Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №5**

з дисципліни «Методи оптимізації та планування експерименту»

на тему: «Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план).»

Виконав**:**

студент 2-го курсу ФІОТ

групи ІО-81

Микита Бєлов

Варіант: 102

Перевірив:  
Регіда П.Г.

Київ – 2020

Завдання





Код

from \_pydecimal import Decimal  
from scipy.stats import f, t  
  
m = 0  
d = 0  
N = 15  
  
x1\_min = -8  
x1\_max = 9  
x2\_min = -6  
x2\_max = 2  
x3\_min = -1  
x3\_max = 5  
  
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)  
  
def sqrt(element):  
 from math import sqrt  
 return sqrt ( element )  
  
  
def fab(element):  
 from math import fabs  
 return fabs ( element )  
  
correct\_input = False  
while not correct\_input: #making sure the input is correct  
 try:  
 m = int(input("Enter number of reiteration: "))  
 p = float(input("Enter confidence probability: "))  
 correct\_input = True  
 except ValueError:  
 pass  
  
#starts cycle where the main code performs  
correct = False  
while not correct:  
 try:  
 array = [#initial 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, +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]  
 ]  
  
  
  
 class Critical\_values: #class which contains static method to calculate criterios 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\_matrix(): #matrix with Y  
 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 count(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(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):  
 b\_dispersion = 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] \* array[row ][ column - 1 ]  
 if (t\_practice / b\_dispersion) < 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  
  
  
 x\_matrix = [[] for x in range(N)]  
 for i in range(len(x\_matrix)):  
 if i < 8:  
 x1 = x1\_min if array[i ][0 ] == -1 else x1\_max  
 x2 = x2\_min if array[i ][1 ] == -1 else x2\_max  
 x3 = x3\_min if array[i ][2 ] == -1 else x3\_max  
 else:  
 x\_lst = x( array[i ][0 ] , array[i ][1 ] , array[i ][2 ] )  
 x1, x2, x3 = x\_lst  
 x\_matrix[i] = [x1, x2, x3, x1 \* x2, x1 \* x3, x2 \* x3, x1 \* x2 \* x3, x1 \*\* 2, x2 \*\* 2, x3 \*\* 2]  
  
 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  
  
 values\_arr = [  
 [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], count( 1 , 1 ), count( 1 , 2 ), count( 1 , 3 ), count( 1 , 4 ), count( 1 , 5 ), count( 1 , 6 ), count( 1 , 7 ), count( 1 , 8 ), count( 1 , 9 ), count( 1 , 10 ) ],  
 [ mx\_i[1], count( 2 , 1 ), count( 2 , 2 ), count( 2 , 3 ), count( 2 , 4 ), count( 2 , 5 ), count( 2 , 6 ), count( 2 , 7 ), count( 2 , 8 ), count( 2 , 9 ), count( 2 , 10 ) ],  
 [ mx\_i[2], count( 3 , 1 ), count( 3 , 2 ), count( 3 , 3 ), count( 3 , 4 ), count( 3 , 5 ), count( 3 , 6 ), count( 3 , 7 ), count( 3 , 8 ), count( 3 , 9 ), count( 3 , 10 ) ],  
 [ mx\_i[3], count( 4 , 1 ), count( 4 , 2 ), count( 4 , 3 ), count( 4 , 4 ), count( 4 , 5 ), count( 4 , 6 ), count( 4 , 7 ), count( 4 , 8 ), count( 4 , 9 ), count( 4 , 10 ) ],  
 [ mx\_i[4], count( 5 , 1 ), count( 5 , 2 ), count( 5 , 3 ), count( 5 , 4 ), count( 5 , 5 ), count( 5 , 6 ), count( 5 , 7 ), count( 5 , 8 ), count( 5 , 9 ), count( 5 , 10 ) ],  
 [ mx\_i[5], count( 6 , 1 ), count( 6 , 2 ), count( 6 , 3 ), count( 6 , 4 ), count( 6 , 5 ), count( 6 , 6 ), count( 6 , 7 ), count( 6 , 8 ), count( 6 , 9 ), count( 6 , 10 ) ],  
 [ mx\_i[6], count( 7 , 1 ), count( 7 , 2 ), count( 7 , 3 ), count( 7 , 4 ), count( 7 , 5 ), count( 7 , 6 ), count( 7 , 7 ), count( 7 , 8 ), count( 7 , 9 ), count( 7 , 10 ) ],  
 [ mx\_i[7], count( 8 , 1 ), count( 8 , 2 ), count( 8 , 3 ), count( 8 , 4 ), count( 8 , 5 ), count( 8 , 6 ), count( 8 , 7 ), count( 8 , 8 ), count( 8 , 9 ), count( 8 , 10 ) ],  
 [ mx\_i[8], count( 9 , 1 ), count( 9 , 2 ), count( 9 , 3 ), count( 9 , 4 ), count( 9 , 5 ), count( 9 , 6 ), count( 9 , 7 ), count( 9 , 8 ), count( 9 , 9 ), count( 9 , 10 ) ],  
 [ mx\_i[9], count( 10 , 1 ), count( 10 , 2 ), count( 10 , 3 ), count( 10 , 4 ), count( 10 , 5 ), count( 10 , 6 ), count( 10 , 7 ), count( 10 , 8 ), count( 10 , 9 ), count( 10 , 10 ) ]  
 ]  
 outputArray = [ 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( values\_arr , outputArray )  
 print("\tRegression equation =>")  
 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\n\t check"  
 .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]))  
  
 y\_dispersion = [0.0 for x in range(N)]  
 homogeneity = False  
 while not homogeneity:  
 y\_dispersion = [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  
 y\_dispersion.append(dispersion\_i / (m - 1))  
  
 f1 = m - 1  
 f2 = N  
 f3 = f1 \* f2  
 q = 1 - p  
  
 Gp = max(y\_dispersion) / sum(y\_dispersion)  
  
 print("\n")  
 print("\tKohrens critetion")  
 Gt = Critical\_values.get\_cohren\_value(f2, f1, q)  
 if Gt > Gp or m >= 25: #because 25 is max number of reiterations  
 print("\t\tDispersion is homogeneous {:.2f}!".format(q))  
 homogeneity = True  
 else:  
 print("\t\tDispersion is not homogeneous {:.2f}!".format(q))  
 m += 1 #new Y column will be added  
 dispersion\_b2 = sum(y\_dispersion) / (N \* N \* m)  
 student\_lst = list(student\_test(beta))  
 print("\tRegression equation after Students criterion")  
 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\n\tcheck"  
 .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("\n")  
 print("\tFishers criterion =>")  
  
 d = 11 - student\_lst.count(0)  
 if fisher\_test():  
 correct = True  
 print("\t\tRegression equation is adequate")#end code  
 else:  
 print("\t\tRegression equation is not adequate")#cycle starts with new Y array until got adequate regresssion  
 except ValueError:  
 pass

Приклад роботи програми

