**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Національний Технічний Університет України**

**«Київський Політехнічний Інститут»**

*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-82

*Роман О.Т.*

**Перевірив:**

*Регіда П.Г.*

**Київ – 2020**





**Код програми:**

from \_pydecimal import Decimal  
from scipy.stats import f, t  
from random import randrange  
from math import sqrt, fabs as fab  
from numpy.linalg import solve  
  
  
#Варіант 217  
min\_x1, max\_x1 = -1, 6  
min\_x2, max\_x2 = -3, 4  
min\_x3, max\_x3 = -3, 7  
  
x01 = (max\_x1 + min\_x1) / 2  
x02 = (max\_x2 + min\_x2) / 2  
x03 = (max\_x3 + min\_x3) / 2  
  
delta\_x1 = max\_x1 - x01  
delta\_x2 = max\_x2 - x02  
delta\_x3 = max\_x3 - x03  
  
min\_y = 200 + int((min\_x1 + min\_x2 + min\_x3) / 3)  
max\_y = 200 + int((max\_x1 + max\_x2 + max\_x3) / 3)  
  
matrix\_of\_planning = [  
 [-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]  
]  
  
def matrixGenerator():  
 matrix\_with\_y = [[randrange(min\_y, max\_y)  
 for y in range(m)] for x in range(N)]  
 return matrix\_with\_y  
  
  
def middleValue(arr, orientation):  
 middle = []  
 if orientation == 1:  
 for rows in range(len(arr)):  
 middle .append(sum(arr[rows]) / len(arr[rows]))  
 else:  
 for column in range(len(arr[0])):  
 arr\_number = []  
 for rows in range(len(arr)):  
 arr\_number.append(arr[rows][column])  
 middle .append(sum(arr\_number) / len(arr\_number))  
 return middle  
  
  
def critCohrenValue(selectionSize, qty\_of\_selections, significance):  
 selectionSize += 1  
 partResult1 = significance / (selectionSize - 1)  
 params = [partResult1, qty\_of\_selections, (selectionSize - 1 - 1) \* qty\_of\_selections]  
 fisher = f.isf(\*params)  
 result = fisher / (fisher + (selectionSize - 1 - 1))  
 return Decimal(result).quantize(Decimal(**'.0001'**)).\_\_float\_\_()  
  
  
def critStudentValue(f3, significance):  
 return Decimal(abs(t.ppf(significance / 2, f3))).quantize(Decimal(**'.0001'**)).\_\_float\_\_()  
  
  
def critFisherValue(f3, f4, significance):  
 return Decimal(abs(f.isf(significance, f4, f3))).quantize(Decimal(**'.0001'**)).\_\_float\_\_()  
  
  
def x(l1, l2, l3): # зоряні точки  
 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 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\_known(number):  
 need\_a = 0  
 for j in range(N):  
 need\_a += middle\_y[j] \* matrix\_x[j][number - 1] / 15  
 return need\_a  
  
  
  
def check\_result(arr\_b, k):  
 y\_i = arr\_b[0] + arr\_b[1] \* matrix[k][0] + arr\_b[2] \* matrix[k][1] + arr\_b[3] \* matrix[k][2] + \  
 arr\_b[4] \* matrix[k][3] + arr\_b[5] \* matrix[k][4] + arr\_b[6] \* matrix[k][5] + arr\_b[7] \* matrix[k][6] + \  
 arr\_b[8] \* matrix[k][7] + arr\_b[9] \* matrix[k][8] + arr\_b[10] \* matrix[k][9]  
 return y\_i  
  
  
def student\_test(arr\_b, number\_x=10):  
 dispersion\_b = sqrt(dispersion\_b2)  
 for column in range(number\_x):  
 t\_practice = 0  
 t\_theoretical = critStudentValue(f3, q)  
 for row in range(N):  
 if column == 0:  
 t\_practice += middle\_y[row] / N  
 else:  
 t\_practice += middle\_y[row] \* matrix\_of\_planning[row][column - 1]  
 if fab(t\_practice / dispersion\_b) < t\_theoretical:  
 arr\_b[column] = 0  
 return arr\_b  
  
