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 №4

“ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ РІВНЯННЯ РЕГРЕСІЇ З УРАХУВАННЯМ ЕФЕКТУ ВЗАЄМОДІЇ.”

Виконав:

студент групи ІВ-83

Щебетін Б. Ю.

Перевірив:

ас. Регіда П.Г.

Київ

2020 р.

**Мета:** провести повний трьохфакторний експеримент, знайти коефіцієнти рівняння, перевірити однорідність дисперсії, знайти незначущі коефіцієнти.

Номер у списку: 22

Варіант завдання: 322





Код програми

from copy import deepcopy  
from math import sqrt  
import numpy as np  
from prettytable import PrettyTable  
  
x1\_min = -30  
x1\_max = 20  
x2\_min = -20  
x2\_max = 40  
x3\_min = -30  
x3\_max = -15  
  
x\_average\_max = (x1\_max + x2\_max + x3\_max) / 3  
x\_average\_min = (x1\_min + x2\_min + x3\_min) / 3  
y\_max = 200 + x\_average\_max  
y\_min = 200 + x\_average\_min  
  
FishersTable = [  
 [164.4, 199.5, 215.7, 224.6, 230.2, 234, 235.8, 237.6],  
 [18.5, 19.2, 19.2, 19.3, 19.3, 19.3, 19.4, 19.4],  
 [10.1, 9.6, 9.3, 9.1, 9, 8.9, 8.8, 8.8],  
 [7.7, 6.9, 6.6, 6.4, 6.3, 6.2, 6.1, 6.1],  
 [6.6, 5.8, 5.4, 5.2, 5.1, 5, 4.9, 4.9],  
 [6, 5.1, 4.8, 4.5, 4.4, 4.3, 4.2, 4.2],  
 [5.5, 4.7, 4.4, 4.1, 4, 3.9, 3.8, 3.8],  
 [5.3, 4.5, 4.1, 3.8, 3.7, 3.6, 3.5, 3.5],  
 [5.1, 4.3, 3.9, 3.6, 3.5, 3.4, 3.3, 3.3],  
 [5, 4.1, 3.7, 3.5, 3.3, 3.2, 3.1, 3.1],  
 [4.8, 4, 3.6, 3.4, 3.2, 3.1, 3, 3],  
 [4.8, 3.9, 3.5, 3.3, 3.1, 3, 2.9, 2.9],  
 [4.7, 3.8, 3.4, 3.2, 3, 2.9, 2.8, 2.8],  
 [4.6, 3.7, 3.3, 3.1, 3, 2.9, 2.8, 2.7],  
 [4.5, 3.7, 3.3, 3.1, 2.9, 2.8, 2.7, 2.7],  
 [4.5, 3.6, 3.2, 3, 2.9, 2.7, 2.6, 2.6],  
 [4.5, 3.6, 3.2, 3, 2.8, 2.7, 2.5, 2.3],  
 [4.4, 3.6, 3.2, 2.9, 2.8, 2.7, 2.5, 2.3],  
 [4.4, 3.5, 3.1, 2.9, 2.7, 2.7, 2.4, 2.3],  
 [4.4, 3.5, 3.1, 2.8, 2.7, 2.7, 2.4, 2.3],  
 [4.4, 3.5, 3.1, 2.8, 2.7, 2.6, 2.4, 2.3],  
 [4.3, 3.4, 3.1, 2.8, 2.7, 2.6, 2.4, 2.3],  
 [4.3, 3.4, 3.1, 2.8, 2.6, 2.6, 2.3, 2.2],  
 [4.3, 3.4, 3, 2.8, 2.6, 2.5, 2.3, 2.2],  
 [4.3, 3.4, 3, 2.8, 2.6, 2.5, 2.3, 2.2],  
 [4.2, 3.4, 3, 2.7, 2.6, 2.5, 2.3, 2.2],  
 [4.2, 3.4, 3, 2.7, 2.6, 2.5, 2.3, 2.2],  
 [4.2, 3.3, 3, 2.7, 2.6, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 3, 2.7, 2.6, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.2, 3.3, 2.9, 2.7, 2.5, 2.4, 2.3, 2.1],  
 [4.1, 3.2, 2.9, 2.6, 2.5, 2.3, 2.3, 2],  
  
