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

Лабораторна робота №6

«Проведення трьох факторного експерименту при використанні рівняння регресії з квадратичними членами»

Виконав:
студент групи ІО-91
Андрейцов Я. Є.
Залікова книжка № ІО-9101
Варіант 1

ПЕРЕВІРИВ:
Егіда П. Г.

Київ-2020

Варіант 1

Таблиця варіантів

№ варіанту	x_1		x_2		x_3		$f(x_1, x_2, x_3)$
	min	max	min	max	min	max	
101	-10	50	20	60	50	55	1,5+1,5*x1+6,8*x2+3,2*x3+2,7*x1*x1+0,1*x2*x2+1,2*x3*x3+6,2*x1*x2+0,7*x1*x3+2,4*x2*x3+6,1*x1*x2*x3

Код програми

```
from math import fabs, sqrt
m = 3
p = 0.95
N = 15
x1_min = -10
x1_max = 50
x2_min = 20
x2_max = 60
x3_min = 50
x3_max = 55
x01 = (x1_max + x1_min) / 2
x02 = (x2_max + x2_min) / 2
x03 = (x3_max + x3_min) / 2
delta_x1 = x1_max - x01
delta_x2 = x2_max - x02
delta_x3 = x3_max - x03


class Perevirku:
    def get_cohren_value(size_of_selections, qty_of_selections,
significance):
        from  pydecimal import Decimal
        from scipy.stats import f
        size_of_selections += 1
        partResult1 = significance / (size_of_selections - 1)
        params = [partResult1, qty_of_selections, (size_of_selections - 1 -
1) * qty_of_selections]
        fisher = f.isf(*params)
        result = fisher / (fisher + (size_of_selections - 1 - 1))
        return Decimal(result).quantize(Decimal('.0001')).__float__()

    def get_student_value(f3, significance):
        from _pydecimal import Decimal
        from scipy.stats import t
        return Decimal(abs(t.ppf(significance / 2,
f3))).quantize(Decimal('.0001')).__float__()

    def get_fisher_value(f3, f4, significance):
        from _pydecimal import Decimal
        from scipy.stats import f
        return Decimal(abs(f.isf(significance, f4,
f3))).quantize(Decimal('.0001')).__float__()


def generate_matrix():
    def f(X1, X2, X3):
        from random import randrange
        y = 1.5 + 1.5 * X1 + 6.8 * X2 + 3.2 * X3 + 2.7 * X1 * X1 + 0.1 * X2 *
X2 + 1.2 * X3 * X3 + 6.2 * X1 * X2 + \
            0.7 * X1 * X3 + 2.4 * X2 * X3 + 6.1 * X1 * X2 * X3 + randrange(0,
10) - 5
        return y

    matrix_with_y = [[f(matrix_x[j][0], matrix_x[j][1], matrix_x[j][2]) for i
in range(m)] for j in range(N)]
    return matrix_with_y
```

```
def x(l1, l2, l3):
    x_1 = l1 * delta_x1 + x01
    x_2 = l2 * delta_x2 + x02
    x_3 = l3 * delta_x3 + x03
    return [x_1, x_2, x_3]


def find_average(lst, orientation):
    average = []
    if orientation == 1:
        for rows in range(len(lst)):
            average.append(sum(lst[rows]) / len(lst[rows]))
    else:
        for column in range(len(lst[0])):
            number_lst = []
            for rows in range(len(lst)):
                number_lst.append(lst[rows][column])
            average.append(sum(number_lst) / len(number_lst))
    return average


def a(first, second):
    need_a = 0
    for j in range(N):
        need_a += matrix_x[j][first - 1] * matrix_x[j][second - 1] / N
    return need_a


def find_known(number):
    need_a = 0
    for j in range(N):
        need_a += average_y[j] * matrix_x[j][number - 1] / 15
    return need_a


def solve(lst_1, lst_2):
    from numpy.linalg import solve
    solver = solve(lst_1, lst_2)
    return solver


def check_result(b_lst, k):
    y_i = b_lst[0] + b_lst[1] * matrix[k][0] + b_lst[2] * matrix[k][1] +
b_lst[3] * matrix[k][2] + \
          b_lst[4] * matrix[k][3] + b_lst[5] * matrix[k][4] + b_lst[6] *
matrix[k][5] + b_lst[7] * matrix[k][6] + \
          b_lst[8] * matrix[k][7] + b_lst[9] * matrix[k][8] + b_lst[10] *
matrix[k][9]
    return y_i


def student_test(b_lst, number_x=10):
    dispersion_b = sqrt(dispersion_b2)
    for column in range(number_x + 1):
        t_practice = 0
        t_theoretical = Perevirku.get_student_value(f3, q)
        for row in range(N):
            if column == 0:
                t_practice += average_y[row] / N
            else:
                t_practice += average_y[row] * matrix_pfe[row][column - 1]
        if fabs(t_practice / dispersion_b) < t_theoretical:
```

```
            b_lst[column] = 0
    return b_lst


def fisher_test():
    dispersion_ad = 0
    f4 = N - d
    for row in range(len(average_y)):
        dispersion_ad += (m * (average_y[row] - check_result(student_lst,
row))) / (N - d)
    F_practice = dispersion_ad / dispersion_b2
    F_theoretical = Perevirku.get_fisher_value(f3, f4, q)
    return F_practice < F_theoretical


matrix_pfe = [
    [-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, -1, +1, +1, +1],
    [+1, +1, -1, +1, -1, -1, -1, +1, +1, +1],
    [+1, +1, +1, +1, +1, +1, +1, +1, +1, +1],
    [-1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
    [+1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],
    [0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],
    [0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],
    [0, 0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929],
