[:, 2],

matrix_x_normal[:, 3])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x1x2x3_normal, axis=1)

        b1_s = []
        for j in range(N):
            s = 0
            for i in range(N):
                s += (matrix_x_normal[i][j] * y_average_list[i]) / N
            b1_s.append(s)

        print(f"Regression with interaction effect:\ny = {round(b1_s[0],3)} +
{round(b1_s[1],3)} * x1 + {round(b1_s[2],3)} * x2 +"
            f" {round(b1_s[3],3)} * x3 + {round(b1_s[4],3)} * x1x2 +
{round(b1_s[5],3)} * x1x3 + {round(b1_s[6],3)} * x2x3 + {round(b1_s[7],3)} * x1x2x3")

        matrix_x_natural = np.array([[x1_min, x2_min, x3_min],
                                     [x1_min, x2_max, x3_max],
                                     [x1_max, x2_min, x3_max],
                                     [x1_max, x2_max, x3_min],
                                     [x1_min, x2_min, x3_max],
                                     [x1_min, x2_max, x3_min],
                                     [x1_max, x2_min, x3_min],
                                     [x1_max, x2_max, x3_max]])
        #Method T used to transpose matrix
        matrix_x_natural = np.insert(matrix_x_natural, 0, 1, axis=1)
        x1x2_natural = np.array([[x1 * x2 for x1, x2 in zip(matrix_x_natural[:, 1],
matrix_x_natural[:, 2])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x1x2_natural, axis=1)
        x1x3_natural = np.array([[x1 * x3 for x1, x3 in zip(matrix_x_natural[:, 1],
matrix_x_natural[:, 3])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x1x3_natural, axis=1)
        x2x3_natural = np.array([[x2 * x3 for x2, x3 in zip(matrix_x_natural[:, 2],
matrix_x_natural[:, 3])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x2x3_natural, axis=1)
        x1x2x3_natural = np.array([[x1 * x2 * x3 for x1, x2, x3 in
zip(matrix_x_natural[:, 1], matrix_x_natural[:, 2],

matrix_x_natural[:, 3])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x1x2x3_natural, axis=1)

        b2_s = np.linalg.solve(matrix_x_natural, y_average_list)

        print(f"Regression with interaction effect:\ny = {round(b2_s[0],3)} +
{round(b2_s[1],3)} * x1 + {round(b2_s[2],3)} * x2 + "
            f"{round(b2_s[3],3)} * x3 + {round(b2_s[4],3)} * x1x2 +
{round(b2_s[5],3)} * x1x3 + {round(b2_s[6],3)} * x2x3 + {round(b2_s[7],3)} * x1x2x3")

        #checking dispersion by Kohren
        dispersion1 = sum([(_ - y_average_list[0]) ** 2 for _ in matrix_y[0]]) / m
        dispersion2 = sum([(_ - y_average_list[1]) ** 2 for _ in matrix_y[1]]) / m
        dispersion3 = sum([(_ - y_average_list[2]) ** 2 for _ in matrix_y[2]]) / m
        dispersion4 = sum([(_ - y_average_list[3]) ** 2 for _ in matrix_y[3]]) / m
        dispersion5 = sum([(_ - y_average_list[4]) ** 2 for _ in matrix_y[4]]) / m
        dispersion6 = sum([(_ - y_average_list[5]) ** 2 for _ in matrix_y[5]]) / m
```

```
        dispersion7 = sum([(_ - y_average_list[6]) ** 2 for _ in matrix_y[6]]) / m
        dispersion8 = sum([(_ - y_average_list[7]) ** 2 for _ in matrix_y[7]]) / m

        #Dispersion list
        dispersion_list = [dispersion1, dispersion2, dispersion3, dispersion4,
dispersion5, dispersion6, dispersion7, dispersion8]

