Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Методи оптимізації та планування експерименту

Лабораторна робота №6

«Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

Виконав:
студент групи ІО-93
Варченко Є. В.
Номер у списку групи – 3

Перевірив:
ас. Регіда П. Г.

Київ – 2021

Тема: «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

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:

$$y_i = f(x_1, x_2, x_3) + \text{random}(10)-5,$$

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.

Варіант

№ варіанту	x_1		x_2		x_3		$f(x_1, x_2, x_3)$
	min	max	min	max	min	max	
303	-20	30	-20	40	-20	-10	6,1+5,4*x1+0,2*x2+7,4*x3+8,8*x1*x1+0,8*x2*x2+5,0*x3*x3+4,5*x1*x2+0,5*x1*x3+4,7*x2*x3+2,6*x1*x2*x3

Код програми

```
from random import *
from math import *
from numpy.linalg import *
from _pydecimal import *
from scipy.stats import *

m = 3
p = 0.95
N = 15

x1_min = -20
x1_max = 30
x2_min = -20
x2_max = 40
x3_min = -20
x3_max = -10

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

planning_matrix = [
    [-1, -1, -1, +1, +1, +1, -1, +1, +1, +1],
    [-1, -1, +1, +1, -1, -1, +1, +1, +1, +1],
    [-1, +1, -1, -1, +1, -1, +1, +1, +1, +1],
    [-1, +1, +1, -1, -1, +1, -1, +1, +1, +1],
    [+1, -1, -1, -1, -1, +1, +1, +1, +1, +1],
```

```
    [+1, -1, +1, -1, +1, -1, -1, +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]
]

x_matrix = [[] for x in range(N)]


def generate_matrix():
    def f(X1, X2, X3):
        y = 6.1 + 5.4*X1 + 0.2*X2 + 7.4*X3 + 8.8*X1*X1 + 0.8*X2*X2 + 5.0*X3*X3 +
4.5*X1*X2 + 0.5*X1*X3 + 4.7*X2*X3 + 2.6*X1*X2*X3 + randrange(0, 10) - 5

        return y

    matrix_with_y = [[f(x_matrix[j][0], x_matrix[j][1], x_matrix[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(list, orientation):
    average = []

    if orientation == 1:
        for rows in range(len(list)):
            average.append(sum(list[rows]) / len(list[rows]))
    else:
        for column in range(len(list[0])):
            number_list = []

            for rows in range(len(list)):
                number_list.append(list[rows][column])

            average.append(sum(number_list) / len(number_list))

    return average


def a(first, second):
    need_a = 0

    for j in range(N):
        need_a += x_matrix[j][first - 1] * x_matrix[j][second - 1] / N

    return need_a


def find_known(number):
    need_a = 0

    for j in range(N):
        need_a += average_y[j] * x_matrix[j][number - 1] / 15

    return need_a
```

```
def solve_equation(list_1, list_2):
    solver = solve(list_1, list_2)

    return solver


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

    return y_i


def student_test(b_list, number_x=10):
    dispersion_b = sqrt(dispersion_b2)

    for column in range(number_x + 1):
        t_practical = 0
        t_theoretical = Check.get_student_value(f3, q)

        for row in range(N):
            if column == 0:
                t_practical += average_y[row] / N
            else:
                t_practical += average_y[row] * planning_matrix[row][column - 1]

        if fabs(t_practical / dispersion_b) < t_theoretical:
            b_list[column] = 0
```

```
    return b_list


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_list,
row))) / (N - d)

    F_practical = dispersion_ad / dispersion_b2
    F_theoretical = Check.get_fisher_value(f3, f4, q)

    return F_practical < F_theoretical


class Check:
    def get_cochran_value(size_of_selections, quantity_of_selections,
significance):
        size_of_selections += 1

        partResult = significance / (size_of_selections - 1)
        params = [partResult, quantity_of_selections, (size_of_selections - 1 - 1)
* quantity_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):
        return Decimal(abs(t.ppf(significance / 2,
f3))).quantize(Decimal('.0001')).__float__()

    def get_fisher_value(f3, f4, significance):
```

```
        return Decimal(abs(f.isf(significance, f4,
f3))).quantize(Decimal('.0001')).__float__()


for i in range(len(x_matrix)):
    if i < 8:
        x_1 = x1_min if planning_matrix[i][0] == -1 else x1_max
        x_2 = x2_min if planning_matrix[i][1] == -1 else x2_max
        x_3 = x3_min if planning_matrix[i][2] == -1 else x3_max
    else:
        x_list = x(planning_matrix[i][0], planning_matrix[i][1],
planning_matrix[i][2])
        x_1, x_2, x_3 = x_list

    x_matrix[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]

adequate = False
homogeneous = False

while not adequate:
    matrix_y = generate_matrix()
    average_x = find_average(x_matrix, 0)
    average_y = find_average(matrix_y, 1)
    matrix = [(x_matrix[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("Рівняння регресії:")
    print("y = {:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 +
{:.3f} * X1X3 + {:.3f} * X2X3"
          "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} * X33^2\n\
nПеревірка:".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("y{} = {:.3f} = {:.3f}".format((i + 1), check_result(beta, i),
average_y[i]))
```

```
    while not homogeneous:
        print("\nМатриця планування експерименту:")
        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("")

        Gt = Check.get_cochran_value(f2, f1, q)
        if Gt > Gp:
```

```
            print("Дисперсія однорідна при рівні значимості {:.2f}.".format(q))
            homogeneous = 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_list = list(student_test(beta))

    print("\nРівняння регресії з урахуванням критерія Стьюдента:")
    print("y = {:.3f} + {:.3f} * X1 + {:.3f} * X2 + {:.3f} * X3 + {:.3f} * X1X2 +
{:.3f} * X1X3 + {:.3f} * X2X3"
          "+ {:.3f} * X1X2X3 + {:.3f} * X11^2 + {:.3f} * X22^2 + {:.3f} * X33^2\n\
nПеревірка:"
          .format(student_list[0], student_list[1], student_list[2],
student_list[3], student_list[4], student_list[5],
                  student_list[6], student_list[7], student_list[8],
student_list[9], student_list[10]))

    for i in range(N):
        print("y{} = {:.3f} = {:.3f}".format((i + 1), check_result(student_list,
i), average_y[i]))

    d = 11 - student_list.count(0)

    if fisher_test():
        print("\nРівняння регресії адекватне оригіналу")
        adequate = True
    else:
        print("\nРівняння регресії неадекватне оригіналу\n\t Проводимо експеремент
повторно")
```