    [0, 0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]

matrix_x = [[] for x in range(N)]
for i in range(len(matrix_x)):
    if i < 8:
        x_1 = x1_min if matrix_pfe[i][0] == -1 else x1_max
        x_2 = x2_min if matrix_pfe[i][1] == -1 else x2_max
        x_3 = x3_min if matrix_pfe[i][2] == -1 else x3_max
    else:
        x_lst = x(matrix_pfe[i][0], matrix_pfe[i][1], matrix_pfe[i][2])
        x_1, x_2, x_3 = x_lst
    matrix_x[i] = [x_1, x_2, x_3, x_1 * x_2, x_1 * x_3, x_2 * x_3, x_1 * x_2
* x_3, x_1 ** 2, x_2 ** 2, x_3 ** 2]


adekvat = False
odnorid = False
while not adekvat:
    matrix_y = generate_matrix()
    average_x = find_average(matrix_x, 0)
    average_y = find_average(matrix_y, 1)
    matrix = [(matrix_x[i] + matrix_y[i]) for i in range(N)]
    mx_i = average_x
    my = sum(average_y) / 15

    unknown = [
        [1, mx_i[0], mx_i[1], mx_i[2], mx_i[3], mx_i[4], mx_i[5], mx_i[6],
mx_i[7], mx_i[8], mx_i[9]],
        [mx_i[0], a(1, 1), a(1, 2), a(1, 3), a(1, 4), a(1, 5), a(1, 6), a(1,
7), a(1, 8), a(1, 9), a(1, 10)],
        [mx_i[1], a(2, 1), a(2, 2), a(2, 3), a(2, 4), a(2, 5), a(2, 6), a(2,
7), a(2, 8), a(2, 9), a(2, 10)],
        [mx_i[2], a(3, 1), a(3, 2), a(3, 3), a(3, 4), a(3, 5), a(3, 6), a(3,
```

```
7), a(3, 8), a(3, 9), a(3, 10)],
        [mx_i[3], a(4, 1), a(4, 2), a(4, 3), a(4, 4), a(4, 5), a(4, 6), a(4,
7), a(4, 8), a(4, 9), a(4, 10)],
        [mx_i[4], a(5, 1), a(5, 2), a(5, 3), a(5, 4), a(5, 5), a(5, 6), a(5,
7), a(5, 8), a(5, 9), a(5, 10)],
        [mx_i[5], a(6, 1), a(6, 2), a(6, 3), a(6, 4), a(6, 5), a(6, 6), a(6,
7), a(6, 8), a(6, 9), a(6, 10)],
        [mx_i[6], a(7, 1), a(7, 2), a(7, 3), a(7, 4), a(7, 5), a(7, 6), a(7,
7), a(7, 8), a(7, 9), a(7, 10)],
        [mx_i[7], a(8, 1), a(8, 2), a(8, 3), a(8, 4), a(8, 5), a(8, 6), a(8,
7), a(8, 8), a(8, 9), a(8, 10)],
        [mx_i[8], a(9, 1), a(9, 2), a(9, 3), a(9, 4), a(9, 5), a(9, 6), a(9,
7), a(9, 8), a(9, 9), a(9, 10)],
        [mx_i[9], a(10, 1), a(10, 2), a(10, 3), a(10, 4), a(10, 5), a(10, 6),
a(10, 7), a(10, 8), a(10, 9), a(10, 10)]
    ]
    known = [my, find_known(1), find_known(2), find_known(3), find_known(4),
find_known(5), find_known(6),
             find_known(7),
             find_known(8), find_known(9), find_known(10)]

    beta = solve(unknown, known)
    print("Отримане рівняння регресії")
    print("{:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 +
{:.3f} * X1X3 + {:.3f} * X2X3"
          "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} *
X33^2 = ŷ\n\tПеревірка"
          .format(beta[0], beta[1], beta[2], beta[3], beta[4], beta[5],
beta[6], beta[7], beta[8], beta[9], beta[10]))
    for i in range(N):
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(beta, i),
average_y[i]))

    while not odnorid:
        print("Матриця планування експеременту:")
        print("      X1           X2           X3          X1X2        X1X3
X2X3         X1X2X3       X1X1"
              "         X2X2         X3X3          Yi ->")
        for row in range(N):
            print( end=' ')
            for column in range(len(matrix[0])):
                print("{:^12.3f}".format(matrix[row][column]), end=' ')
            print("")

        dispersion_y = [0.0 for x in range(N)]
        for i in range(N):
            dispersion_i = 0
            for j in range(m):
                dispersion_i += (matrix_y[i][j] - average_y[i]) ** 2
            dispersion_y.append(dispersion_i / (m - 1))
        f1 = m - 1
        f2 = N
        f3 = f1 * f2
        q = 1 - p
        Gp = max(dispersion_y) / sum(dispersion_y)
        print("Критерій Кохрена:")
        Gt = Perevirku.get_cohren_value(f2, f1, q)
        if Gt > Gp:
            print("Дисперсія однорідна при рівні значимості
{:.2f}.".format(q))
            odnorid = True
        else:
            print("Дисперсія не однорідна при рівні значимості {:.2f}!
Збільшуємо m.".format(q))
```

```
            m += 1

    dispersion_b2 = sum(dispersion_y) / (N * N * m)
    student_lst = list(student_test(beta))
    print("Отримане рівняння регресії з урахуванням критерія Стьюдента")
    print("{:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 +
{:.3f} * X1X3 + {:.3f} * X2X3"
          "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} *
X33^2 = ŷ\n\tПеревірка"
          .format(student_lst[0], student_lst[1], student_lst[2],
student_lst[3], student_lst[4], student_lst[5],
                  student_lst[6], student_lst[7], student_lst[8],
student_lst[9], student_lst[10]))
    for i in range(N):
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1),
check_result(student_lst, i), average_y[i]))

    print("Критерій Фішера")
    d = 11 - student_lst.count(0)
    if fisher_test():
        print("Рівняння регресії адекватне  оригіналу")
        adekvat = True
    else:
        print("Рівняння регресії неадекватне  оригіналу\n\t Проводимо
експеремент повторно")
```