  
 ]  
  
def replace\_column(list\_: list, column, list\_replace):  
 list\_ = deepcopy(list\_)  
 for i in range(len(list\_)):  
 list\_[i][column] = list\_replace[i]  
 return list\_  
  
  
def main(m, n):  
 if n == 8:  
 print(  
 'ŷ = b0 + b1 \* x1 + b2 \* x2 + b3 \* x3 + b12 \* x1 \* x2 + b13 \* x1 \* x3 + b23 \* x2 \* x3 + b123 \* x1 \* x2 \* x3')  
 norm\_x = [  
 [+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]  
 ]  
  
 for i in range(len(norm\_x)):  
 norm\_x[i].append(norm\_x[i][1] \* norm\_x[i][2])  
 norm\_x[i].append(norm\_x[i][1] \* norm\_x[i][3])  
 norm\_x[i].append(norm\_x[i][2] \* norm\_x[i][3])  
 norm\_x[i].append(norm\_x[i][1] \* norm\_x[i][2] \* norm\_x[i][3])  
  
 x = [  
 [x1\_min, x2\_min, x3\_min],  
 [x1\_min, x2\_max, x3\_max],  
 [x1\_max, x2\_min, x3\_max],  
 [x1\_max, x2\_max, x3\_min],  
 [x1\_min, x2\_min, x3\_max],  
 [x1\_min, x2\_max, x3\_min],  
 [x1\_max, x2\_min, x3\_min],  
 [x1\_max, x2\_max, x3\_max]  
 ]  
 for i in range(len(x)):  
 x[i].append(x[i][0] \* x[i][1])  
 x[i].append(x[i][0] \* x[i][2])  
 x[i].append(x[i][1] \* x[i][2])  
 x[i].append(x[i][0] \* x[i][1] \* x[i][2])  
  
 if n == 4:  
 print('ŷ = b0 + b1 \* x1 + b2 \* x2 + b3 \* x3')  
 norm\_x = [  
 [+1, -1, -1, -1],  
 [+1, -1, +1, +1],  
 [+1, +1, -1, +1],  
 [+1, +1, +1, -1],  
 ]  
 x = [  
 [x1\_min, x2\_min, x3\_min],  
 [x1\_min, x2\_max, x3\_max],  
 [x1\_max, x2\_min, x3\_max],  
 [x1\_max, x2\_max, x3\_min],  
 ]  
 y = np.random.randint(y\_min, y\_max, size=(n, m))  
 y\_av = list(np.average(y, axis=1))  
  
 for i in range(len(y\_av)):  
 y\_av[i] = round(y\_av[i], 3)  
 if n == 8:  
 t = PrettyTable(['N', 'norm\_x\_0', 'norm\_x\_1', 'norm\_x\_2', 'norm\_x\_3', 'norm\_x\_1\_x\_2', 'norm\_x\_1\_x\_3', 'norm\_x\_2\_x\_3', 'norm\_x\_1\_x\_2\_x\_3', 'x\_1', 'x\_2', 'x\_3', 'x\_1\_x\_2', 'x\_1\_x\_3', 'x\_2\_x\_3', 'x\_1\_x\_2\_x\_3'] + [f'y\_{i + 1}' for i in range(m)] + ['y\_av'])  
 for i in range(n):  
 t.add\_row([i + 1] + list(norm\_x[i]) + list(x[i]) + list(y[i]) + [y\_av[i]])  
 print(t)  
 sums\_of\_columns\_x = np.sum(x, axis=0)  
 m\_ij = [[n] + [i for i in sums\_of\_columns\_x]]  
 for i in range(len(sums\_of\_columns\_x)):  
 m\_ij.append(  
 [sums\_of\_columns\_x[i]] + [sum([x[k][i] \* x[k][j] for k in range(len(x[i]))]) for j in range(len(x[i]))])  
  
