Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 5

з дисципліни «Методи наукових досліджень»

на тему «Проведення трьохфакторного експерименту

при використанні рівняння регресії з урахуванням квадратичних членів.»

ВИКОНАВ:

студент 2 курсу

групи ІВ-91

Кучеренко І.Д.

Залікова – 9117

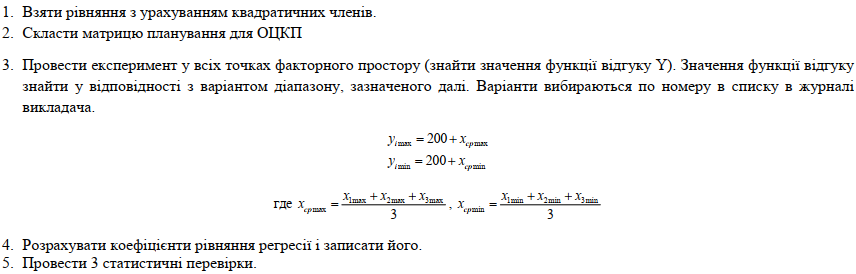
ПЕРЕВІРИВ:

ас. Регіда П. Г.

Київ – 2021

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

**Завдання на лабораторну роботу**



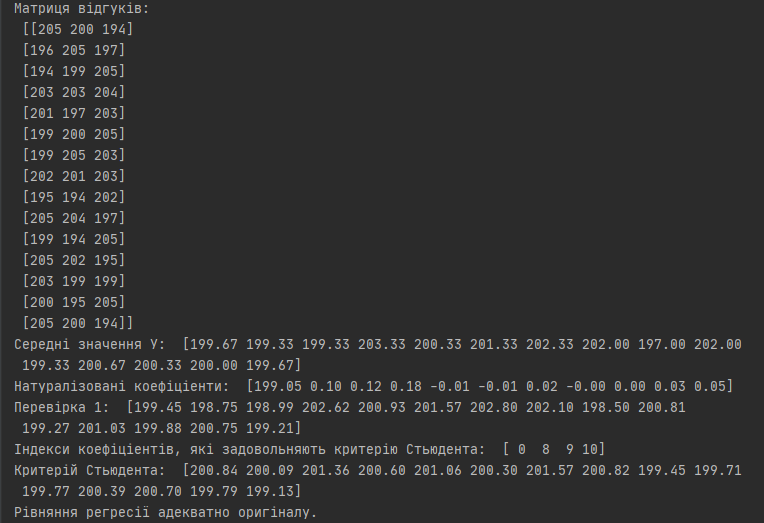
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № варіанту | x1 | | x2 | | x3 | |
| min | max | min | max | min | max |
| 116 | -7 | 10 | -4 | 6 | -5 | 3 |

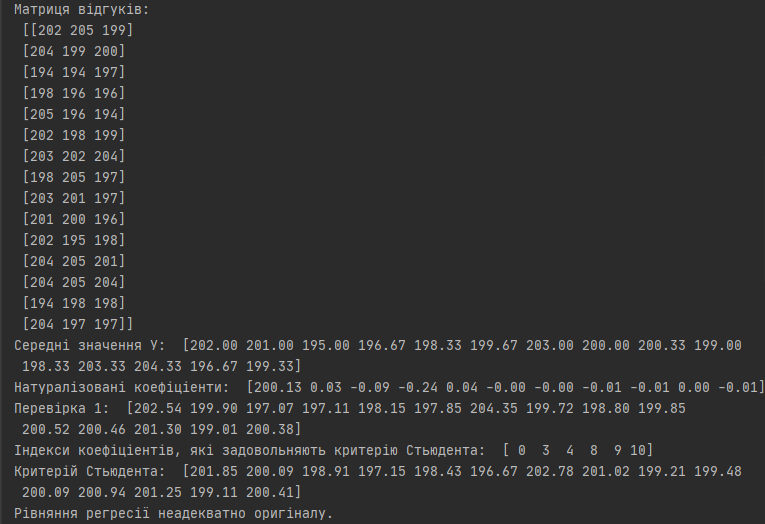
**Програмний код**

import numpy as np  
from itertools import product, combinations  
  
np.set\_printoptions(formatter={'float\_kind': lambda x: "%.2f" % (x)})  
Tt = 1.45  
Ft = 2.16  
Gt = 0.3346  
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([[-7, 10], [-4, 6], [-5, 3]])  
m = 3  
  
  
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  
 return np.where(t > Tt)  
  
  
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  
 return Fp < Ft  
  
  
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))  
 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.")

**Результат роботи програми**

Позитивний результат



Негативний результат

**Висновок:**

Під час виконання даної лабораторної роботи я провів трьохфакторний експеримент при використанні регресії з урахуванням квадратичних членів, перевірив однорідність дисперсії за критерієм Кохрена, отримав коефіцієнти рівняння регресії, оцінив значимість знайдених коефіцієнтів за критеріями Стьюдента та Фішера.

Мета лабораторної роботи була досягнута.