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»**

# «Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план)»

Виконав:

студент групи ІО-82

Мартинюк Н.О.

Залікова книжка № IO-8213

Варіант: 212

Перевірив:

Регіда П.Г.

Київ 2020

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

# Завдання:

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, зазначеного далі.

4. Розрахувати коефіцієнти рівняння регресії і записати його. 5. Провести 3 статистичні перевірки.

**Варіант:**



**Код програми:**

from scipy.stats import t, f  
import numpy as np  
from itertools import product, combinations  
  
np.set\_printoptions(formatter={'float\_kind': lambda x: "%.2f" % (x)})  
  
gt = {12: {1: 0.5410, 2: 0.3924, 3: 0.3264, 4: 0.2880, 5: 0.2624, 6: 0.2439, 7: 0.2299, 8: 0.2187, 9: 0.2098, 10: 0.2020},  
 15: {1: 0.4709, 2: 0.3346, 3: 0.2758, 4: 0.2419, 5: 0.2159, 6: 0.2034, 7: 0.1911, 8: 0.1815, 9: 0.1736, 10: 0.1671}}  
tt = {24: 2.064, 30: 2.042, 32: 1.96} # m = [3, 6]  
ft = {1: 4.2, 2: 3.3, 3: 2.9, 4: 2.7, 5: 2.5, 6: 2.4}  
matrix\_with\_min\_max\_x = np.array([[-3, 8], [0, 6], [-6, 1]])  
m = 3  
  
def table\_student(prob, n, m):  
 x\_vec = [i\*0.0001 for i in range(int(5/0.0001))]  
 par = 0.5 + prob/0.1\*0.05  
 f3 = (m - 1) \* n  
 for i in x\_vec:  
 if abs(t.cdf(i, f3) - par) < 0.000005:  
 return i  
  
  
def table\_fisher(prob, n, m, d):  
 x\_vec = [i\*0.001 for i in range(int(10/0.001))]  
 f3 = (m - 1) \* n  
 for i in x\_vec:  
 if abs(f.cdf(i, n-d, f3)-prob) < 0.0001:  
 return i  
  
  
  
def students\_t\_test(norm\_matrix\_, Y\_matrix\_, N):  
 mean\_Y\_ = np.mean(Y\_matrix\_, axis=1)  
 dispersion\_Y = np.mean((Y\_matrix\_.T - mean\_Y\_) \*\* 2, axis=0)  
 mean\_dispersion = np.mean(dispersion\_Y)  
 sigma = np.sqrt(mean\_dispersion / (N \* m))  
 betta = np.mean(norm\_matrix\_.T \* mean\_Y\_, axis=1)  
 t = np.abs(betta) / sigma  
 if (m - 1) \* N > 32:  
 return np.where(t > table\_student(0.95, N, m))  
 return np.where(t > table\_student(0.95, N, m))  
  
  
def phisher\_criterion(Y\_matrix, d, N):  
 if d == N:  
 return False  
 Sad = (m / (N - d)) \* np.sum((check2 - mean\_Y)\*\*2)  
 mean\_dispersion = np.mean(np.mean((Y\_matrix.T - mean\_Y) \*\* 2, axis=0))  
 Fp = Sad / mean\_dispersion  
 if (m-1)\*N > 32:  
 if N-d > 6:  
 return table\_fisher(0.95, N, m, d)  
 return Fp < table\_fisher(0.95, N, m, d)  
 if N - d > 6:  
 return Fp < table\_fisher(0.95, N, m, d)  
 return Fp < table\_fisher(0.95, N, m, d)  
  
