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»**

Виконав:
студент групи ІО-93
Бернадін Олександр
Залікова книжка № ІО-9302

Перевірив:
ас. Регіда П.Г.

Київ – 2021

1. **Тема**: «Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами.»
2. **Мета**: Провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.
3. **Завдання**:Взяти рівняння з урахуванням квадратичних членів.
    1. Ознайомитися з теоретичними відомостями.
    2. Вибрати з таблиці варіантів і записати в протокол інтервали значень x_1, x_2, x_3. Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1;+ l; - l; 0 для $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$.
    3. Значення функції відгуку знайти за допомогою підстановки в формулу:

$$y_i = f(x_1, x_2, x_3) + random(10)-5,$$

де $f(x_1, x_2, x_3)$ вибирається по номеру в списку в журналі викладача.

    4. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках використовувати натуральні значення факторів.
    5. Зробити висновки по виконаній роботі.

Порядок виконання роботи:

1) Вибір рівняння регресії (лінійна форма, рівняння з урахуванням ефекту взаємодії і з урахуванням квадратичних членів);

2) Вибір кількості повторів кожної комбінації (m = 2);

3) Складення матриці планування експерименту і вибір кількості рівнів (N)

4) Проведення експериментів;

5) Перевірка однорідності дисперсії. Якщо не однорідна – повертаємося на п. 2 і збільшуємо m на 1);

6) Розрахунок коефіцієнтів рівняння регресії. При розрахунку використовувати ***натуральні*** *значення x_1, x_2 и x_3.*

7) Перевірка нуль-гіпотези. Визначення значимих коефіцієнтів;

8) Перевірка адекватності моделі рівняння оригіналу. При неадекватності – повертаємося на п.1, змінивши при цьому рівняння регресії;

Варіант завдання:

301	-10	50	20	60	-10	5	7,1+9,5*x1+7,9*x2+4,9*x3+1,5*x1*x1+0,9*x2*x2+9,7*x3*x3+1,6*x1*x2+0,1*x1*x3+3,8*x2*x3+4,9*x1*x2*x3

Роздруківка тексту програми:

```
from math import fabs, sqrt

m = 2
p = 0.95 #ймовірність
N = 15

#значення за варіантом
x1_min = -10
x1_max = 50
x2_min = 20
x2_max = 60
x3_min = -10
x3_max = 5

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 Check:
    def get_cohren_value(size, qty_of_selections, significance):
        from _pydecimal import Decimal
        from scipy.stats import f
        size += 1
        partResult1 = significance / (size - 1)
        params = [partResult1, qty_of_selections, (size - 1 - 1) * qty_of_selections]
        fisher = f.isf(*params)
        result = fisher / (fisher + (size - 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 =
7.1+9.5*x1+7.9*x2+4.9*x3+1.5*x1*x1+0.9*x2*x2+9.7*x3*x3+1.6*x1*x2+0.1*x1*x3+3.8*x2*x3+4.9*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 = Check.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 = Check.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\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]))
```

```
    print("-----------------------------------------------------------------")
    for i in range(N):
        print("-----------------------------------------------------------------")
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(beta, i), average_y[i]))

    while not odnorid:
        print("-----------------------------------------------------------------")
        print("\nМатриця 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("-----------------------------------------------------------------")
        print("\nКритерій Кохрена:")
        Gt = Check.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("-----------------------------------------------------------------")
    print("\nОтримане 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\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("-----------------------------------------------------------------")
        print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check_result(student_lst, i), average_y[i]))

    print("За критерієм 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__':
    run_experiment()
```

Результати роботи програми:

Отримане рівняння регресії

6.616 + 9.444 * X1 + 7.972 * X2 + 5.118 * X3 + 1.601 * X1X2 + 0.094 * X1X3 + 3.797 * X2X3+ 4.900 * X1X2X3 + 1.500 * X11^2 + 0.899 * X22^2 + 9.691 * X33^2 = ŷ