        Gp = max(dispersion_list) / sum(dispersion_list)
        f1, f2 = m - 1, N
        f_value = f.ppf(p / f1, f2, (f1 - 1) * f2)
        Gt = f_value / (f_value + f1 - 1)
        if Gp < Gt:
            print(f"Homogeneous dispersion with {p} probability:\n{round(Gp,3)} <
{Gt}")
        else:
            print(f"Inhomogeneous dispersion with {p} probability:\n{round(Gp,3)} >
{Gt}")
            return False


        s_b = sum(dispersion_list) / N
        s2_b_s = s_b / (N * m)
        s_b_s = pow(s2_b_s, 1 / 2)


        beta0 = sum([matrix_x_normal[_][0] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta1 = sum([matrix_x_normal[_][1] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta2 = sum([matrix_x_normal[_][2] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta3 = sum([matrix_x_normal[_][3] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta4 = sum([matrix_x_normal[_][4] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta5 = sum([matrix_x_normal[_][5] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta6 = sum([matrix_x_normal[_][6] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta7 = sum([matrix_x_normal[_][7] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        #Beta list
        beta_list = [beta0,beta1,beta2,beta3,beta4,beta5,beta6,beta7]

        #t list
        t_list = [abs(beta)/s_b_s for beta in beta_list ]


        f3 = f1 * f2

        #b list
        beta_s = []
        t_table = t.ppf((1 + p) / 2, f3)
        for i in range(len(beta_list)):
            if t_list[i]>t_table:
                beta_s.append(beta_list[i])
            else:
                beta_s.append(0)


        ch0 = beta_s[0] + beta_s[1] * matrix_x_natural[0][1] + beta_s[2] *
matrix_x_natural[0][2] + beta_s[3] * matrix_x_natural[0][3] + \
```

```
                 beta_s[4] * matrix_x_natural[0][4] + beta_s[5] *
matrix_x_natural[0][5] + beta_s[6] * matrix_x_natural[0][6] + \
                 beta_s[7] * matrix_x_natural[0][7]
        ch1 = beta_s[0] + beta_s[1] * matrix_x_natural[1][1] + beta_s[2] *
matrix_x_natural[1][2] + beta_s[3] * matrix_x_natural[1][3] + \
                 beta_s[4] * matrix_x_natural[1][4] + beta_s[5] *
matrix_x_natural[1][5] + beta_s[6] * matrix_x_natural[1][6] + \
                 beta_s[7] * matrix_x_natural[1][7]
        ch2 = beta_s[0] + beta_s[1] * matrix_x_natural[2][1] + beta_s[2] *
matrix_x_natural[2][2] + beta_s[3] * matrix_x_natural[2][3] + \
                 beta_s[4] * matrix_x_natural[2][4] + beta_s[5] *
matrix_x_natural[2][5] + beta_s[6] * matrix_x_natural[2][6] + \
                 beta_s[7] * matrix_x_natural[2][7]
        ch3 = beta_s[0] + beta_s[1] * matrix_x_natural[3][1] + beta_s[2] *
matrix_x_natural[3][2] + beta_s[3] * matrix_x_natural[3][3] + \
                 beta_s[4] * matrix_x_natural[3][4] + beta_s[5] *
matrix_x_natural[3][5] + beta_s[6] * matrix_x_natural[3][6] + \
                 beta_s[7] * matrix_x_natural[3][7]
        ch4 = beta_s[0] + beta_s[1] * matrix_x_natural[4][1] + beta_s[2] *
matrix_x_natural[4][2] + beta_s[3] * matrix_x_natural[4][3] + \
                 beta_s[4] * matrix_x_natural[4][4] + beta_s[5] *
matrix_x_natural[4][5] + beta_s[6] * matrix_x_natural[4][6] + \
                 beta_s[7] * matrix_x_natural[4][7]
        ch5 = beta_s[0] + beta_s[1] * matrix_x_natural[5][1] + beta_s[2] *
matrix_x_natural[5][2] + beta_s[3] * matrix_x_natural[5][3] + \
                 beta_s[4] * matrix_x_natural[5][4] + beta_s[5] *
matrix_x_natural[5][5] + beta_s[6] * matrix_x_natural[5][6] + \
                 beta_s[7] * matrix_x_natural[5][7]
        ch6 = beta_s[0] + beta_s[1] * matrix_x_natural[6][1] + beta_s[2] *
matrix_x_natural[6][2] + beta_s[3] * matrix_x_natural[6][3] + \
                 beta_s[4] * matrix_x_natural[6][4] + beta_s[5] *
matrix_x_natural[6][5] + beta_s[6] * matrix_x_natural[6][6] + \
                 beta_s[7] * matrix_x_natural[6][7]
        ch7 = beta_s[0] + beta_s[1] * matrix_x_natural[7][1] + beta_s[2] *
matrix_x_natural[7][2] + beta_s[3] * matrix_x_natural[7][3] + \
                 beta_s[4] * matrix_x_natural[7][4] + beta_s[5] *
matrix_x_natural[7][5] + beta_s[6] * matrix_x_natural[7][6] + \
                 beta_s[7] * matrix_x_natural[7][7]
        list_ch = [ch0, ch1, ch2, ch3, ch4, ch5, ch6, ch7]
        beta_s = np.array(beta_s)
        d = len(beta_s[np.array(beta_s) != 0])
        f4 = N - d
        s_ad = m / f4 * sum([(list_ch[_] - y_average_list[_]) ** 2 for _ in
range(len(y_average_list))])
        Fp = s_ad / s2_b_s
        Ft = f.ppf(p, f4, f3)