  
  
def cochran\_check(Y\_matrix\_, N):  
 mean\_Y = np.mean(Y\_matrix, axis=1)  
 dispersion\_Y = np.mean((Y\_matrix.T - mean\_Y) \*\* 2, axis=0)  
 Gp = np.max(dispersion\_Y) / (np.sum(dispersion\_Y))  
 fisher = table\_fisher(0.95, N, m, 1)  
 Gt = fisher / (fisher + (m - 1) - 2)  
 return Gp < Gt  
  
  
def make\_plan\_matrix\_from\_norm\_matrix(norm\_matrix):  
 plan\_matrix = np.empty((len(norm\_matrix), len(norm\_matrix[0])), dtype=np.float)  
 for i in range(len(norm\_matrix)):  
 for j in range(len(norm\_matrix[i])):  
 if norm\_matrix[i, j] == -1:  
 plan\_matrix[i, j] = matrix\_with\_min\_max\_x[j-1][0]  
 elif norm\_matrix[i, j] == 1 and j != 0:  
 plan\_matrix[i, j] = matrix\_with\_min\_max\_x[j-1][1]  
 elif norm\_matrix[i, j] == 1 and j == 0:  
 plan\_matrix[i, j] = 1  
 else:  
 mean = np.mean(matrix\_with\_min\_max\_x[j-1])  
 plan\_matrix[i, j] = norm\_matrix[i, j] \* (matrix\_with\_min\_max\_x[j-1][1] - mean) + mean  
 return plan\_matrix  
  
  
def make\_linear\_equation():  
 norm\_matrix = np.array(list(product("01", repeat=3)), dtype=np.int)  
 norm\_matrix[norm\_matrix == 0] = -1  
 norm\_matrix = np.insert(norm\_matrix, 0, 1, axis=1)  
 plan\_matrix = make\_plan\_matrix\_from\_norm\_matrix(norm\_matrix)  
 return norm\_matrix, plan\_matrix  
  
  
def make\_equation\_with\_interaction\_effect(current\_norm\_matrix, current\_plan\_matrix):  
 plan\_matr = current\_plan\_matrix  
 norm\_matrix = current\_norm\_matrix  
 combination = list(combinations(range(1, 4), 2))  
 for i in combination:  
 plan\_matr = np.append(plan\_matr, np.reshape(plan\_matr[:, i[0]] \* plan\_matr[:, i[1]], (len(norm\_matrix), 1)),axis=1)  
 norm\_matrix = np.append(norm\_matrix, np.reshape(norm\_matrix[:, i[0]] \* norm\_matrix[:, i[1]], (len(norm\_matrix), 1)), axis=1)  
 plan\_matr = np.append(plan\_matr, np.reshape(plan\_matr[:, 1] \* plan\_matr[:, 2] \* plan\_matr[:, 3], (len(norm\_matrix), 1)), axis=1)  
 norm\_matrix = np.append(norm\_matrix, np.reshape(norm\_matrix[:, 1] \* norm\_matrix[:, 2] \* norm\_matrix[:, 3], (len(norm\_matrix), 1)), axis=1)  
 return norm\_matrix, plan\_matr  
  
  
def make\_equation\_with\_quadratic\_terms(current\_norm\_matrix):  
 norm\_matrix\_second\_part = np.empty((3, 7))  
 key = 0  
 for i in range(3):  
 j = 0  
 while j < 7:  
 if j == key:  
 norm\_matrix\_second\_part[i][key] = -1.215  
 norm\_matrix\_second\_part[i][key + 1] = 1.215  
 j += 1  
 else:  
 norm\_matrix\_second\_part[i][j] = 0  
 j += 1  
 key += 2  
  
 norm\_matrix\_second\_part = np.insert(norm\_matrix\_second\_part, 0, 1, axis=0)  
 norm\_matrix = np.append(current\_norm\_matrix, norm\_matrix\_second\_part.T, axis=0)  
 plan\_matrix = make\_plan\_matrix\_from\_norm\_matrix(norm\_matrix)  
 plan\_matrix = make\_equation\_with\_interaction\_effect(norm\_matrix, plan\_matrix)[1]  
 plan\_matrix = np.append(plan\_matrix, plan\_matrix[:, 1:4] \*\* 2, axis=1)  
 norm\_matrix = make\_equation\_with\_interaction\_effect(norm\_matrix, plan\_matrix)[0]  
 norm\_matrix = np.append(norm\_matrix, norm\_matrix[:, 1:4] \*\* 2, axis=1)  
 return norm\_matrix, plan\_matrix  
  