Результати виконання роботи

```
"E:\Anaconda2 24.12.2020\envs\Lab3\python.exe" "E:/IV семестр/МОПЕ/Лаби/Лаба 6/Лаба 6.py"
Отримане рівняння регресії
470.719 + 2.714 * X1 + 7.410 * X2 + -15.428 * X3 + 6.166 * X1X2 + 0.676 * X1X3 + 2.386 * X2X3+ 6.101 * X1X2X3 + 2.700 * X11^2 + 0.101 * X22^2 + 1.385 * X33^2 = ŷ
    Перевірка
ŷ1 = -56598.350 ≈ -56598.833
ŷ2 = -61844.287 ≈ -61843.500
ŷ3 = -175684.038 ≈ -175684.167
ŷ4 = -192653.975 ≈ -192652.833
ŷ5 = 325511.635 ≈ 325511.167
ŷ6 = 357072.364 ≈ 357073.167
ŷ7 = 953301.280 ≈ 953301.167
ŷ8 = 1046345.676 ≈ 1046346.833
ŷ9 = -406073.063 ≈ -406073.495
ŷ10 = 964524.457 ≈ 964523.989
ŷ11 = 41298.010 ≈ 41297.969
ŷ12 = 502847.989 ≈ 502847.129
ŷ13 = 249835.888 ≈ 249836.907
ŷ14 = 294119.239 ≈ 294117.320
ŷ15 = 271951.660 ≈ 271951.667
Матриця планування експеременту:
```

