Національний технічний університет України
«Київський політехнічний інститут імені Ігоря 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-92
Педенко Данило Денисович
Перевірив:
ас. Регіда П. Г.

Київ
2021 р.

Лабораторна робота № 6

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 x1, x2, x3. 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:
$$y_i = f(x_1, x_2, x_3) + random(10) - 5;$$
4. де f(x_1, x_2, x_3) вибирається по номеру в списку в журналі викладача.
5. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках використовувати натуральні значення факторів.
6. Зробити висновки по виконаній роботі.

Завдання відповідно за номером варіанту:

№ варіанту	x_1		x_2		x_3		f(x_1, x_2, x_3)
	min	max	min	max	min	max	
215	10	50	-20	60	10	15	7,2+5,5*x1+6,3*x2+3,1*x3+4,6*x1*x1+0,2*x2*x2+3,2*x3*x3+4,3*x1*x2+0,9*x1*x3+8,0*x2*x3+7,0*x1*x2*x3

Роздруківка тексту програми:

```
import numpy as np
import random as ra
import math as ma
from scipy.stats import f
from copy import deepcopy
from prettytable import PrettyTable


def cochrane(eq, n, list_y, list_av_y, list_x, koef, list_xn):
    i = 0
    f1 = eq - 1
    f2 = n
    list_g = [9065, 7679, 6841, 6287, 5892, 5598, 5365, 5175, 5017, 4884]
    list_sig = []
    for i in range(len(list_y)):
        tem = 0
        for j in range(len(list_y[i])):
            tem += pow(list_y[i][j] - list_av_y[i], 2)
        list_sig.append(round(tem / len(list_y[i]), 4))
    gp = max(list_sig) / sum(list_sig)
    print("F1 = ", f1)
    print("F2 = ", f2)
    print("q = 0.05")
    print("Значення дисперсій по рядках")
    print(list_sig)
    print("\nGp = ", gp)
    for i in range(len(list_g)):
        if i == f1 - 1:
            if gp < list_g[i] / 10000:
                print("\nGp = {0} < Gt = {1}".format(gp, list_g[i] / 10000))
                print("Дисперсія однорідна\n")
                print("Оцінимо значимість коефіцієнтів регресії згідно критерію
```

```
Стьюдента")
                if student(eq, n, list_sig, list_av_y, list_x, koef, list_xn):
                    return True
                else:
                    return False
            else:
                print("Дисперсія не однорідна")


def student(eq, n, list_sig, list_av_y, list_x, koef, list_xn):
    list_t_prover = [12.71, 4.303, 3.182, 2.776, 2.571, 2.447, 2.365, 2.306, 2.262,
2.228, 2.201, 2.179, 2.160, 2.145,
                     2.131, 2.120, 2.110, 2.101, 2.093, 2.086, 2.080, 2.074, 2.069,
2.064, 2.060, 2.056, 2.052, 2.048,
                     2.045, 2.042]
    sv = sum(list_sig) / len(list_sig)
    s_sq_beta = sv / (n * eq)
    s_beta = ma.sqrt(s_sq_beta)
    list_beta = []
    new_koef = []
    no_matter_koef = []
    for i in range(len(list_x)):
        pol = 0
        for j in range(len(list_x[i])):
            pol += list_x[i][j] * list_av_y[j]
        list_beta.append(pol / len(list_av_y))
    print("m = ", eq)
    print("N = ", n)
    print("Отримані значення βi")
    print(list_beta)
    list_t = [abs(list_beta[i]) / s_beta for i in range(len(list_beta))]
    print("Отримані значення ti")
    print(list_t)
    print("\nf3 = ", (eq - 1) * n)
    print("q = 0.05")
    for i in range(len(list_t_prover)):
        if i == (eq - 1) * n - 1:
            for j in range(len(list_t)):
                if list_t[j] < list_t_prover[i]:
                    print("\nt{0} = {1} < ттабл = {2}".format(j, list_t[j],
list_t_prover[i]))
                    print("b{0} - виключається з рівняння".format(j))
