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 № 6

з дисципліни «Методи оптимізації та планування експерименту»

Виконав:

студент ІІ курсу ФІОТ

групи ІО-82

Кузенний П.В.

Залікова книжка № 8209

Варіант: 208

Перевірив:

ст. вик.

Регіда П. Г.

Київ – 2020

Код програми:

import numpy as np  
from prettytable import PrettyTable  
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  
  
  
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(number):  
 a = 0  
 for j in range(N):  
 a += average\_y[j] \* matrix\_x[j][number - 1] / 15  
 return a  
  
  
def check(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  
  
  
# Variant №208  
  
m, d = 3, 0  
p = 0.95  
N = 15  
  
x1\_min, x1\_max = -30, 0  
x2\_min, x2\_max = 10, 60  
x3\_min, x3\_max = 10, 35  
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]]  
  
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 = (  
 matrix\_pfe[i][0] \* delta\_x1 + x01, matrix\_pfe[i][1] \* delta\_x2 + x02, matrix\_pfe[i][2] \* delta\_x3 + x03)  
 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 = [[(8.0 + 5.3 \* matrix\_x[j][0] + 0.5 \* matrix\_x[j][1] + 5.6 \* matrix\_x[j][2] + 3.2 \* matrix\_x[j][0] \*  
 matrix\_x[j][0] + 0.7 \* matrix\_x[j][1] \* matrix\_x[j][1] + 4.1 \* matrix\_x[j][2] \* matrix\_x[j][2] + 8.9 \*  
 matrix\_x[j][0] \* matrix\_x[j][1] + 0.5 \* matrix\_x[j][0] \* matrix\_x[j][2] + 1.5 \* matrix\_x[j][1] \*  
 matrix\_x[j][2] + 1.2 \* matrix\_x[j][0] \* matrix\_x[j][1] \* matrix\_x[j][2] + randrange(0, 10) - 5) for i  
 in range(m)] for j in range(N)]  
 average\_x = []  
 for column in range(len(matrix\_x[0])):  
 number\_lst = []  
 for rows in range(len(matrix\_x)):  
 number\_lst.append(matrix\_x[rows][column])  
 average\_x.append(sum(number\_lst) / len(number\_lst))  
  
 average\_y = []  
 for rows in range(len(matrix\_y)):  
 average\_y.append(sum(matrix\_y[rows]) / len(matrix\_y[rows]))  
  
 matrix = [(matrix\_x[i] + matrix\_y[i]) for i in range(N)]  
 mx\_i = average\_x  
 my = sum(average\_y) / 15  
  
 unk = [  
 [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)]]  
  
 k = [my, find(1), find(2), find(3), find(4), find(5), find(6),  
 find(7), find(8), find(9), find(10)]  
  
 beta = np.linalg.solve(unk, k)  
 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(beta, i), average\_y[i]))  
  
 while not homogeneity:  
 dispersion\_y = [0.0 for x in range(N)]  
 print("\n\n\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("Матриця планування:")  
  
 x\_norm = np.array(matrix\_x)  
 matrix\_plan = np.array(matrix\_y)  
 my\_table = np.hstack((x\_norm, matrix\_plan))  
 table = PrettyTable()  
 table.field\_names = ["X1", "X2", "X3", "X1X2", "X1X3", "X2X3", "X1X2X3", "X1^2", "X2^2", "X3^2", "Y1", "Y2", "Y3"]  
 for i in range(len(my\_table)):  
 table.add\_row(my\_table[i])  
  
 print(table)  
  
 print("Критерій Кохрена")  
  
 f2 += 1  
 partResult1 = q / (f2 - 1)  
 params = [partResult1, f1, (f2 - 1 - 1) \* f1]  
 fisher = f.isf(\*params)  
 result = fisher / (fisher + (f2 - 1 - 1))  
 Gt = Decimal(result).quantize(Decimal('.0001')).\_\_float\_\_()  
 if Gt > Gp:  
 print("Дисперсія однорідна при q = {:.2f}\n".format(q))  
 homogeneity = True  
 else:  
 print("Дисперсія не однорідна при q = {:.2f}\n".format(q))  
 m += 1  
  
 dispersion\_b2 = sum(dispersion\_y) / (N \* N \* m)  
  
 dispersion\_b = sqrt(dispersion\_b2)  
 number\_x = 10  
 for column in range(number\_x + 1):  
 t\_practice = 0  
 t\_theoretical = Decimal(abs(t.ppf(q / 2, f3))).quantize(Decimal('.0001')).\_\_float\_\_()  
 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:  
 beta[column] = 0  
  
 student\_lst = list(beta)  
 print("Отримане рівняння регресії з урахуванням критерія Стьюдента")  
 print("y = {:.3f} + {:.3f} \* x1 + {:.3f} \* x2 + {:.3f} \* 3 + {:.3f} \* x1x2 + {:.3f} \* x1x3 + {:.3f} \* x2x3"  
 "+ {:.3f} \* x1x2x3 + {:.3f} \* x11^2 + {:.3f} \* x22^2 + {:.3f} \* x33^2 \n\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("y{} = {:.3f} ≈ {:.3f}".format((i + 1), check(student\_lst, i), average\_y[i]))  
  
 print("\nКритерій Фішера")  
 d = 11 - student\_lst.count(0)  
 dispersion\_ad = 0  
 f4 = N - d  
 for row in range(len(average\_y)):  
 dispersion\_ad += (m \* (average\_y[row] - check(student\_lst, row))) / (N - d)  
 F\_practice = dispersion\_ad / dispersion\_b2  
 F\_theoretical = Decimal(abs(f.isf(q, f4, f3))).quantize(Decimal('.0001')).\_\_float\_\_()  
 if F\_practice < F\_theoretical:  
 print("Рівняння регресії адекватне стосовно оригіналу")  
 adequacy = True  
 else:  
 print("Рівняння регресії неадекватне стосовно оригіналу. Потрібно провести експеремент повторно!")