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3	Yi ->		
-10.000	20.000	50.000	-200.000	-500.000	1000.000	-10000.000	100.000	400.000	2500.000	-56596.500	-56599.500	-56600.500
-10.000	20.000	55.000	-200.000	-550.000	1100.000	-11000.000	100.000	400.000	3025.000	-61844.500	-61842.500	-61843.500
-10.000	60.000	50.000	-600.000	-500.000	3000.000	-30000.000	100.000	3600.000	2500.000	-175686.500	-175681.500	-175684.500
-10.000	60.000	55.000	-600.000	-550.000	3300.000	-33000.000	100.000	3600.000	3025.000	-192650.500	-192656.500	-192651.500
50.000	20.000	50.000	1000.000	2500.000	1000.000	50000.000	2500.000	400.000	2500.000	325507.500	325516.500	325509.500
50.000	20.000	55.000	1000.000	2750.000	1100.000	55000.000	2500.000	400.000	3025.000	357076.500	357069.500	357073.500
50.000	60.000	50.000	3000.000	2500.000	3000.000	150000.000	2500.000	3600.000	2500.000	953301.500	953299.500	953302.500
50.000	60.000	55.000	3000.000	2750.000	3300.000	165000.000	2500.000	3600.000	3025.000	1046346.500	1046345.500	1046348.500
-31.900	40.000	52.500	-1276.000	-1674.750	2100.000	-66990.000	1017.610	1600.000	2756.250	-406069.828	-406071.828	-406078.828
71.900	40.000	52.500	2876.000	3774.750	2100.000	150990.000	5169.610	1600.000	2756.250	964525.322	964523.322	964523.322
20.000	5.400	52.500	108.000	1050.000	283.500	5670.000	400.000	29.160	2756.250	41300.636	41296.636	41296.636
20.000	74.600	52.500	1492.000	1050.000	3916.500	78330.000	400.000	5565.160	2756.250	502847.796	502845.796	502847.796
20.000	40.000	48.175	800.000	963.500	1927.000	38540.000	400.000	1600.000	2320.831	249832.907	249837.907	249839.907
20.000	40.000	56.825	800.000	1136.500	2273.000	45460.000	400.000	1600.000	3229.081	294113.987	294118.987	294118.987
20.000	40.000	52.500	800.000	1050.000	2100.000	42000.000	400.000	1600.000	2756.250	271951.000	271953.000	271951.000

```
Критерій Кохрена:
Дисперсія однорідна при рівні значимості 0.05.
Отримане рівняння регресії з урахуванням критерія Стьюдента
470.719 + 2.714 * X1 + 7.410 * X2 + -15.428 * X3 + 6.166 * X1X2 + 0.676 * X1X3 + 2.386 * X2X3+ 6.101 * X1X2X3 + 2.700 * X11^2 + 0.101 * X22^2 + 1.385 * X33^2 = ŷ
    Перевірка
ŷ1 = -56598.350 ≈ -56598.833
ŷ2 = -61844.287 ≈ -61843.500
ŷ3 = -175684.038 ≈ -175684.167
ŷ4 = -192653.975 ≈ -192652.833
ŷ5 = 325511.635 ≈ 325511.167
ŷ6 = 357072.364 ≈ 357073.167
ŷ7 = 953301.280 ≈ 953301.167
ŷ8 = 1046345.676 ≈ 1046346.833
ŷ9 = -406073.063 ≈ -406073.495
ŷ10 = 964524.457 ≈ 964523.989
ŷ11 = 41298.010 ≈ 41297.969
ŷ12 = 502847.989 ≈ 502847.129
ŷ13 = 249835.888 ≈ 249836.907
ŷ14 = 294119.239 ≈ 294117.320
ŷ15 = 271951.660 ≈ 271951.667
Критерій Фішера
Рівняння регресії адекватне  оригіналу

Process finished with exit code 0
```