                    no_matter_koef.append([j, koef[j]])
                else:
                    print("\nt{0} = {1} > ттабл = {2}".format(j, list_t[j],
list_t_prover[i]))
                    new_koef.append([j, koef[j]])
    print("\nПерепишемо рівняння враховуючи вилучених коефіцієнтів")
    print(vivod(new_koef))
    print("\nРівняння з використанням незначимих коефіцієнтів")
    print(vivod(no_matter_koef))
    list_res_y = []
    print("\nПідставимо необхідні значення X")
    for i in range(len(list_xn)):
        pol = 0
        for j in range(len(new_koef)):
            if new_koef[j][0] == 0:
                pol += new_koef[j][1]
            else:
                pol += list_xn[i][new_koef[j][0] - 1] * new_koef[j][1]
        list_res_y.append(pol)
        print("y{0} = {1}".format(i, pol))
```

```
    print("\nКритерій Фішера")
    if fisher(len(new_koef), n, eq, list_res_y, list_av_y, sv):
        return True
    else:
        return False


def fisher(d, n, eq, list_res_y, list_av_y, sv):
    f4 = n - d
    f3 = (eq - 1) * n
    temp = 0
    print("d = ", d)
    print("f3 = ", f3)
    print("f4 = ", f4)
    print("q = 0.05")
    for i in range(n):
        temp += pow((list_res_y[i] - list_av_y[i]), 2)
    sad = temp * (eq / (n - d))
    fp = sad / sv
    ft = f.ppf(q=1 - 0.05, dfn=f4, dfd=f3)
    if fp > ft:
        print("\nFp = {0} > Ft = {1}".format(fp, ft))
        print("Рівняння регресії неадекватно оригіналу")
        return True
    else:
        print("\nFp = {0} < Ft = {1}".format(fp, ft))
        print("Рівняння регресії адекватно оригіналу")
        return False


def vivod(ar):
    rivn = "y = "
    for i in range(len(ar)):
        if ar[i][0] == 0:
            rivn += str(round(ar[i][1], 4))
        elif i == 0:
            rivn += str(round(ar[i][1], 4)) + " * x{0}".format(ar[i][0])
        else:
            rivn += " + " + str(round(ar[i][1], 4)) + " * x{0}".format(ar[i][0])
    return rivn


def output_equation(res):
    rivn = "y = "
    for i in range(len(res)):
        if i == 0:
            rivn += "{0}".format(round(res[i], 4))
        else:
            rivn += " + {0} * x{1}".format(round(res[i], 4), i)
    return rivn


def find_average(lst):
    average = []
    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 matr_zor(list_x, list_delta, list_x0):
    l_zor = [1.73, 0, 0]
    copy_l = deepcopy(l_zor)
    for i in range(len(l_zor)):
        list_x.append([])
        for j in range(len(l_zor)):
            list_x[len(list_x) - 1].append(-copy_l[j] * list_delta[j] + list_x0[j])

        list_x.append([])
        for j in range(len(l_zor)):
            list_x[len(list_x) - 1].append(copy_l[j] * list_delta[j] + list_x0[j])
        copy_l.insert(0, 0)
    list_x.append(list_x0)
    return list_x


def expanded_matr(list_x):
    for i in range(len(list_x)):
        list_x[i].append(list_x[i][0] * list_x[i][1])
        list_x[i].append(list_x[i][0] * list_x[i][2])
        list_x[i].append(list_x[i][1] * list_x[i][2])
        list_x[i].append(list_x[i][0] * list_x[i][1] * list_x[i][2])
        if len(list_x) > 9:
            for j in range(3):
                list_x[i].append(list_x[i][j] ** 2)
    return list_x


def koef_rivn(mx_i, my):
    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)]
    return np.linalg.solve(unknown, known)


def main1(m, n):
    global array_xn_zor, array_aver_y
