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 ІВ-91
Вігор Дмитро
Залікова книжка № 9106

Перевірив:
ас. Регіда П.Г.

Київ – 2021

Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами

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

Завдання на лабораторну роботу:

1. Ознайомитися з теоретичними відомостями.
2. Вибрати з таблиці варіантів і записати в протокол інтервали значень x1, x2, x3. Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1;+; -; 0 для 1, 2, 3.
3. Значення функції відгуку знайти за допомогою підстановки в формулу:

$$yi = f(x1, x2, x3) + random(10)-5,$$

де f(x1, x2, x3) 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.

Варіант завдання:

106	10	40	25	45	40	45	6,7+9,1*x1+1,6*x2+9,1*x3+3,3*x1*x1+0,2*x2*x2+6,1*x3*x3+8,5*x1*x2+0,7*x1*x3+6,6*x2*x3+8,1*x1*x2*x3

Роздруківка тексту програми:

```
from math import fabs, sqrt
import time
m = 2
p = 0.95
N = 15
x1_min = 10
x1_max = 40
x2_min = 25
x2_max = 45
x3_min = 40
x3_max = 45
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(self, 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 = 6.7 + 9.1 * X1 + 1.6 * X2 + 9.1 * X3 + 3.3 * X1 * X1 + 0.2 * X2 * X2 + 6.1
* X3 * X3 + 8.5 * X1 * X2 + \
            0.7 * X1 * X3 + 6.6 * X2 * X3 + 8.1 * 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]

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 ->	
10.000	25.000	40.000	250.000	400.000	1000.000	10000.000	100.000	625.000	1600.000	100723.700	100717.700
10.000	25.000	45.000	250.000	450.000	1125.000	11250.000	100.000	625.000	2025.000	114345.700	114343.700
10.000	45.000	40.000	450.000	400.000	1800.000	18000.000	100.000	2025.000	1600.000	172815.700	172811.700
10.000	45.000	45.000	450.000	450.000	2025.000	20250.000	100.000	2025.000	2025.000	195194.700	195191.700
40.000	25.000	40.000	1000.000	1600.000	1000.000	40000.000	1600.000	625.000	1600.000	356156.700	356160.700
40.000	25.000	45.000	1000.000	1800.000	1125.000	45000.000	1600.000	625.000	2025.000	400258.700	400266.700
40.000	45.000	40.000	1800.000	1600.000	1800.000	72000.000	1600.000	2025.000	1600.000	627754.700	627754.700
40.000	45.000	45.000	1800.000	1800.000	2025.000	81000.000	1600.000	2025.000	2025.000	704909.700	704911.700
-0.950	35.000	42.500	-33.250	-40.375	1487.500	-1413.125	0.902	1225.000	1806.250	9764.208	9771.208
50.950	35.000	42.500	1783.250	2165.375	1487.500	75788.125	2595.903	1225.000	1806.250	661117.398	661114.398
25.000	17.700	42.500	442.500	1062.500	752.250	18806.250	625.000	313.290	1806.250	175597.028	175589.028
25.000	52.300	42.500	1307.500	1062.500	2222.750	55568.750	625.000	2735.290	1806.250	490969.838	490964.838
25.000	35.000	38.175	875.000	954.375	1336.125	33403.125	625.000	1225.000	1457.331	299327.109	299324.109
25.000	35.000	46.825	875.000	1170.625	1638.875	40971.875	625.000	1225.000	2192.581	367340.249	367339.249
25.000	35.000	42.500	875.000	1062.500	1487.500	37187.500	625.000	1225.000	1806.250	333221.075	333218.075

Критерій Кохрена:

Отримане рівняння регресії

235.042 + 5.597 * X1 + -0.409 * X2 + 0.002 * X3 + 8.587 * X1X2 + 0.788 * X1X3 + 6.647 * X2X3+ 8.098 * X1X2X3 + 3.300 * X11^2 + 0.202 * X22^2 + 6.186 * X33^2 = ŷ

