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

Лабораторна робота 6
з дисципліни «Методи оптимізації та планування експерименту»

Виконав:
Студент 2 курсу ФІОТ
групи ІО-91
Самойленко Т.П.

Перевірив:
Регіда П.Г.

Київ 2021

Мета роботи: Провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи **рототабельний** композиційний план.

Варіант:

120	-30	20	-70	-10	-70	-40	2,1+1,7*x1+6,8*x2+6,6*x3+9,5*x1*x1+1,0*x2*x2+3,9*x3*x3+3,0*x1*x2+0,1*x1*x3+4,5*x2*x3+1,8*x1*x2*x3

Роздруківка програми:

```
from math import fabs, sqrt
import time
m = 2
p = 0.95
N = 15
x1_min = -30
x1_max = 20
x2_min = -70
x2_max = -10
x3_min = -70
x3_max = -40
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

average_y = None
matrix = None
dispersion_b2 = None
student_lst = None
d = None
q = None
f3 = None


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 = 2.1 + 1.7 * X1 + 6.8 * X2 + 6.6 * X3 + 9.5 * X1 * X1 + 1.0 * X2 * X2 + 3.9
* X3 * X3 + 3.0 * X1 * X2 + \
```

```
            0.1 * X1 * X3 + 4.5 * X2 * X3 + 1.8 * 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]

def run_experiment():
    adekvat = False
    odnorid = False

    global average_y
    global matrix
    global dispersion_b2
    global student_lst
    global d
    global q
    global m
    global f3
    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")
    return adekvat


if __name__ == '__main__':
    start = time.time()
    cnt = 0
    adekvat = 0

    while (time.time() - start) <= 10:
        cnt += 1

        try:
            adekvat += run_experiment()
        except Exception:
            continue

    print(f'За 10 секунд експеремент був адекватним {adekvat} разів з {cnt}')
```

Результати роботи програми:

Матриця планування експеременту:

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3	Yi ->		
-30.000	-70.000	-70.000	2100.000	2100.000	4900.000	-147000.000	900.000	4900.000	4900.000	-204470.900	-204466.900	-204465.900
-30.000	-70.000	-40.000	2100.000	1200.000	2800.000	-84000.000	900.000	4900.000	1600.000	-113274.900	-113275.900	-113277.900
-30.000	-10.000	-70.000	300.000	2100.000	700.000	-21000.000	900.000	100.000	4900.000	-6363.900	-6362.900	-6361.900
-30.000	-10.000	-40.000	300.000	1200.000	400.000	-12000.000	900.000	100.000	1600.000	-4268.900	-4269.900	-4266.900
20.000	-70.000	-70.000	-1400.000	-1400.000	4900.000	98000.000	400.000	4900.000	4900.000	221020.100	221019.100	221020.100
20.000	-70.000	-40.000	-1400.000	-800.000	2800.000	56000.000	400.000	4900.000	1600.000	123355.100	123359.100	123353.100
20.000	-10.000	-70.000	-200.000	-1400.000	700.000	14000.000	400.000	100.000	4900.000	50122.100	50126.100	50121.100
20.000	-10.000	-40.000	-200.000	-800.000	400.000	8000.000	400.000	100.000	1600.000	25360.100	25360.100	25365.100
-48.250	-40.000	-55.000	1930.000	2653.750	2200.000	-106150.000	2328.062	1600.000	3025.000	-140311.456	-140316.456	-140316.456
38.250	-40.000	-55.000	-1530.000	-2103.750	2200.000	84150.000	1463.062	1600.000	3025.000	183296.344	183298.344	183296.344
-5.000	-91.900	-55.000	459.500	275.000	5054.500	-25272.500	25.000	8445.610	3025.000	-1853.960	-1852.960	-1849.960
-5.000	11.900	-55.000	-59.500	275.000	-654.500	3272.500	25.000	141.610	3025.000	14678.380	14677.380	14678.380
-5.000	-40.000	-80.950	200.000	404.750	3238.000	-16190.000	25.000	1600.000	6552.903	12650.625	12647.625	12653.625
-5.000	-40.000	-29.050	200.000	145.250	1162.000	-5810.000	25.000	1600.000	843.903	46.115	42.115	41.115
-5.000	-40.000	-55.000	200.000	275.000	2200.000	-11000.000	25.000	1600.000	3025.000	3721.100	3716.100	3719.100

```
Критерій Кохрена:
Дисперсія однорідна при рівні значимості 0.05.
Отримане рівняння регресії з урахуванням критерія Стьюдента
11.505 + 1.461 * X1 + 6.856 * X2 + 7.009 * X3 + 2.999 * X1X2 + 0.096 * X1X3 + 4.501 * X2X3+ 1.800 * X1X2X3 + 9.501 * X11^2 + 1.000 * X22^2 + 3.903 * X33^2 = ŷ
    Перевірка
ŷ1 = -204467.361 ≈ -204467.900
ŷ2 = -113276.869 ≈ -113276.233
ŷ3 = -6362.527 ≈ -6362.900
ŷ4 = -4269.369 ≈ -4268.567
ŷ5 = 221020.239 ≈ 221019.767
ŷ6 = 123355.065 ≈ 123355.767
ŷ7 = 50123.406 ≈ 50123.100
ŷ8 = 25360.898 ≈ 25361.767
ŷ9 = -140314.647 ≈ -140314.790
ŷ10 = 183297.308 ≈ 183297.010
ŷ11 = -1852.266 ≈ -1852.293
ŷ12 = 14678.460 ≈ 14678.047
ŷ13 = 12649.487 ≈ 12650.625
ŷ14 = 44.693 ≈ 43.115
ŷ15 = 3718.764 ≈ 3718.767
Критерій Фішера
Рівняння регресії адекватне  оригіналу
За 10 секунд експеремент був адекватним 722 разів з 723
```

Висновок:

В даній лабораторній роботі я провів трьохфакторний експеримент і отримав адекватну модель – рівняння регресії, використовуючи ротабельний композиційний план.