    array_xd = [[10, 50], [-20, 60], [10, 15]]
    array_koef_var = [7.2, 5.5, 6.3, 3.1, 4.3, 0.9, 8.0, 7.0, 4.6, 0.2, 3.2]
    array_xp = np.array(
        [[1, -1, -1, -1],
         [1, -1, -1, 1],
         [1, -1, 1, -1],
         [1, -1, 1, 1],
         [1, 1, -1, -1],
         [1, 1, -1, 1],
         [1, 1, 1, -1],
         [1, 1, 1, 1]])
    array_x0 = [(max(array_xd[i]) + min(array_xd[i])) / len(array_xd[i]) for i in
range(len(array_xd))]
    array_xd_delta = [max(array_xd[i]) - array_x0[i] for i in range(len(array_xd))]
    array_y = []
    array_xn_zor = []
    for i in range(len(array_xp)):
        array_xn_zor.append([])
        for j in range(len(array_xd)):
            if array_xp[i][j + 1] == -1:
                array_xn_zor[i].append(min(array_xd[j]))
            else:
                array_xn_zor[i].append(max(array_xd[j]))
    array_xp_zor = [list(array_xp[i][1:]) for i in range(len(array_xp))]
    matr_zor(array_xp_zor, [1, 1, 1], [0, 0, 0])
    matr_zor(array_xn_zor, array_xd_delta, array_x0)
    expanded_matr(array_xp_zor)
    expanded_matr(array_xn_zor)
    for i in array_xp_zor:
        i.insert(0, 1)
    for i in range(len(array_xn_zor)):
        array_y.append([])
        temp = 0
        flag = 0
        for j in range(len(array_xn_zor[i]) + 1):
            if j == 0:
                temp += array_koef_var[j]
            else:
                temp += (array_xn_zor[i][flag] * array_koef_var[j])
                flag += 1
        for k in range(m):
            array_y[i].append(temp + ra.randrange(0, 10) - 5)
    array_aver_y = [sum(array_y[i]) / len(array_y[i]) for i in range(len(array_y))]
    ta_zor = PrettyTable()
    ta_zor.field_names = ["X0", "X1", "X2", "X3", "X1X2", "X1X3", "X2X3", "X1X2X3",
"X1^2", "X2^2", "X3^2",
                          "Y1", "Y2", "Y3", "avg(Y)"]
    for i in range(n):
        ta_zor.add_row(array_xp_zor[i] + array_y[i] + [round(array_aver_y[i], 4)])
    print("Матриця ПЕ для РЦКП із нормованими значеннями")
    print(ta_zor)
    ta1_zor = PrettyTable()
    ta1_zor.field_names = ["X1", "X2", "X3", "X1X2", "X1X3", "X2X3", "X1X2X3",
```

```
"X1^2", "X2^2", "X3^2", "Y1",
                            "Y2", "Y3", "avg(Y)"]
    for i in range(n):
        ta1_zor.add_row(array_xn_zor[i] + array_y[i] + [round(array_aver_y[i], 4)])
    print("\nМатриця ПЕ для РЦКП із натуралізованими значеннями")
    print(ta1_zor)
    average_x = find_average(array_xn_zor)
    res_last = koef_rivn(average_x, sum(array_aver_y) / 15)
    print("Значення коефіцієнтів рівняння регресії")
    print(res_last)
    print("\nРівняння регресії")
    print(output_equation(res_last))
    print("\nПеревірка однорідності дисперсії за критерієм Кохрена")
    if cochrane(m, n, array_y, array_aver_y, np.array(array_xp_zor).transpose(),
res_last, array_xn_zor):
        stoper = input("Якщо ви хочете зупинити програму напишіть \"stop\": ")
        if stoper == "stop":
            return print("Завершуємо програму")
        else:
            print("\nПерезапускаємо програму\n")
            main1(m, n)


def a(first, second):
    global array_xn_zor
    need_a = 0
    for j in range(15):
        need_a += array_xn_zor[j][first - 1] * array_xn_zor[j][second - 1] / 15
    return need_a


def find_known(number):
    global array_xn_zor, array_aver_y
    need_a = 0
    for j in range(15):
        need_a += array_aver_y[j] * array_xn_zor[j][number - 1] / 15
    return need_a


main1(3, 15)
```

