НаціональнийтехнічнийуніверситетУкраїни

«КиївськийполітехнічнийінститутіменіІгоряСікорського»

Факультет інформатики та обчислювальноїтехніки

Кафедра обчислювальноїтехніки

**Методи оптимізації та планування експерименту**

Лабораторна робота №6:

«ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З КВАДРАТИЧНИМИ ЧЛЕНФМИ(РОТОТАБЕЛЬНИЙ КОМПОЗИЦІЙНИЙ ПЛАН)»

Виконав:

студент групи ІО-83

Бурлака Давид Сергійович

Залікова книжка № 8310

Перевірив Регіда П. Г.

Київ 2020р.

**Лабораторна робота №6**

**Тема:**ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ ПРИ ВИКОРИСТАННІ РІВНЯННЯ РЕГРЕСІЇ З КВАДРАТИЧНИМИЧЛЕНАМИ(РОТОТАБЕЛЬНИЙ КОМПОЗИЦІЙНИЙ ПЛАН).

**Мета:**провести трьохфакторний експеремент з урахуванням квадратичних членів, використовуючи рототабельний композиційний план. Знайти рівняння регресії адекватнее об'єкту.

**Виконання:**

Варіант – 304.

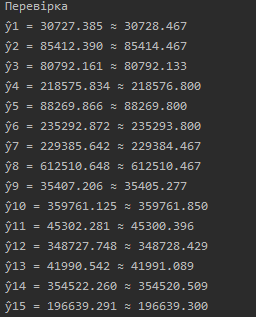


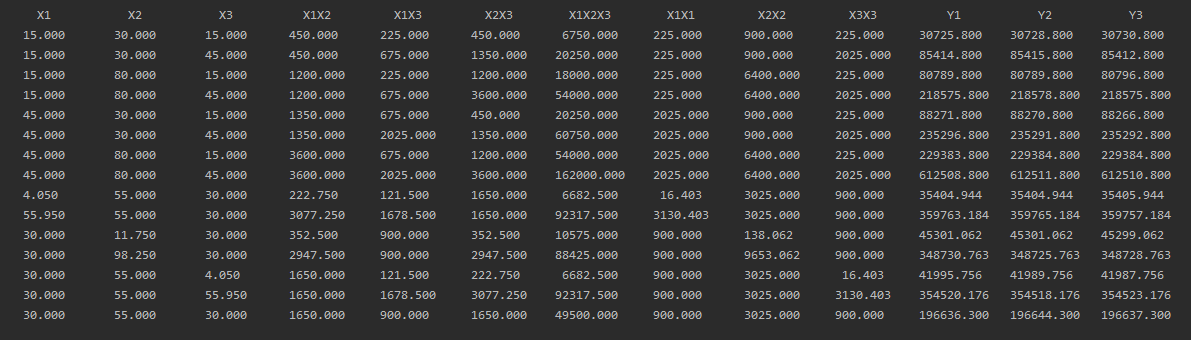
1. Лістинг програми:

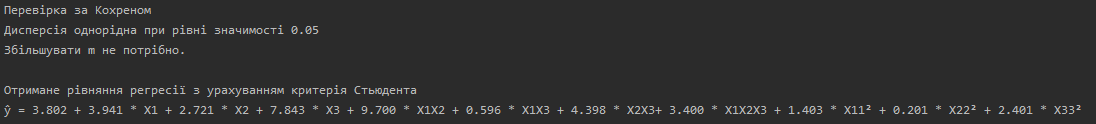
from numpy.linalg import solve  
from \_pydecimal import Decimal  
from scipy.stats import f  
from scipy.stats import t  
from random import randrange  
from math import sqrt  
from math import fabs as fab  
  
m, d = 3, 0  
p = 0.95  
N = 15  
  
x1\_min, x1\_max = 15, 45  
x2\_min, x2\_max = 30, 80  
x3\_min, x3\_max = 15, 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  
  
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]  
]  
  
