Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**ЛАБОРАТОРНА РОБОТА № 5**

з дисципліни «Методи оптимізації та планування»

Тема: ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З УРАХУВАННЯМ КВАДРАТИЧНИХ ЧЛЕНІВ (ЦЕНТРАЛЬНИЙ ОРТОГОНАЛЬНИЙ КОМПОЗИЦІЙНИЙ ПЛАН)

ВИКОНАВ:

студент ІІ курсу ФІОТ

групи ІВ-83

Литвиненко Олександи

Варіант-317

ПЕРЕВІРИВ:

Регіда П.Г.

Київ – 2020

**Лабораторна робота №5**

**Тема:** Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів(центральний ортогональний композиційний план).

**Мета:** Провести трьохфакторний експеримент з урахуванням квадратичних членів, використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Введемо такі позначення:

N – кількість точок плану (рядків матриці планування)

k – кількість факторів(кількість x)

m – кількість дослідів y за однієї і тієї ж комбінації факторів (test)

s

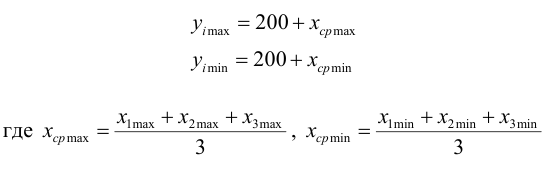
x - нормовані значення факторів(s =1,k)

**Завдання на лабораторну роботу**

1. Взяти рівняння з урахуванням квадратичних членів.

2. Скласти матрицю планування для ОЦКП.

3. Провести експеримент у всіх точках факторного простору (знайти значення функції відгуку Y). Значення функції відгуку знайти у відповідності з варіантом діапазону, азначеного далі. Варіанти вибираються по номеру в списку в журналі викладача.



4. Розрахувати коефіцієнти рівняння регресії і записати його.

5. Провести 3 статистичні перевірки.

**Код програми:**

from numpy.linalg import solve  
from math import fabs as fab  
from math import sqrt  
from \_pydecimal import Decimal  
from scipy.stats import f, t  
  
  
class Critical\_values:  
 @staticmethod  
 def get\_cohren\_value(size\_of\_selections, qty\_of\_selections, significance):  
 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\_\_()  
  
 @staticmethod  
 def get\_student\_value(f3, significance):  
 return Decimal(abs(t.ppf(significance / 2, f3))).quantize(Decimal('.0001')).\_\_float\_\_()  
  
 @staticmethod  
 def get\_fisher\_value(f3, f4, significance):  
 return Decimal(abs(f.isf(significance, f4, f3))).quantize(Decimal('.0001')).\_\_float\_\_()  
  
  
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 generate\_y\_matrix():  
 from random import randrange  
 matrix\_with\_y = [[randrange(y\_min, y\_max) for y in range(m)] for x in range(N)]  
 return matrix\_with\_y  
  
  
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\_right(number):  
 need\_a = 0  
 for j in range(N):  
 need\_a += average\_y[j] \* matrix\_x[j][number - 1] / 15  
 return need\_a  
  
  
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):  
 t\_practice = 0  
 t\_theoretical = Critical\_values.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 fab(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 = Critical\_values.get\_fisher\_value(f3, f4, q)  
 return F\_practice < F\_theoretical  
  
  
m, d = 3, 0  
N = 15  
p = 0.95  
  
x1\_min, x1\_max = -7, -8  
x2\_min, x2\_max = -7, 9  
x3\_min, x3\_max = -4, 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  
y\_min = 200 + int((x1\_min + x2\_min + x3\_min) / 3)  
y\_max = 200 + int((x1\_max + x2\_max + x3\_max) / 3)  
  
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],  
 [-1.215, 0, 0, 0, 0, 0, 0, 1.4623, 0, 0],  
 [+1.215, 0, 0, 0, 0, 0, 0, 1.4623, 0, 0],  
 [0, -1.215, 0, 0, 0, 0, 0, 0, 1.4623, 0],  
 [0, +1.215, 0, 0, 0, 0, 0, 0, 1.4623, 0],  
 [0, 0, -1.215, 0, 0, 0, 0, 0, 0, 1.4623],  
 [0, 0, +1.215, 0, 0, 0, 0, 0, 0, 1.4623],  
 [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:  
 x1 = x1\_min if matrix\_pfe[i][0] == -1 else x1\_max  
 x2 = x2\_min if matrix\_pfe[i][1] == -1 else x2\_max  
 x3 = 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])  
 x1, x2, x3 = x\_lst  
 matrix\_x[i] = [x1, x2, x3, x1 \* x2, x1 \* x3, x2 \* x3, x1 \* x2 \* x3, x1 \*\* 2, x2 \*\* 2, x3 \*\* 2]  
  