Роздруківка результату роботи програми:

Матриця ПЕ для РЦКП із нормованими значеннями

X0	X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1^2	X2^2	X3^2	Y1	Y2	Y3	avg(Y)
1	-1	-1	-1	1	1	1	-1	1	1	1	-15544.8	-15540.8	-15539.8	-15541.8
1	-1	-1	1	1	-1	-1	1	1	1	1	-22878.3	-22886.3	-22881.3	-22881.9667
1	-1	1	-1	-1	1	-1	1	1	1	1	51438.2	51441.2	51436.2	51438.5333
1	-1	1	1	-1	-1	1	-1	1	1	1	75300.7	75297.7	75302.7	75300.3667
1	1	-1	-1	-1	-1	1	1	1	1	1	-63359.8	-63363.8	-63359.8	-63361.1333
1	1	-1	1	-1	1	-1	-1	1	1	1	-98518.3	-98525.3	-98518.3	-98520.6333
1	1	1	-1	1	-1	-1	-1	1	1	1	241383.2	241385.2	241377.2	241381.8667
1	1	1	1	1	1	1	1	1	1	1	349420.7	349420.7	349416.7	349419.3667
1	-1.73	0	0	-0.0	-0.0	0	-0.0	2.9929	0	0	-5676.364000000003	-5673.364000000003	-5677.364000000003	-5675.6973
1	1.73	0	0	0.0	0.0	0	0.0	2.9929	0	0	141634.136	141639.136	141631.136	141634.8027
1	0	-1.73	0	-0.0	0	-0.0	-0.0	0	2.9929	0	-135057.182	-135054.182	-135051.182	-135054.182
1	0	1.73	0	0.0	0	0.0	0.0	0	2.9929	0	261919.538	261917.538	261913.538	261916.8713
1	0	0	-1.73	0	-0.0	-0.0	-0.0	0	0	2.9929	43201.1255	43197.1255	43200.1255	43199.4588
1	0	0	1.73	0	0.0	0.0	0.0	0	0	2.9929	81870.4905	81866.4905	81864.4905	81867.1572
1	0	0	0	0	0	0	0	0	0	0	62472.45	62477.45	62469.45	62473.1167

Матриця ПЕ для РЦКП із натуралізованими значеннями

X1	X2	X3	X1X2	X1X3	X2X3	X1X2X3	X1^2	X2^2	X3^2	Y1	Y2	Y3	avg(Y)
10	-20	10	-200	100	-200	-2000	100	400	100	-15544.8	-15540.8	-15539.8	-15541.8
10	-20	15	-200	150	-300	-3000	100	400	225	-22878.3	-22886.3	-22881.3	-22881.9667
10	60	10	600	100	600	6000	100	3600	100	51438.2	51441.2	51436.2	51438.5333
10	60	15	600	150	900	9000	100	3600	225	75300.7	75297.7	75302.7	75300.3667
50	-20	10	-1000	500	-200	-10000	2500	400	100	-63359.8	-63363.8	-63359.8	-63361.1333
50	-20	15	-1000	750	-300	-15000	2500	400	225	-98518.3	-98525.3	-98518.3	-98520.6333
50	60	10	3000	500	600	30000	2500	3600	100	241383.2	241385.2	241377.2	241381.8667
50	60	15	3000	750	900	45000	2500	3600	225	349420.7	349420.7	349416.7	349419.3667
-4.600000000000001	20.0	12.5	-92.00000000000003	-57.500000000000014	250.0	-1150.0000000000005	21.160000000000014	400.0	156.25	-5676.364000000003	-5673.364000000003	-5677.364000000003	-5675.6973
64.6	20.0	12.5	1292.0	807.4999999999999	250.0	16150.0	4173.159999999999	400.0	156.25	141634.136	141639.136	141631.136	141634.8027
30.0	-49.2	12.5	-1476.0	375.0	-615.0	-18450.0	900.0	2420.6400000000003	156.25	-135057.182	-135054.182	-135051.182	-135054.182
30.0	89.2	12.5	2676.0	375.0	1115.0	33450.0	900.0	7956.64	156.25	261919.538	261917.538	261913.538	261916.8713
30.0	20.0	8.175	600.0	245.25000000000003	163.5	4905.0	900.0	400.0	66.83062500000001	43201.1255	43197.1255	43200.1255	43199.4588
30.0	20.0	16.825	600.0	504.75	336.5	10095.0	900.0	400.0	283.080625	81870.4905	81866.4905	81864.4905	81867.1572
30.0	20.0	12.5	600.0	375.0	250.0	7500.0	900.0	400.0	156.25	62472.45	62477.45	62469.45	62473.1167