Перевірка

ŷ1 = 10229.034 ≈ 10228.600

ŷ2 = -3996.447 ≈ -3995.400

ŷ3 = 30865.026 ≈ 30866.100

ŷ4 = -10482.955 ≈ -10480.400

ŷ5 = -42541.453 ≈ -42542.900

ŷ6 = 31520.566 ≈ 31520.600

ŷ7 = -135665.461 ≈ -135665.400

ŷ8 = 87679.058 ≈ 87680.600

ŷ9 = 16252.127 ≈ 16250.215

ŷ10 = -20779.366 ≈ -20778.935

ŷ11 = -293.624 ≈ -292.621

ŷ12 = -10161.557 ≈ -10164.041

ŷ13 = -56968.436 ≈ -56967.464

ŷ14 = 47624.369 ≈ 47621.916

ŷ15 = -6303.536 ≈ -6303.525

Матриця планування експеременту:

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3	Yi ->
-10.000	20.000	-10.000	-200.000	100.000	-200.000	2000.000	100.000	400.000	100.000	10230.100	10227.100
-10.000	20.000	5.000	-200.000	-50.000	100.000	-1000.000	100.000	400.000	25.000	-3995.900	-3994.900
-10.000	60.000	-10.000	-600.000	100.000	-600.000	6000.000	100.000	3600.000	100.000	30868.100	30864.100
-10.000	60.000	5.000	-600.000	-50.000	300.000	-3000.000	100.000	3600.000	25.000	-10482.900	-10477.900
50.000	20.000	-10.000	1000.000	-500.000	-200.000	-10000.000	2500.000	400.000	100.000	-42542.900	-42542.900
50.000	20.000	5.000	1000.000	250.000	100.000	5000.000	2500.000	400.000	25.000	31524.100	31517.100
50.000	60.000	-10.000	3000.000	-500.000	-600.000	-30000.000	2500.000	3600.000	100.000	-135663.900	-135666.900
50.000	60.000	5.000	3000.000	250.000	300.000	15000.000	2500.000	3600.000	25.000	87679.100	87682.100
-31.900	40.000	-2.500	-1276.000	79.750	-100.000	3190.000	1017.610	1600.000	6.250	16250.215	16250.215
71.900	40.000	-2.500	2876.000	-179.750	-100.000	-7190.000	5169.610	1600.000	6.250	-20783.435	-20774.435
20.000	5.400	-2.500	108.000	-50.000	-13.500	-270.000	400.000	29.160	6.250	-294.121	-291.121

20.000 74.600 -2.500 1492.000 -50.000 -186.500 -3730.000 400.000 5565.160 6.250 -10164.041 -10164.041

20.000 40.000 -15.475 800.000 -309.500 -619.000 -12380.000 400.000 1600.000 239.476 -56964.964 -56969.964

20.000 40.000 10.475 800.000 209.500 419.000 8380.000 400.000 1600.000 109.726 47622.916 47620.916

20.000 40.000 -2.500 800.000 -50.000 -100.000 -2000.000 400.000 1600.000 6.250 -6307.525 -6299.525

Критерій Кохрена:

Дисперсія однорідна при рівні значимості 0.05.

Отримане рівняння регресії з урахуванням критерія Стьюдента

6.616 + 9.444 * X1 + 7.972 * X2 + 5.118 * X3 + 1.601 * X1X2 + 0.094 * X1X3 + 3.797 * X2X3+ 4.900 * X1X2X3 + 1.500 * X11^2 + 0.899 * X22^2 + 9.691 * X33^2 = ŷ

Перевірка

ŷ1 = 10229.034 ≈ 10228.600

ŷ2 = -3996.447 ≈ -3995.400

ŷ3 = 30865.026 ≈ 30866.100

ŷ4 = -10482.955 ≈ -10480.400

ŷ5 = -42541.453 ≈ -42542.900

ŷ6 = 31520.566 ≈ 31520.600

ŷ7 = -135665.461 ≈ -135665.400

ŷ8 = 87679.058 ≈ 87680.600

ŷ9 = 16252.127 ≈ 16250.215

ŷ10 = -20779.366 ≈ -20778.935

ŷ11 = -293.624 ≈ -292.621

ŷ12 = -10161.557 ≈ -10164.041

ŷ13 = -56968.436 ≈ -56967.464

ŷ14 = 47624.369 ≈ 47621.916

ŷ15 = -6303.536 ≈ -6303.525

За критерієм Фішера:

Рівняння регресії адекватне оригіналу

Висновки:

У ході виконання лабораторної роботи я провів повний трьохфакторний експеримент при використанні рівняння регресії з квадратичними членами. Закріпив отримані знання практичним їх використанням при написанні програми, що реалізує завдання лабораторної роботи. Мета лабораторної роботи досягнута.