  
count = 2  
flag\_of\_model = False  
while flag\_of\_model is False:  
 norm\_matrix = make\_linear\_equation()[0]  
 plan\_matr = make\_linear\_equation()[1]  
 if count == 1:  
 norm\_matrix = make\_equation\_with\_interaction\_effect(norm\_matrix, plan\_matr)[0]  
 plan\_matr = make\_equation\_with\_interaction\_effect(norm\_matrix, plan\_matr)[1]  
 elif count > 1:  
 plan\_matr = make\_equation\_with\_quadratic\_terms(norm\_matrix)[1]  
 norm\_matrix = make\_equation\_with\_quadratic\_terms(norm\_matrix)[0]  
 plan\_matr\_for\_calc\_Y = plan\_matr  
 N = len(plan\_matr)  
 Y\_matrix = []  
 mean\_Y = []  
 indexes = []  
 flag\_of\_dispersion = False  
 while flag\_of\_dispersion is False:  
 Y\_matrix = np.random.randint(200 + np.mean(matrix\_with\_min\_max\_x, axis=0)[0],  
 200 + np.mean(matrix\_with\_min\_max\_x, axis=0)[1], size=(N, m))  
 mean\_Y = np.mean(Y\_matrix, axis=1)  
 if cochran\_check(Y\_matrix, N):  
 flag\_of\_dispersion = True  
 b\_natura = np.linalg.lstsq(plan\_matr, mean\_Y, rcond=None)[0]  
 b\_norm = np.linalg.lstsq(norm\_matrix, mean\_Y, rcond=None)[0]  
 check1 = np.sum(b\_natura \* plan\_matr, axis=1)  
 indexes = students\_t\_test(norm\_matrix, Y\_matrix, N)  
 check2 = np.sum(b\_natura[indexes] \* np.reshape(plan\_matr[:, indexes], (N, np.size(indexes))), axis=1)  
 print("Матриця плану експерименту: \n", plan\_matr)  
 print("Нормована матриця: \n", norm\_matrix)  
 print("Матриця відгуків: \n", Y\_matrix)  
 print("Середні значення У: ", mean\_Y)  
 print("Натуралізовані коефіціенти: ", b\_natura)  
 print("Перевірка 1: ", check1)  
 print("Індекси коефіціентів, які задовольняють критерію Стьюдента: ", np.array(indexes)[0])  
 print("Критерій Стьюдента: ",check2)  
 else:  
 m += 1  
 print("Дисперсія неоднорідна!")  
 if phisher\_criterion(Y\_matrix, np.size(indexes), N):  
 flag\_of\_model = True  
 print("Рівняння регресії адекватно оригіналу.")  
 else:  
 count += 1  
 print("Рівняння регресії неадекватно оригіналу.")

**Результати виконання:**

Матриця плану експерименту:

[[1.00 -3.00 0.00 -6.00 -0.00 18.00 -0.00 0.00 9.00 0.00 36.00]

[1.00 -3.00 0.00 1.00 -0.00 -3.00 0.00 -0.00 9.00 0.00 1.00]

[1.00 -3.00 6.00 -6.00 -18.00 18.00 -36.00 108.00 9.00 36.00 36.00]

[1.00 -3.00 6.00 1.00 -18.00 -3.00 6.00 -18.00 9.00 36.00 1.00]

[1.00 8.00 0.00 -6.00 0.00 -48.00 -0.00 -0.00 64.00 0.00 36.00]

[1.00 8.00 0.00 1.00 0.00 8.00 0.00 0.00 64.00 0.00 1.00]

[1.00 8.00 6.00 -6.00 48.00 -48.00 -36.00 -288.00 64.00 36.00 36.00]