Значення коефіцієнтів рівняння регресії
[12.60403217 5.55030207 6.16376485 1.91688785 4.30395833 0.89708333
  8.008125   6.9996875   4.60011319 0.20023712 3.25179398]

Рівняння регресії
y = 12.604 + 5.5503 * x1 + 6.1638 * x2 + 1.9169 * x3 + 4.304 * x4 + 0.8971 * x5 + 8.0081 * x6 + 6.9997 * x7 + 4.6001 * x8 + 0.2002 * x9 + 3.2518 * x10

Перевірка однорідності дисперсії за критерієм Кохрена
F1 = 2
F2 = 15
q = 0.05
Значення дисперсій по рядках
[4.6667, 10.8889, 4.2222, 4.2222, 3.5556, 10.8889, 11.5556, 3.5556, 2.8889, 10.8889, 6.0, 6.2222, 2.8889, 6.2222, 10.8889]

Gp = 0.11607170659238998

Gp = 0.11607170659238998 < Gt = 0.7679
Дисперсія однорідна

Оцінимо значимість коефіцієнтів регресії згідно критерію Стьюдента
m = 3
N = 15
Отримані значення βi
[64506.40848888889, 39696.76655555556, 106973.70592888888, 10419.652318888888, 39168.02222222223, 3757.0888888888876, 11626.6, 7466.333333333333, 61609.773756808885, 59794.796193715556, 59436.4316684266 74]
Отримані значення ti
[167966.11748973935, 103365.10606355804, 278545.31788821454, 27131.390300518528, 101988.32606766046, 9782.960306016801, 30274.12170904822, 19441.340035234465, 160423.66548952894, 155697.70505016754, 154764.57143803668]

f3 = 30
q = 0.05

t0 = 167966.11748973935 > tтабл = 2.042

t1 = 103365.10606355804 > tтабл = 2.042

t2 = 278545.31788821454 > tтабл = 2.042

t3 = 27131.390300518528 > tтабл = 2.042

t4 = 101988.32606766046 > tтабл = 2.042

t5 = 9782.960306016801 > tтабл = 2.042

t6 = 30274.12170904822 > tтабл = 2.042

```
t7 = 19441.340035234465 > ttабл = 2.042

t8 = 160423.66548952894 > ttабл = 2.042

t9 = 155697.70505016754 > ttабл = 2.042

t10 = 154764.57143803668 > ttабл = 2.042

Перепишемо рівняння враховуючи вилучених коефіцієнтів
y = 12.604 + 5.5503 * x1 + 6.1638 * x2 + 1.9169 * x3 + 4.304 * x4 + 0.8971 * x5 + 8.0081 * x6 + 6.9997 * x7 + 4.6001 * x8 + 0.2002 * x9 + 3.2518 * x10

Рівняння з використанням незначимих коефіцієнтів
y =

Підставимо необхідні значення X
y0 = -15542.797132271235
y1 = -22882.38427899289
y2 = 51438.22952214933
y3 = 75300.64237541953
y4 = -63362.34672900243
y5 = -98521.26720906179
y6 = 241381.34659207892
y7 = 349419.4261120222
y8 = -5675.32074124562
y9 = 141635.67928973856
y10 = -135052.75386515612
y11 = 261916.69641365067
y12 = 43200.755406165634
y13 = 81867.1138089977
y14 = 62473.10776883903

Критерій Фішера
d =  11
f3 =  30
f4 =  4
q = 0.05

Fp = 0.9208136244796061 < Ft = 2.6896275736914177
Рівняння регресії адекватно оригіналу

Process finished with exit code 0
```

Висновок:

В ході лабораторної роботи було проведено трьохфакторний експеримент і отримано адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план. Під час виконання лабораторної роботи проблем не виникло, що підтверджують дані наведені вище.