Перевірка

ŷ1 = 100721.262 ≈ 100722.700
ŷ2 = 114342.788 ≈ 114343.200
ŷ3 = 172813.260 ≈ 172814.200
ŷ4 = 195197.286 ≈ 195197.200
ŷ5 = 356158.849 ≈ 356160.200
ŷ6 = 400265.375 ≈ 400265.700
ŷ7 = 627750.847 ≈ 627751.700
ŷ8 = 704913.373 ≈ 704913.200
ŷ9 = 9765.654 ≈ 9764.708
ŷ10 = 661116.643 ≈ 661115.898
ŷ11 = 175592.949 ≈ 175591.528
ŷ12 = 490966.107 ≈ 490965.838
ŷ13 = 299324.141 ≈ 299322.109
ŷ14 = 367345.408 ≈ 367345.749
ŷ15 = 333219.063 ≈ 333219.075

Матриця планування експеременту:

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3	Yi ->	
10.000	25.000	40.000	250.000	400.000	1000.000	10000.000	100.000	625.000	1600.000	100722.700	100721.700
10.000	25.000	45.000	250.000	450.000	1125.000	11250.000	100.000	625.000	2025.000	114341.700	114346.700
10.000	45.000	40.000	450.000	400.000	1800.000	18000.000	100.000	2025.000	1600.000	172810.700	172816.700
10.000	45.000	45.000	450.000	450.000	2025.000	20250.000	100.000	2025.000	2025.000	195194.700	195196.700
40.000	25.000	40.000	1000.000	1600.000	1000.000	40000.000	1600.000	625.000	1600.000	356158.700	356155.700
40.000	25.000	45.000	1000.000	1800.000	1125.000	45000.000	1600.000	625.000	2025.000	400265.700	400259.700
40.000	45.000	40.000	1800.000	1600.000	1800.000	72000.000	1600.000	2025.000	1600.000	627755.700	627750.700
40.000	45.000	45.000	1800.000	1800.000	2025.000	81000.000	1600.000	2025.000	2025.000	704917.700	704915.700
-0.950	35.000	42.500	-33.250	-40.375	1487.500	-1413.125	0.902	1225.000	1806.250	9771.208	9767.208
50.950	35.000	42.500	1783.250	2165.375	1487.500	75788.125	2595.903	1225.000	1806.250	661113.398	661115.398
25.000	17.700	42.500	442.500	1062.500	752.250	18806.250	625.000	313.290	1806.250	175596.028	175589.028
25.000	52.300	42.500	1307.500	1062.500	2222.750	55568.750	625.000	2735.290	1806.250	490968.838	490965.838
25.000	35.000	38.175	875.000	954.375	1336.125	33403.125	625.000	1225.000	1457.331	299320.109	299327.109
25.000	35.000	46.825	875.000	1170.625	1638.875	40971.875	625.000	1225.000	2192.581	367348.249	367345.249
25.000	35.000	42.500	875.000	1062.500	1487.500	37187.500	625.000	1225.000	1806.250	333221.075	333218.075

Критерій Кохрена:

За 10 секунд експеремент був адекватним 0 разів з 5532

Висновки:

Під час виконання лабораторної роботи було змодельовано трьохфакторний експеримент при використанні лінійного рівняння регресії, рівняння регресії з ефектом взаємодії та рівняння регресії з квадратичними членами, складено матрицю планування експерименту, було визначено коефіцієнти рівнянь регресії (натуралізовані та нормовані), для форми з квадратичними членами - натуралізовані, виконано перевірку правильності розрахунку коефіцієнтів рівнянь регресії. Також було проведено 3 статистичні перевірки(використання критеріїв Кохрена, Стьюдента та Фішера) для кожної форми рівняння регресії . При виявленні неадекватності лінійного рівняння регресії оригіналу було застосовано ефект взаємодії факторів, при неадекватності і такого рівняння регресії було затосовано рівняння регресії з квадратичними членами. Довірча ймовірність в даній роботі дорівнює 0.95, відповідно рівень значимості q = 0.05.