matrix\_y = generate\_y\_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  
  
left = [  
 [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)]  
]  
right = [my, find\_right(1), find\_right(2), find\_right(3), find\_right(4), find\_right(5), find\_right(6), find\_right(7),  
 find\_right(8), find\_right(9), find\_right(10)]  
  
beta = solve(left, right)  
  
print("(ಠ‿ಠ)﻿"\*8)  
print("—" \* 50)  
  
print("Рівняння регресії з урахуванням квадратичних членів:")  
print("{:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"  
 "+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11^2 + {:.3f} \* X22^2 + {:.3f} \* X33^2 = ŷ\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("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check\_result(beta, i), average\_y[i]))  
  
cohren = False  
while not cohren:  
 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("—" \* 50)  
  
 print("Критерій Кохрена:")  
 Gt = Critical\_values.get\_cohren\_value(f2, f1, q)  
 if Gt > Gp or m >= 25:  
 print("Дисперсія однорідна при рівні значимості {:.2f}! Збільшувати m не потрібно.".format(q))  
 print("—" \* 50)  
 cohren = True  
 else:  
 print("Дисперсія не однорідна при рівні значимості {:.2f}!".format(q))  
 print("—" \* 50)  
 m += 1  
 if m == 25:  
 exit()  
  
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} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"  
 "+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11^2 + {:.3f} \* X22^2 + {:.3f} \* X33^2 = ŷ\nПеревірка:"  
 .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("—" \* 50)  
  
print("Критерій Фішера:")  
d = 11 - student\_lst.count(0)  
if fisher\_test():  
 print("Рівняння регресії адекватне стосовно оригіналу.")  
else:  
 print("Рівняння регресії неадекватне стосовно оригіналу.")  
print("—" \* 50)  
print("(◕‿◕)♡"\*7)

**Результат:**

——————————————————————————————————

Рівняння регресії з урахуванням квадратичних членів:

237.052 + 12.561 \* X1 + 0.648 \* X2 + -0.821 \* X3 + 0.071 \* Х1X2 + -0.092 \* Х1X3 + -0.039 \* Х2X3+ -0.003 \* Х1Х2X3 + 0.956 \* X11^2 + -0.003 \* X22^2 + 0.037 \* X33^2 = ŷ

Перевірка:

ŷ1 = 195.590 ≈ 195.667

ŷ2 = 198.002 ≈ 197.667

ŷ3 = 198.994 ≈ 199.000

ŷ4 = 197.406 ≈ 197.000

ŷ5 = 197.590 ≈ 198.000

ŷ6 = 201.002 ≈ 201.000

ŷ7 = 199.661 ≈ 200.000

ŷ8 = 199.740 ≈ 199.667

ŷ9 = 195.790 ≈ 196.333

ŷ10 = 198.220 ≈ 197.667

ŷ11 = 195.789 ≈ 195.667

ŷ12 = 196.888 ≈ 197.000

ŷ13 = 198.683 ≈ 198.000

ŷ14 = 199.993 ≈ 200.667

ŷ15 = 196.652 ≈ 196.667

——————————————————————————————————

Критерій Кохрена:

Дисперсія однорідна при рівні значимості 0.05! Збільшувати m не потрібно.

——————————————————————————————————

Отримане рівняння регресії з урахуванням критерія Стьюдента:

237.052 + 12.561 \* X1 + 0.648 \* X2 + -0.821 \* X3 + 0.071 \* Х1X2 + -0.092 \* Х1X3 + -0.039 \* Х2X3+ 0.000 \* Х1Х2X3 + 0.956 \* X11^2 + -0.003 \* X22^2 + 0.037 \* X33^2 = ŷ

Перевірка:

ŷ1 = 195.007 ≈ 195.667

ŷ2 = 199.460 ≈ 197.667

ŷ3 = 199.744 ≈ 199.000

ŷ4 = 195.531 ≈ 197.000

ŷ5 = 196.923 ≈ 198.000

ŷ6 = 202.669 ≈ 201.000

ŷ7 = 200.518 ≈ 200.000

ŷ8 = 197.597 ≈ 199.667

ŷ9 = 195.728 ≈ 196.333

ŷ10 = 198.148 ≈ 197.667

ŷ11 = 196.373 ≈ 195.667

ŷ12 = 196.170 ≈ 197.000

ŷ13 = 198.806 ≈ 198.000

ŷ14 = 199.737 ≈ 200.667

ŷ15 = 196.585 ≈ 196.667

——————————————————————————————————

Критерій Фішера:

Рівняння регресії неадекватне стосовно оригіналу.

——————————————————————————————————

Process finished with exit code 0

**Висновок:** У ході виконання лабораторної роботи ми провели трьохфакторний експеримент при використанні рівняння з урахуванням квадратичних членів. Склали матрицю планування, знайшли коефіцієнти рівняння регресії, провели статистичні перевірки. Була написана текстова програма, результати наведені вище. Результати співпадають із калькулятором. Кінцева мета роботи досягнута!