 k\_i = [sum(y\_av)]  
 for i in range(len(sums\_of\_columns\_x)):  
 k\_i.append(sum(y\_av[j] \* x[j][i] for j in range(len(x[i]))))  
  
 det = np.linalg.det(m\_ij)  
 det\_i = [np.linalg.det(replace\_column(m\_ij, i, k\_i)) for i in range(len(k\_i))]  
  
 b\_i = [i / det for i in det\_i]  
  
 print(  
 f"\nThe normalized regression equation: y = {b\_i[0]:.5f} + {b\_i[1]:.5f} \* x1 + {b\_i[2]:.5f} \* x2 + "  
 f"{b\_i[3]:.5f} \* x3 + {b\_i[4]:.5f} \* x1 \* x2 + "  
 f"{b\_i[5]:.5f} \* x1 \* x3 + {b\_i[6]:.5f} \* x2 \* x3 + {b\_i[7]:.5f} \* x1 \* x2 \* x3")  
  
 if n == 4:  
 t = PrettyTable(  
 ['N', 'norm\_x\_0', 'norm\_x\_1', 'norm\_x\_2', 'norm\_x\_3', 'x\_1', 'x\_2', 'x\_3'] + [f'y\_{i + 1}' for i in  
 range(m)] + ['y\_av'])  
 for i in range(n):  
 t.add\_row([i + 1] + list(norm\_x[i]) + list(x[i]) + list(y[i]) + [y\_av[i]])  
 print(t)  
  
 mx\_1, mx\_2, mx\_3 = [i / len(x) for i in np.sum(x, axis=0)]  
 my = sum(y\_av) / len(y\_av)  
  
 a\_1 = sum([x[i][0] \* y\_av[i] for i in range(len(x))]) / len(x)  
 a\_2 = sum([x[i][1] \* y\_av[i] for i in range(len(x))]) / len(x)  
 a\_3 = sum([x[i][2] \* y\_av[i] for i in range(len(x))]) / len(x)  
  
 a\_11 = sum([x[i][0] \*\* 2 for i in range(len(x))]) / len(x)  
 a\_22 = sum([x[i][1] \*\* 2 for i in range(len(x))]) / len(x)  
 a\_33 = sum([x[i][2] \*\* 2 for i in range(len(x))]) / len(x)  
 a\_12 = sum([x[i][0] \* x[i][1] for i in range(len(x))]) / len(x)  
 a\_13 = sum([x[i][0] \* x[i][2] for i in range(len(x))]) / len(x)  
 a\_23 = a\_32 = sum([x[i][1] \* x[i][2] for i in range(len(x))]) / len(x)  
  
 matrix = [  
 [1, mx\_1, mx\_2, mx\_3],  
 [mx\_1, a\_11, a\_12, a\_13],  
 [mx\_2, a\_12, a\_22, a\_32],  
 [mx\_3, a\_13, a\_23, a\_33]  
 ]  
  
 answers = [my, a\_1, a\_2, a\_3]  
  
 det = np.linalg.det(matrix)  
 det\_i = [np.linalg.det(replace\_column(matrix, i, answers)) for i in range(len(answers))]  
  
 b\_i = [i / det for i in det\_i]  
 print(  
 f"\nThe normalized regression equation: y = {b\_i[0]:.5f} + {b\_i[1]:.5f} \* x1 + {b\_i[2]:.5f} \* x2 + {b\_i[3]:.5f} \* x3\n")  
  
 print("\n[ Kohren's test ]")  
 f\_1 = m - 1  
 f\_2 = n  
 s\_i = [sum([(i - y\_av[j]) \*\* 2 for i in y[j]]) / m for j in range(len(y))]  
 g\_p = max(s\_i) / sum(s\_i)  
  