Результати роботи програми

Рівняння регресії:
y = -2.136 + 5.529 * X1 + 0.357 * X2 + 6.203 * X3 + 4.498 * X1X2 + 0.510 * X1X3 + 4.709 * X2X3+ 2.600 * X1X2X3 + 8.800 * X11^2 + 0.800 * X22^2 + 4.961 * X33^2

Перевірка:
y1 = -11332.332 = -11331.900
y2 = -3402.516 = -3402.233
y3 = 40998.638 = 40998.767
y4 = 20554.454 = 20554.433
y5 = 40334.411 = 40334.767
y6 = 22519.893 = 22520.100
y7 = -49835.286 = -49835.233
y8 = -18025.804 = -18025.900
y9 = 26550.937 = 26550.625
y10 = 4136.611 = 4136.475
y11 = 12806.649 = 12806.075
y12 = -10733.225 = -10733.099
y13 = -1066.409 = -1066.806
y14 = -426.051 = -426.103
y15 = -1117.403 = -1117.400

Матриця планування експерименту:

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1X1	X2X2	X3X3	Yi ->		
-20.000	-20.000	-20.000	400.000	400.000	400.000	-8000.000	400.000	400.000	400.000	-11329.900	-11332.900	-11332.900
-20.000	-20.000	-10.000	400.000	200.000	200.000	-4000.000	400.000	400.000	100.000	-3404.900	-3396.900	-3404.900
-20.000	40.000	-20.000	-800.000	400.000	-800.000	16000.000	400.000	1600.000	400.000	40999.100	40995.100	41002.100
-20.000	40.000	-10.000	-800.000	200.000	-400.000	8000.000	400.000	1600.000	100.000	20551.100	20556.100	20556.100
30.000	-20.000	-20.000	-600.000	-600.000	400.000	12000.000	900.000	400.000	400.000	40337.100	40336.100	40331.100
30.000	-20.000	-10.000	-600.000	-300.000	200.000	6000.000	900.000	400.000	100.000	22519.100	22521.100	22520.100
30.000	40.000	-20.000	1200.000	-600.000	-800.000	-24000.000	900.000	1600.000	400.000	-49836.900	-49832.900	-49835.900
30.000	40.000	-10.000	1200.000	-300.000	-400.000	-12000.000	900.000	1600.000	100.000	-18029.900	-18023.900	-18023.900
-38.250	10.000	-15.000	-382.500	573.750	-150.000	5737.500	1463.062	100.000	225.000	26547.625	26552.625	26551.625
48.250	10.000	-15.000	482.500	-723.750	-150.000	-7237.500	2328.062	100.000	225.000	4137.475	4140.475	4131.475
5.000	-41.900	-15.000	-209.500	-75.000	628.500	3142.500	25.000	1755.610	225.000	12804.408	12810.408	12803.408
5.000	61.900	-15.000	309.500	-75.000	-928.500	-4642.500	25.000	3831.610	225.000	-10730.432	-10734.432	-10734.432
5.000	10.000	-23.650	50.000	-118.250	-236.500	-1182.500	25.000	100.000	559.322	-1068.473	-1066.473	-1065.473
5.000	10.000	-6.350	50.000	-31.750	-63.500	-317.500	25.000	100.000	40.322	-422.103	-427.103	-429.103
5.000	10.000	-15.000	50.000	-75.000	-150.000	-750.000	25.000	100.000	225.000	-1114.400	-1117.400	-1120.400

Дисперсія однорідна при рівні значимості 0.05.

Рівняння регресії з урахуванням критерія Стьюдента:
y = -2.136 + 5.529 * X1 + 0.357 * X2 + 6.203 * X3 + 4.498 * X1X2 + 0.510 * X1X3 + 4.709 * X2X3+ 2.600 * X1X2X3 + 8.800 * X11^2 + 0.800 * X22^2 + 4.961 * X33^2

Перевірка:
y1 = -11332.332 = -11331.900
y2 = -3402.516 = -3402.233
y3 = 40998.638 = 40998.767
y4 = 20554.454 = 20554.433
y5 = 40334.411 = 40334.767
y6 = 22519.893 = 22520.100
y7 = -49835.286 = -49835.233
y8 = -18025.804 = -18025.900
y9 = 26550.937 = 26550.625
y10 = 4136.611 = 4136.475
y11 = 12806.649 = 12806.075
y12 = -10733.225 = -10733.099
y13 = -1066.409 = -1066.806
y14 = -426.051 = -426.103
y15 = -1117.403 = -1117.400

Рівняння регресії адекватне оригіналу

Висновки

- Ознайомилися з темою роботи.
- Були здобуті необхідні навички для виконання завдань.
- Розроблено програму, яка виконує поставлену задачу.
- Вище приведені результати свідчать про успішне виконання умов завдань.
- Основну мету лабораторної роботи було досягнуто.