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 generate\_matrix():  
 def f(X1, X2, X3):  
 y = 0.3 + 4.1 \* X1 + 2.8 \* X2 + 7.8 \* X3 + 1.4 \* X1 \* X1 + 0.2 \* X2 \* X2 + 2.4 \* X3 \* X3 + 9.7 \* X1 \* X2 + \  
 0.6 \* X1 \* X3 + 4.4 \* X2 \* X3 + 3.4 \* X1 \* X2 \* X3 + randrange(0, 10) - 5  
 return y  
  
 matrix\_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\_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: # Середнє значення по рядку  
 for rows in range(len(lst)):  
 average.append(sum(lst[rows]) / len(lst[rows]))  
 else: # Середнє значення по колонкі  
 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): # first = 1, second = 2 : пошук а12  
 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\_a(number):  
 a = 0  
 for j in range(N):  
 a += average\_y[j] \* matrix\_x[j][number - 1] / 15  
 return 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(b\_lst, number\_x=10):  
 disp\_b = sqrt(disp\_b2)  
 for column in range(number\_x + 1):  
 tp = 0  
 tt = Critical\_values.get\_student\_value(f3, q)  
 for row in range(N):  
 if column == 0:  
 tp += average\_y[row] / N  
 else:  
 tp += average\_y[row] \* matrix\_pfe[row][column - 1]  
 if fab(tp / disp\_b) < tt:  
 b\_lst[column] = 0  
 return b\_lst  
  
  
def fisher():  
 disp\_ad = 0  
 f4 = N - d  
 for row in range(len(average\_y)):  
 disp\_ad += (m \* (average\_y[row] - check\_result(student\_lst, row))) / (N - d)  
 Fp = disp\_ad / disp\_b2  
 Ft = Critical\_values.get\_fisher\_value(f3, f4, q)  
 return Fp < Ft  
  
  
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]  
  
adequacy, homogeneity = False, False  
while not adequacy:  
 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\_a(1), find\_known\_a(2), find\_known\_a(3), find\_known\_a(4), find\_known\_a(5), find\_known\_a(6),  
 find\_known\_a(7), find\_known\_a(8), find\_known\_a(9), find\_known\_a(10)]  
  
 beta = solve(unknown, known)  
 print("Отримане рівняння регресії")  
 print("ŷ = {:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"  
 "+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11² + {:.3f} \* X22² + {:.3f} \* X33²\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]))  
  
 while not homogeneity:  
 print("\n{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}{:^13}"  
 .format("X1", "X2", "X3", "X1X2", "X1X3", "X2X3", "X1X2X3", "X1X1", "X2X2", "X3X3", "Y1", "Y2", "Y3"))  
 for row in range(N):  
 for column in range(len(matrix[0])):  
 print("{:^12.3f}".format(matrix[row][column]), end=' ')  
 print()  
 disp\_y = [0.0 for x in range(N)]  
 for i in range(N):  
 disp\_i = 0  
 for j in range(m):  
 disp\_i += (matrix\_y[i][j] - average\_y[i]) \*\* 2  
 disp\_y.append(disp\_i / (m - 1))  
 f1 = m - 1  
 f2 = N  
 f3 = f1 \* f2  
 q = 1 - p  
 Gp = max(disp\_y) / sum(disp\_y)  
 print("\nПеревірка за Кохреном")  
 Gt = Critical\_values.get\_cohren\_value(f2, f1, q)  
 if Gt > Gp:  
 print("Дисперсія однорідна при рівні значимості {:.2f}\nЗбільшувати m не потрібно.".format(q))  
 homogeneity = True  
 else:  
 print("Дисперсія не однорідна при рівні значимості {:.2f}".format(q))  
 m += 1  
  
 disp\_b2 = sum(disp\_y) / (N \* N \* m)  
 student\_lst = list(student(beta))  
 print("\nОтримане рівняння регресії з урахуванням критерія Стьюдента")  
 print("ŷ = {:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"  
 "+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11² + {:.3f} \* X22² + {:.3f} \* X33²\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("\nПеревірка за Фішером")  
 d = 11 - student\_lst.count(0)  
 if fisher():  
 print("Рівняння регресії адекватне оригіналу")  
 adequacy = True  
 else:  
 print("Рівняння регресії неадекватне оригіналу")

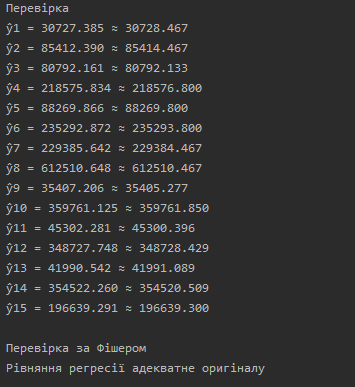
1. Результатвиконанняроботи програми:

****

****

****

****

****

**Висновок:**Отже, у ході виконання лабораторної роботи № 6 провели трьохфакторний експеримент при використанні рівняння з урахуванням квадратичних членів. Склали матрицю планування, знайшли коефіцієнти рівняння регресії, провели 3 статистичні перевірки.Була написана текстова програма, результати наведені вище. Результати співпадають із калькулятором. Кінцева мета роботи досягнута!