        if Fp > Ft:
            print(f"Equation is not adequate to the original with probability -
{p}:\n{round(Fp,3)} > {round(Ft,3)}")
            new_result = False
            while not new_result:
                print('\n')
                new_result = after_main(x1_min, x1_max, x2_min, x2_max, x3_min,
x3_max,m)
                if not new_result:
                    m += 1
        else:
            print(f"Equation is  adequate to the original with probability -
{p}:\n{round(Fp,3)} < {round(Ft,3)}")
        return True
    else:
```

```
        return True


def after_main_2(x1_min, x1_max, x2_min, x2_max, x3_min, x3_max,m=3 , N=15, p=0.95):
    matrix_y = np.random.randint(y_min, y_max, size=(N, m))

    result_ = after_main(x1_min, x1_max, x2_min, x2_max, x3_min, x3_max, matrix_y)
    if not result_:
        y_average_list = [sum(y) / len(y) for y in matrix_y]

        l = 1.215
        x0 = [(x1_min + x1_max) / 2, (x2_min + x2_max) / 2, (x3_min + x3_max) / 2]
        delta_x = [x1_max - x0[0], x2_max - x0[1], x3_max - x0[2]]

        matrix_x_normal = [[-1, -1, -1],
                           [-1, -1, +1],
                           [-1, +1, -1],
                           [-1, +1, +1],
                           [+1, -1, -1],
                           [+1, -1, +1],
                           [+1, +1, -1],
                           [+1, +1, +1],
                           [-1.215, 0, 0],
                           [+1.215, 0, 0],
                           [0, -1.215, 0],
                           [0, +1.215, 0],
                           [0, 0, -1.215],
                           [0, 0, +1.215],
                           [0, 0, 0]]

        matrix_x_normal = np.insert(matrix_x_normal, 0, 1, axis=1)
        x1x2_normal = np.array([[x1 * x2 for x1, x2 in zip(matrix_x_normal[:, 1],
matrix_x_normal[:, 2])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x1x2_normal, axis=1)
        x1x3_normal = np.array([[x1 * x3 for x1, x3 in zip(matrix_x_normal[:, 1],
matrix_x_normal[:, 3])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x1x3_normal, axis=1)
        x2x3_normal = np.array([[x2 * x3 for x2, x3 in zip(matrix_x_normal[:, 2],
matrix_x_normal[:, 3])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x2x3_normal, axis=1)
        x1x2x3_normal = np.array([[x1 * x2 * x3 for x1, x2, x3 in
zip(matrix_x_normal[:, 1], matrix_x_normal[:, 2],

matrix_x_normal[:, 3])]]).T
        matrix_x_normal = np.append(matrix_x_normal, x1x2x3_normal, axis=1)
        pow_x1_normal = np.array([[round(x1 ** 2, 5) for x1 in matrix_x_normal[:,
1]]]).T
        matrix_x_normal = np.append(matrix_x_normal, pow_x1_normal, axis=1)
        pow_x2_normal = np.array([[round(x2 ** 2, 5) for x2 in matrix_x_normal[:,
2]]]).T
        matrix_x_normal = np.append(matrix_x_normal, pow_x2_normal, axis=1)
        pow_x3_normal = np.array([[round(x3 ** 2, 5) for x3 in matrix_x_normal[:,
3]]]).T
        matrix_x_normal = np.append(matrix_x_normal, pow_x3_normal, axis=1)