 KohrenTable = {3: 0.6841, 4: 0.6287, 5: 0.5892, 6: 0.5598, 7: 0.5365, 8: 0.5175, 9: 0.5017, 10: 0.4884,  
 range(11, 17): 0.4366, range(17, 37): 0.3720, range(37, 145): 0.3093}  
 g\_t = KohrenTable.get(m)  
  
 if g\_p < g\_t:  
 print(f"The variance is homogeneous: Gp = {g\_p:.5} < Gt = {g\_t}")  
 else:  
 print(f"The variance is not homogeneous Gp = {g\_p:.5} < Gt = {g\_t}\nStart again with m = m + 1")  
 return main(m=m + 1, n=n)  
  
 print("\n[ Student's test ]")  
 s2\_b = sum(s\_i) / n  
 s2\_beta\_s = s2\_b / (n \* m)  
 s\_beta\_s = sqrt(s2\_beta\_s)  
  
 beta\_i = [sum([norm\_x[i][j] \* y\_av[i] for i in range(len(norm\_x))]) / n for j in range(len(norm\_x))]  
  
 t = [abs(i) / s\_beta\_s for i in beta\_i]  
  
 f\_3 = f\_1 \* f\_2  
 t\_table = {8: 2.306, 9: 2.262, 10: 2.228, 11: 2.201, 12: 2.179, 13: 2.160, 14: 2.145, 15: 2.131, 16: 2.120,  
 17: 2.110, 18: 2.101, 19: 2.093, 20: 2.086, 21: 2.08, 22: 2.074, 23: 2.069, 24: 2.064, 25: 2.06}  
 d = deepcopy(n)  
 for i in range(len(t)):  
 if t\_table.get(f\_3) > t[i]:  
 beta\_i[i] = 0  
 d -= 1  
 if n == 8:  
 print(  
 f"\nThe normalized regression equation: y = {beta\_i[0]:.5f} + {beta\_i[1]:.5f} \* x1 + {beta\_i[2]:.5f} \* x2 + "  
 f"{beta\_i[3]:.5f} \* x3 + {beta\_i[4]:.5f} \* x1 \* x2 + "  
 f"{beta\_i[5]:.5f} \* x1 \* x3 + {beta\_i[6]:.5f} \* x2 \* x3 + {beta\_i[7]:.5f} \* x1 \* x2 \* x3")  
 check\_i = [  
 beta\_i[0] + beta\_i[1] \* i[0] + beta\_i[2] \* i[1] + beta\_i[3] \* i[2] + beta\_i[4] \* i[3] + beta\_i[5] \* i[4] +  
 beta\_i[6] \* i[5] + beta\_i[7] \* i[6] for i in x]  
 print("Values are normalized: ", check\_i)  
  
 if n == 4:  
 print(  
 f"\nThe normalized regression equation: y = {beta\_i[0]:.5f} + {beta\_i[1]:.5f} \* x1 + {beta\_i[2]:.5f} \* x2 + "  
 f"{beta\_i[3]:.5f} \* x3")  
 check\_i = [  
 beta\_i[0] + beta\_i[1] \* i[0] + beta\_i[2] \* i[1] + beta\_i[3] \* i[2] for i in x]  
 print("Values are normalized: ", check\_i)  
  
 print("\n[ Fisher's test ]")  
 f\_4 = n - d  
 s2\_ad = m / f\_4 \* sum([(check\_i[i] - y\_av[i]) \*\* 2 for i in range(len(y\_av))])  
 f\_p = s2\_ad / s2\_b  
  
 if f\_p > FishersTable[f\_3][f\_4]:  
 print(  
 f"fp = {f\_p} > ft = {FishersTable[f\_3][f\_4]}.\nThe mathematical model is not adequate to the experimental "  
 f"data\nStart again with m = m + 1")  
 main(m=m + 1, n=8)  
 else:  
 print(  
 f"fP = {f\_p} < fT = {FishersTable[f\_3][f\_4]}.\nThe mathematical model is adequate to the experimental data\n")  
  
  
main(m=3, n=4)

**Результати виконання роботи**





