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 № 5**

з дисципліни «Методи оптимізації та планування»

# Тема: Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів

**(центральний ортогональний композиційний план)**

ВИКОНАВ:

студент ІІ курсу ФІОТ

групи ІО-83 Коломієць Д.Ю. Залікова – 8313 Варіант-312

ПЕРЕВІРИВ:

Регіда П.Г.

Київ – 2020

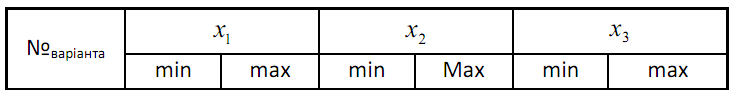
Мета роботи: Провести трьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Завдання на лабораторну роботу:

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. Варіанти вибираються по номеру в списку в журналі викладача.
4. Розрахувати коефіцієнти рівняння регресії і записати його.
5. Провести 3 статистичні перевірки.

Варіант:

312

****

**D:\screenshots\2020.04.15 19-39-08.jpg**

**Код програми:**

from \_pydecimal import Decimal, ROUND\_UP, ROUND\_FLOOR

from IPython.display import HTML, display

import random

import numpy as np

from scipy.stats import f, t, ttest\_ind, norm

class Criteries:

@staticmethod

def get\_cohren\_value(size\_of\_selections,qty\_of\_selections, significance ):

size\_of\_selections += 1

partResult1 = significance/(size\_of\_selections-1)

params = [partResult1, qty\_of\_selections, (size\_of\_selections-1-1)\*qty\_of\_selections]

fisher = f.isf(\*params)

result = fisher/(fisher+(size\_of\_selections-1-1))

return Decimal(result).quantize(Decimal('.0001')).\_\_float\_\_()

@staticmethod

def get\_student\_value(f3,significance):

return Decimal(abs(t.ppf(significance/2,f3))).quantize(Decimal('.0001')).\_\_float\_\_()

@staticmethod

def get\_fisher\_value(f3,f4,significance):

return Decimal(abs(f.isf(significance,f4,f3))).quantize(Decimal('.0001')).\_\_float\_\_()

cr = Criteries()

def dob(\*args):

res = [1 for \_ in range(len(args[0]))]

for i in range(len(args[0])):

for j in args:

res[i] \*= j[i]

return res

def getcolumn(arr, n):

return [i[n] for i in arr]

def getb(xs, ys):

forb = [getcolumn(xs, i) for i in range(len(xs[0]))]

ms = [[sum(dob(i, j))/N for j in forb] for i in forb]

k = [sum(dob(ys, i))/N for i in forb]

numerators = [[ms[i][0:j] + [k[i]] + ms[i][j+1:] for i in range(len(ms))] for j in range(len(ms))]

return [np.linalg.det(i)/np.linalg.det(ms) for i in numerators]

def getXnat(Xextr, Xnorm, level=0):

res = [[i[j]\*(Xextr[j][1] - sum(Xextr[j])/2)+sum(Xextr[j])/2 for j in range(len(i))] for i in Xnorm]

if level == 0:

return res

for i in range(len(res)):

res[i]+= [res[i][1]\*res[i][2],

res[i][1]\*res[i][3],

res[i][2]\*res[i][3],

res[i][1]\*res[i][2]\*res[i][3]]

if level == 2:

res[i] += [res[i][1]\*\*2,

res[i][2]\*\*2,

res[i][3]\*\*2]

return res

"""

x1min = 0

x1max = 2

x2min = -5

x2max = 9

x3min = -6

x3max = 10

"""

Xextr = ((1, 1), (0, 2), (-5, 9), (-6, 10))

Yextr = (200 + sum([i[0] for i in Xextr]), 200 + sum([i[1] for i in Xextr]))

m = 3

N = 15

Y = list(list(random.randrange(Yextr[0], Yextr[1]) for \_ in range(m)) for \_ in range(N))

p = 0.95

x0123n = (

(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),

(1, 1.215, 0, 0),

(1, 0, -1.215, 0),

(1, 0, 1.215, 0),

(1, 0, 0, -1.215),

(1, 0, 0, 1.215),

(1, 0, 0, 0)

)

while True:

Yavg = [sum(i) / len(i) for i in Y]

S = [sum([(Yavg[i] - Y[j][i]) \*\* 2 for i in range(m)]) / m for j in range(N)]

Gp = max(S) / sum(S)

F1 = m - 1

F2 = N

q = 1 - p

Gt = cr.get\_cohren\_value(F2, F1, q)

print("Y:", Y)

print("\nYavg:", Yavg)

print("Критерій Кохрена\nGp = %s Gt = %s" % (Gp, Gt))

if Gp < Gt:

print("Дисперсія однорідна.")

break

else:

print("Отримали неоднорідну дисперсію, збільшуємо m.")