  
def fisher\_test():  
 dispersion\_ad = 0  
 f4 = N - d  
 for row in range(len(middle\_y)):  
 dispersion\_ad += (m \* (middle\_y[row] - check\_result(student\_arr, row))) / (N - d)  
 F\_practice = dispersion\_ad / dispersion\_b2  
 F\_theoretical = critFisherValue(f3, f4, q)  
 return F\_practice < F\_theoretical  
  
  
m, d = 0, 0  
N = 15  
correct\_input = False  
while not correct\_input:  
 try:  
 m = int(input(**"Введіть кількість повторень: "**))  
 p = float(input(**"Введіть довірчу імовірність: "**))  
 correct\_input = True  
 except ValueError:  
 pass  
  
  
matrix\_x = [[] for x in range(N)]  
for i in range(len(matrix\_x)):  
 if i < 8:  
 x1 = min\_x1 if matrix\_of\_planning[i][0] == -1 else max\_x1  
 x2 = min\_x2 if matrix\_of\_planning[i][1] == -1 else max\_x2  
 x3 = min\_x3 if matrix\_of\_planning[i][2] == -1 else max\_x3  
 else:  
 arr\_x = x(matrix\_of\_planning[i][0], matrix\_of\_planning[i][1], matrix\_of\_planning[i][2])  
 x1, x2, x3 = arr\_x  
 matrix\_x[i] = [x1, x2, x3, x1 \* x2, x1 \* x3, x2 \* x3, x1 \* x2 \* x3, x1 \*\* 2, x2 \*\* 2, x3 \*\* 2]  
  
matrix\_y = matrixGenerator()  
middle\_x = middleValue(matrix\_x, 0)  
middle\_y = middleValue(matrix\_y, 1)  
matrix = [(matrix\_x[i] + matrix\_y[i]) for i in range(N)]  
mx\_i = middle\_x  
my = sum(middle\_y) / 15  
  
values = [  
 [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)]  
]  
known\_values = [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, known\_values)  
print(**"Отримане рівняння регресії"**)  
print(**"^y = {:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"  
 "+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11^2 + {:.3f} \* X22^2 + {:.3f} \* X33^2"** .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\_result(beta, i), middle\_y[i]))  
  
  
while 1:  
 dispersion\_y = [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] - middle\_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)  
 Gt = critCohrenValue(f2, f1, q)  
 if Gt > Gp or m >= 25:  
 print(**"Дисперсія однорідна при рівні значимості {:.2f}"**.format(q))  
 break  
 else:  
 print(**"Дисперсія не однорідна при рівні значимості {:.2f}!"**.format(q))  
 m += 1  
 if m == 25:  
 exit()  
  
dispersion\_b2 = sum(dispersion\_y) / (N \* N \* m)  
student\_arr = list(student\_test(beta))  
print(**"Отримане рівняння регресії з урахуванням критерія Стьюдента:"**)  
print(**"^y = {:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"  
 "+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11^2 + {:.3f} \* X22^2 + {:.3f} \* X33^2"** .format(student\_arr[0], student\_arr[1], student\_arr[2], student\_arr[3], student\_arr[4], student\_arr[5],  
 student\_arr[6], student\_arr[7], student\_arr[8], student\_arr[9], student\_arr[10]))  
for i in range(N):  
 print(**"^y{} = {:.3f} ~= {:.3f}"**.format((i + 1), check\_result(student\_arr, i), middle\_y[i]))  
  
d = 11 - student\_arr.count(0)  
if fisher\_test():  
 print(**"Рівняння регресії адекватне стосовно оригіналу"**)  
else:  
 print(**"Рівняння регресії неадекватне стосовно оригіналу"**)

**Результати роботи програми:**

