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-93
Миколаєнко І.І.

Перевірив:
Регіда П.Г.

Київ 2021

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

Варіант:

320	15	45	-15	45	15	20	3,3+7,8*x1+2,9*x2+7,7*x3+8,9*x1*x1+0,1*x2*x2+5,6*x3*x3+1,1*x1*x2+0,9*x1*x3+1,1*x2*x3+2,7*x1*x2*x3

Роздруківка програми:

```
from math import fabs, sqrt
import time
m = 2
p = 0.95
N = 15
x1_min = 15
x1_max = 45
x2_min = -15
x2_max = 45
x3_min = 15
x3_max = 20
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 = 3.3 + 7.8 * X1 + 2.9 * X2 + 7.7 * X3 + 8.9 * X1 * X1 + 0.1 * X2 * X2 + 5.6
* X3 * X3 + 1.1 * X1 * X2 + \
```

```
            0.9 * X1 * X3 + 1.1 * X2 * X3 + 2.7 * 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 ->
    15.000      -15.000      15.000     -225.000     225.000     -225.000    -3375.000    225.000     225.000     225.000    -5923.700    -5928.700    -5932.700
    15.000      -15.000      20.000     -225.000     300.000     -300.000    -4500.000    225.000     225.000     400.000    -7963.700    -7959.700    -7963.700
    15.000       45.000      15.000      675.000     225.000      675.000    10125.000    225.000    2025.000     225.000    32851.300    32854.300    32853.300
    15.000       45.000      20.000      675.000     300.000      900.000    13500.000    225.000    2025.000     400.000    43300.300    43304.300    43298.300
    45.000      -15.000      15.000     -675.000     675.000     -225.000   -10125.000   2025.000     225.000     225.000    -7989.700    -7985.700    -7991.700
    45.000      -15.000      20.000     -675.000     900.000     -300.000   -13500.000   2025.000     225.000     400.000   -15964.700   -15960.700   -15958.700
    45.000       45.000      15.000     2025.000     675.000      675.000    30375.000   2025.000    2025.000     225.000   105672.300   105675.300   105677.300
    45.000       45.000      20.000     2025.000     900.000      900.000    40500.000   2025.000    2025.000     400.000   134477.300   134484.300   134485.300
     4.050       15.000      17.500       60.750      70.875      262.500     1063.125     16.403     225.000     306.250     5388.422     5389.422     5388.422
    55.950       15.000      17.500      839.250     979.125      262.500    14686.875   3130.403     225.000     306.250    71961.742    71967.742    71963.742
    30.000      -36.900      17.500    -1107.000     525.000     -645.750   -19372.500    900.000    1361.610     306.250   -43633.074   -43632.074   -43636.074
    30.000       66.900      17.500     2007.000     525.000     1170.750    35122.500    900.000    4475.610     306.250   109537.396   109541.396   109540.396
    30.000       15.000      13.175      450.000     395.250      197.625     5928.750    900.000     225.000     173.581    26461.537    26462.537    26460.537
    30.000       15.000      21.825      450.000     654.750      327.375     9821.250    900.000     225.000     476.331    39110.567    39112.567    39112.567
    30.000       15.000      17.500      450.000     525.000      262.500     7875.000    900.000     225.000     306.250    32681.800    32680.800    32677.800
```

```
Критерій Кохрена:
Дисперсія однорідна при рівні значимості 0.05.
Отримане рівняння регресії з урахуванням критерія Стьюдента
18.267 + 7.490 * X1 + 2.658 * X2 + 6.129 * X3 + 1.105 * X1X2 + 0.907 * X1X3 + 1.111 * X2X3+ 2.700 * X1X2X3 + 8.903 * X11^2 + 0.101 * X22^2 + 5.645 * X33^2 = ŷ
    Перевірка
ŷ1 = -5927.358 ≈ -5928.367
ŷ2 = -7961.316 ≈ -7962.367
ŷ3 = 32854.615 ≈ 32852.967
ŷ4 = 43302.657 ≈ 43300.967
ŷ5 = -7989.258 ≈ -7989.033
ŷ6 = -15961.549 ≈ -15961.367
ŷ7 = 105675.382 ≈ 105674.967
ŷ8 = 134482.758 ≈ 134482.300
ŷ9 = 5386.351 ≈ 5388.756
ŷ10 = 71964.854 ≈ 71964.409
ŷ11 = -43633.981 ≈ -43633.741
ŷ12 = 109538.010 ≈ 109539.729
ŷ13 = 26460.606 ≈ 26461.537
ŷ14 = 39110.871 ≈ 39111.900
ŷ15 = 32680.147 ≈ 32680.133
Критерій Фішера
Рівняння регресії адекватне  оригіналу
За 10 секунд експеремент був адекватним 664 разів з 665
```

Висновок:

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