        # checking dispersion by Kohren
        dispersion1 = sum([(_ - y_average_list[0]) ** 2 for _ in matrix_y[0]]) / m
        dispersion2 = sum([(_ - y_average_list[1]) ** 2 for _ in matrix_y[1]]) / m
        dispersion3 = sum([(_ - y_average_list[2]) ** 2 for _ in matrix_y[2]]) / m
        dispersion4 = sum([(_ - y_average_list[3]) ** 2 for _ in matrix_y[3]]) / m
        dispersion5 = sum([(_ - y_average_list[4]) ** 2 for _ in matrix_y[4]]) / m
```

```python
        dispersion6 = sum([(_ - y_average_list[5]) ** 2 for _ in matrix_y[5]]) / m
        dispersion7 = sum([(_ - y_average_list[6]) ** 2 for _ in matrix_y[6]]) / m
        dispersion8 = sum([(_ - y_average_list[7]) ** 2 for _ in matrix_y[7]]) / m
        dispersion9 = sum([(_ - y_average_list[8]) ** 2 for _ in matrix_y[8]]) / m
        dispersion10 = sum([(_ - y_average_list[9]) ** 2 for _ in matrix_y[9]]) / m
        dispersion11 = sum([(_ - y_average_list[10]) ** 2 for _ in matrix_y[10]]) / m
        dispersion12 = sum([(_ - y_average_list[11]) ** 2 for _ in matrix_y[11]]) / m
        dispersion13 = sum([(_ - y_average_list[12]) ** 2 for _ in matrix_y[12]]) / m
        dispersion14 = sum([(_ - y_average_list[13]) ** 2 for _ in matrix_y[13]]) / m
        dispersion15 = sum([(_ - y_average_list[14]) ** 2 for _ in matrix_y[14]]) / m

        # Dispersion list
        dispersion_list = [dispersion1, dispersion2, dispersion3, dispersion4,
dispersion5, dispersion6, dispersion7,
                           dispersion8,dispersion9, dispersion10, dispersion11,
dispersion12, dispersion13, dispersion14, dispersion15]

        Gp = max(dispersion_list) / sum(dispersion_list)
        f1, f2 = m - 1, N
        f_value = f.ppf(p / f1, f2, (f1 - 1) * f2)
        Gt = f_value / (f_value + f1 - 1)
        if Gp < Gt:
            print(f"Homogeneous dispersion with {p} probability:\n{round(Gp, 3)} <
{Gt}")
        else:
            print(f"Inhomogeneous dispersion with {p} probability:\n{round(Gp, 3)} >
{Gt}")
            return False

        s_b = sum(dispersion_list) / N
        s2_b_s = s_b / (N * m)
        s_b_s = pow(s2_b_s, 1 / 2)


        matrix_x_natural = [[x1_min, x2_min, x3_min],
                            [x1_min, x2_min, x3_max],
                            [x1_min, x2_max, x3_min],
                            [x1_min, x2_max, x3_max],
                            [x1_max, x2_min, x3_min],
                            [x1_max, x2_min, x3_max],
                            [x1_max, x2_max, x3_min],
                            [x1_max, x2_max, x3_max],
                            [round(-l * delta_x[0] + x0[0], 5), x0[1], x0[2]],
                            [round(l * delta_x[0] + x0[0], 5), x0[1], x0[2]],
                            [x0[0], round(-l * delta_x[1] + x0[1], 5), x0[2]],
                            [x0[0], round(l * delta_x[1] + x0[1], 5), x0[2]],
                            [x0[0], x0[1], round(-l * delta_x[2] + x0[2], 5)],
                            [x0[0], x0[1], round(l * delta_x[2] + x0[2], 5)],
                            [x0[0], x0[1], x0[2]]]



        matrix_x_natural = np.insert(matrix_x_natural, 0, 1, axis=1)
        x1x2_natural = np.array([[round(x1 * x2, 5) for x1, x2 in
zip(matrix_x_natural[:, 1], matrix_x_natural[:, 2])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x1x2_natural, axis=1)
        x1x3_natural = np.array([[x1 * x3 for x1, x3 in zip(matrix_x_natural[:, 1],
matrix_x_natural[:, 3])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x1x3_natural, axis=1)
        x2x3_natural = np.array([[x2 * x3 for x2, x3 in zip(matrix_x_natural[:, 2],
matrix_x_natural[:, 3])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x2x3_natural, axis=1)
```