[1.00 8.00 6.00 1.00 48.00 8.00 6.00 48.00 64.00 36.00 1.00]

[1.00 -4.18 3.00 -2.50 -12.55 10.46 -7.50 31.37 17.49 9.00 6.25]

[1.00 9.18 3.00 -2.50 27.55 -22.96 -7.50 -68.87 84.32 9.00 6.25]

[1.00 2.50 -0.65 -2.50 -1.61 -6.25 1.61 4.03 6.25 0.42 6.25]

[1.00 2.50 6.65 -2.50 16.61 -6.25 -16.61 -41.53 6.25 44.16 6.25]

[1.00 2.50 3.00 -6.75 7.50 -16.88 -20.26 -50.64 6.25 9.00 45.60]

[1.00 2.50 3.00 1.75 7.50 4.38 5.26 13.14 6.25 9.00 3.07]

[1.00 2.50 3.00 -2.50 7.50 -6.25 -7.50 -18.75 6.25 9.00 6.25]]

Нормована матриця:

[[1.00 -1.00 -1.00 -1.00 1.00 1.00 1.00 -1.00 1.00 1.00 1.00]

[1.00 -1.00 -1.00 1.00 1.00 -1.00 -1.00 1.00 1.00 1.00 1.00]

[1.00 -1.00 1.00 -1.00 -1.00 1.00 -1.00 1.00 1.00 1.00 1.00]

[1.00 -1.00 1.00 1.00 -1.00 -1.00 1.00 -1.00 1.00 1.00 1.00]

[1.00 1.00 -1.00 -1.00 -1.00 -1.00 1.00 1.00 1.00 1.00 1.00]

[1.00 1.00 -1.00 1.00 -1.00 1.00 -1.00 -1.00 1.00 1.00 1.00]

[1.00 1.00 1.00 -1.00 1.00 -1.00 -1.00 -1.00 1.00 1.00 1.00]

[1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00]

[1.00 -1.22 0.00 0.00 -0.00 -0.00 0.00 -0.00 1.48 0.00 0.00]

[1.00 1.22 0.00 0.00 0.00 0.00 0.00 0.00 1.48 0.00 0.00]

[1.00 0.00 -1.22 0.00 -0.00 0.00 -0.00 -0.00 0.00 1.48 0.00]

[1.00 0.00 1.22 0.00 0.00 0.00 0.00 0.00 0.00 1.48 0.00]

[1.00 0.00 0.00 -1.22 0.00 -0.00 -0.00 -0.00 0.00 0.00 1.48]

[1.00 0.00 0.00 1.22 0.00 0.00 0.00 0.00 0.00 0.00 1.48]

[1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]]

Матриця відгуків:

[[199 204 200]

[198 201 202]

[201 200 201]

[204 198 204]

[197 204 200]

[198 197 199]

[197 197 204]

[198 204 204]

[203 202 201]

[197 201 201]

[197 204 204]

[197 202 197]

[199 203 197]

[203 197 204]

[204 198 197]]

Середні значення У: [201.00 200.33 200.67 202.00 200.33 198.00 199.33 202.00 202.00 199.67

201.67 198.67 199.67 201.33 199.67]

Натуралізовані коефіціенти: [200.22 -0.27 0.23 -0.07 0.03 -0.02 0.07 0.01 0.01 -0.01 0.01]

Перевірка 1: [201.50 201.14 200.21 201.85 200.61 198.58 198.65 201.62 201.47 199.88

199.93 200.08 200.00 200.67 200.16]

Індекси коефіціентів, які задовольняють критерію Стьюдента: [ 0 8 9 10]

Критерій Стьюдента: [200.68 200.34 200.25 199.91 201.30 200.96 200.87 200.53 200.38 201.14

200.35 199.83 200.63 200.22 200.25]

Рівняння регресії адекватно оригіналу.

**Висновок:** Під час виконання роботи проблем не виникало. Отримані результати збігаються з очікуваними.