```
        x1x2x3_natural = np.array([[x1 * x2 * x3 for x1, x2, x3 in
zip(matrix_x_natural[:, 1], matrix_x_natural[:, 2],

matrix_x_natural[:, 3])]]).T
        matrix_x_natural = np.append(matrix_x_natural, x1x2x3_natural, axis=1)
        pow_x1_natural = np.array([[round(x1 ** 2, 5) for x1 in matrix_x_natural[:,
1]]]).T
        matrix_x_natural = np.append(matrix_x_natural, pow_x1_natural, axis=1)
        pow_x2_natural = np.array([[round(x2 ** 2, 5) for x2 in matrix_x_natural[:,
2]]]).T
        matrix_x_natural = np.append(matrix_x_natural, pow_x2_natural, axis=1)
        pow_x3_natural = np.array([[round(x3 ** 2, 5) for x3 in matrix_x_natural[:,
3]]]).T
        matrix_x_natural = np.append(matrix_x_natural, pow_x3_natural, axis=1)

        def value_supp_func(*array):
            return np.average(reduce(lambda a, b: a * b, array))

        def x(i):
            all_raw = list(map(lambda x: [1] + x,
                               [row + [row[0] * row[1],
                                row[0] * row[2],
                                row[1] * row[2],
                                row[0] * row[1] * row[2]] +
                                list(map(lambda x: round(x ** 2, 5), row)) for row in
matrix_x_natural]))
            array = [row[i] for row in all_raw]
            return np.array(array)


        coeff = [[value_supp_func(x(col) * x(row)) for col in range(11)] for row in
range(11)]
        value = [value_supp_func(y_average_list, x(i)) for i in range(11)]
        b3_s= np.linalg.solve(coeff, value)

        print(f"Regression based on square terms:\ny = {round(b3_s[0], 3)} +
{round(b3_s[1], 3)} * x1 + {round(b3_s[2], 3)} * x2 + "
            f"{round(b3_s[3], 3)} * x3 + {round(b3_s[4], 3)} * x1x2 + {round(b3_s[5],
3)} * x1x3 + {round(b3_s[6], 3)} * x2x3 + {round(b3_s[7], 3)} * x1x2x3 + "
            f"{round(b3_s[8], 3)} * x1^2 + {round(b3_s[9], 3)} * x2^2 +
{round(b3_s[10], 3)} * x3^2")


        ch_list = []
        for _ in range(len(matrix_x_natural)):
            ch_list.append(b3_s[0] + b3_s[1] * matrix_x_natural[_][1] + \
                           b3_s[2] * matrix_x_natural[_][2] + \
                           b3_s[3] * matrix_x_natural[_][3] + \
                           b3_s[4] * matrix_x_natural[_][4] + \
                           b3_s[5] * matrix_x_natural[_][5] + \
                           b3_s[6] * matrix_x_natural[_][6] + \
                           b3_s[7] * matrix_x_natural[_][7] + \
                           b3_s[8] * matrix_x_natural[_][8] + \
                           b3_s[9] * matrix_x_natural[_][9] + \
                           b3_s[10] * matrix_x_natural[_][10])


        beta0 = sum([matrix_x_normal[_][0] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta1 = sum([matrix_x_normal[_][1] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta2 = sum([matrix_x_normal[_][2] * y_average_list[_] for _ in
```

```
range(len(matrix_x_normal))]) / N
        beta3 = sum([matrix_x_normal[_][3] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta4 = sum([matrix_x_normal[_][4] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta5 = sum([matrix_x_normal[_][5] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta6 = sum([matrix_x_normal[_][6] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta7 = sum([matrix_x_normal[_][7] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta8 = sum([matrix_x_normal[_][8] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta9 = sum([matrix_x_normal[_][9] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N
        beta10 = sum([matrix_x_normal[_][10] * y_average_list[_] for _ in
range(len(matrix_x_normal))]) / N

        beta_list = [beta0, beta1, beta2, beta3, beta4, beta5, beta6, beta7]

        # t list
        t_list = [abs(beta) / s_b_s for beta in beta_list]

        f3 = f1 * f2

        t_table = t.ppf((1 + p) / 2, f3)

        for i in range(len(beta_list)):
            if t_list[i] > t_table:
                beta_s.append(beta_list[i])
            else:
                beta_s.append(0)

        ch2_list = []
        for _ in range(len(matrix_x_natural)):
            ch2_list.append(
                beta_list[0] + beta_list[1] * matrix_x_natural[_][1] + \
                beta_list[2] * matrix_x_natural[_][2] + \
                beta_list[3] * matrix_x_natural[_][3] + \
                beta_list[4] * matrix_x_natural[_][4] + \
                beta_list[5] * matrix_x_natural[_][5] + \
                beta_list[6] * matrix_x_natural[_][6] + \
                beta_list[7] * matrix_x_natural[_][7] + \
                beta_list[8] * matrix_x_natural[_][8] + \
                beta_list[9] * matrix_x_natural[_][9] + \
                beta_list[10] * matrix_x_natural[_][10])


        d = len(beta_list[np.array(beta_list) != 0])
        f4 = N - d


        s_ad = m / f4 * sum([(ch2_list[_] - y_average_list[_]) ** 2 for _ in
range(len(y_average_list))])
        Fp = s_ad / s2_b_s
        Ft = f.ppf(p, f4, f3)

        if Fp > Ft:
            print(f"Equation is not adequate to the original with probability -
{p}:\n{round(Fp, 3)} > {round(Ft, 3)}")
            new_result = False
            while not new_result:
```

```
                print('\n')
                new_result = after_main_2(x1_min, x1_max, x2_min, x2_max, x3_min,
x3_max, m)
                if not new_result:
                    m += 1
        else:
            print(f"Equation is  adequate to the original with probability -
{p}:\n{round(Fp, 3)} < {round(Ft, 3)}")
        return True
    else:
        return True



if __name__=="__main__":
    m=3
    result = False
    while not result:
        result = after_main_2(x1_min, x1_max, x2_min, x2_max, x3_min, x3_max, m)
        if not result:
            m += 1
{p}:\n{round(Fp,3)} < {round(Ft,3)}")
        return True
    else:
        return True

if __name__=="__main__":
    m=3
    result = False
    while not result:
        result = after_main(x1_min, x1_max, x2_min, x2_max, x3_min, x3_max, m)
        if not result:
            m += 1
```

Результати

```
Matrix of Y
[[199 202 196]
 [198 201 196]
 [202 199 202]
 [203 199 198]]
Matrix of normalized X
[[-6  0 -6]
 [-6  2  8]
 [ 2  0  8]
 [ 2  2 -6]]
Regression y = 200.45 + 0.23*X1 + -0.42*X2 + 0.01*X3
Homogeneous dispersion with 0.95 probability:   0.355 < 0.9634146341463417
Regression y = 200.45 + 0*X1 + 0*X2 + 0*X3
Equation is not adequate to the original with probability - 0.95:   20.074 > 19.164
False
Regression with interaction effect:
y = 199.875 + 0.542 * x1 + -0.292 * x2 + -0.042 * x3 + -0.125 * x1x2 + 0.125 * x1x3 + -0.375 * x2x3 + 0.292 * x1x2x3
Regression with interaction effect:
y = 200.464 + 0.173 * x1 + -0.321 * x2 + 0.036 * x3 + -0.042 * x1x2 + -0.006 * x1x3 + -0.033 * x2x3 + 0.01 * x1x2x3
Homogeneous dispersion with 0.95 probability:
0.218 < 0.4885654502272499
Equation is not adequate to the original with probability - 0.95:
10.544 > 2.657
```

```
Matrix of Y
[[198 196 199]
 [202 200 196]
 [200 198 199]
 [200 196 197]]
Matrix of normalized X
[[-6  0 -6]
 [-6  2  8]
 [ 2  0  8]
 [ 2  2 -6]]
Regression y = 198.18 + -0.02*X1 + 0.08*X2 + 0.11*X3
Homogeneous dispersion with 0.95 probability:   0.549 < 0.9634146341463417
Regression y = 198.18 + 0*X1 + 0*X2 + 0*X3
Equation is adequate to the original with probability - 0.95:   10.678 < 19.164
True
```