for i in range(len(Y)):

m += 1

Y[i].append(random.randrange(Yextr[0], Yextr[1]))

level = 2

S2\_B = sum(Yavg) / N

while True:

Xcode = getXnat(((1, 1), (-1, 1), (-1, 1), (-1, 1)), x0123n, level)

Xnat = getXnat(Xextr, x0123n, level)

b = getb(Xnat, Yavg)

yr = [sum(dob(b, i)) for i in Xnat]

print('\nb:', b)

print("Значення Y для рівняння з отриманими коефіцієнтами:", yr)

S2\_beta = S2\_B / (N \* m)

beta = [sum(dob(getcolumn(Xcode, i), Yavg)) / N for i in range(len(Xnat[0]))]

ts = [abs(i) / ((S2\_beta) \*\* (1 / 2)) for i in beta]

F3 = F1 \* F2

tt = cr.get\_student\_value(F3, q)

bzn = [0 if ts[i] < tt else b[i] for i in range(len(b))]

d = len(ts) - ts.count(0)

yzn = [sum(dob(bzn, i)) for i in Xnat]

print("\nКритерій Стьюдента:")

print("tt = %s t = %s" % (tt, ts))

print("d = %s Значимі коефіцієнти: %s" % (d, bzn))

print("Значення Y враховуючи лише значимі коефіцієнти:", yzn)

# Fisher

print("\nКритерій Фішера:")

Sad = m \* sum([(yzn[i] - Yavg[i]) \*\* 2 for i in range(len(Yavg))]) / (N - d)

F4 = N - d

Ft = cr.get\_fisher\_value(F3, F4, q)

Fp = Sad / S2\_B

print("Sad = %s Ft = %s Fp = %s" % (Sad, Ft, Fp))

if Fp < Ft or level == 2:

break

else:

print("Отримане рівняння регресії - неадекватне. Збільшуємо к-сть членів ряду")

level += 1

print("Отримане рівняння регресії - адекватне.") if Fp < Ft else print("Отримане рівняння регресії - неадекватне.")

print("\n\nСередні/отримані/отримані лише зі значимими коефіцієнтами значення Y")

display(HTML(

'<table><tr>{}</tr></table>'.format(

'</tr><tr>'.join(

'<td>{}</td>'.format('</td><td>'.join(str(\_) for \_ in row)) for row in [Yavg, yr, yzn])

)

))

data = [[i + 1] + [round(j, 4) for j in Xnat[i]] + [j for j in Y[i]] + [round(Yavg[i], 4)] for i in range(15)]

foroutpX = 'x<sub>0</sub> x<sub>1</sub> x<sub>2</sub> x<sub>3</sub> '

foroutpX += 'x<sub>1</sub>x<sub>2</sub> x<sub>1</sub>x<sub>3</sub> x<sub>2</sub>x<sub>3</sub> x<sub>1</sub>x<sub>2</sub>x<sub>3</sub> ' if level >= 1 else ''

foroutpX += 'x<sub>1</sub><sup>2</sup> x<sub>2</sub><sup>2</sup> x<sub>3</sub><sup>2</sup> ' if level >= 2 else ''

for i in range(m):

foroutpX += "y<sub>%s</sub> " % (i + 1)

foroutpX += 'Y<sub>avg<sub>'

data.insert(0, ["№"] + list(foroutpX.split()))

display(HTML(

'<table><tr>{}</tr></table>'.format(

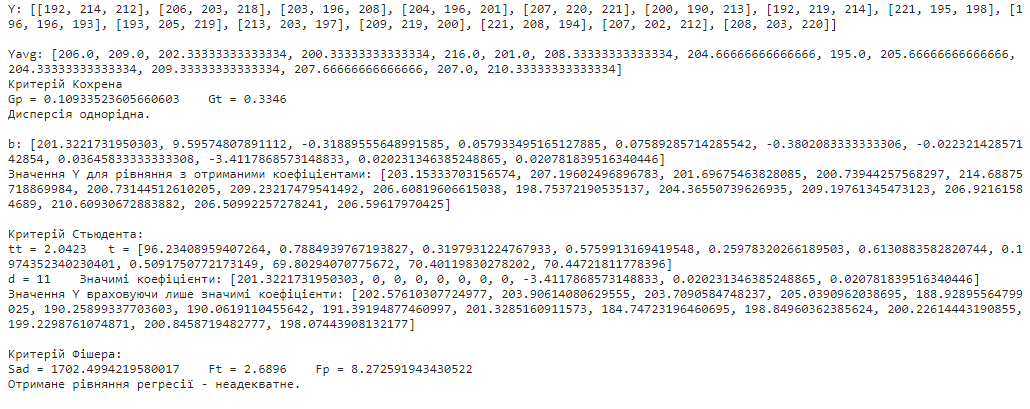
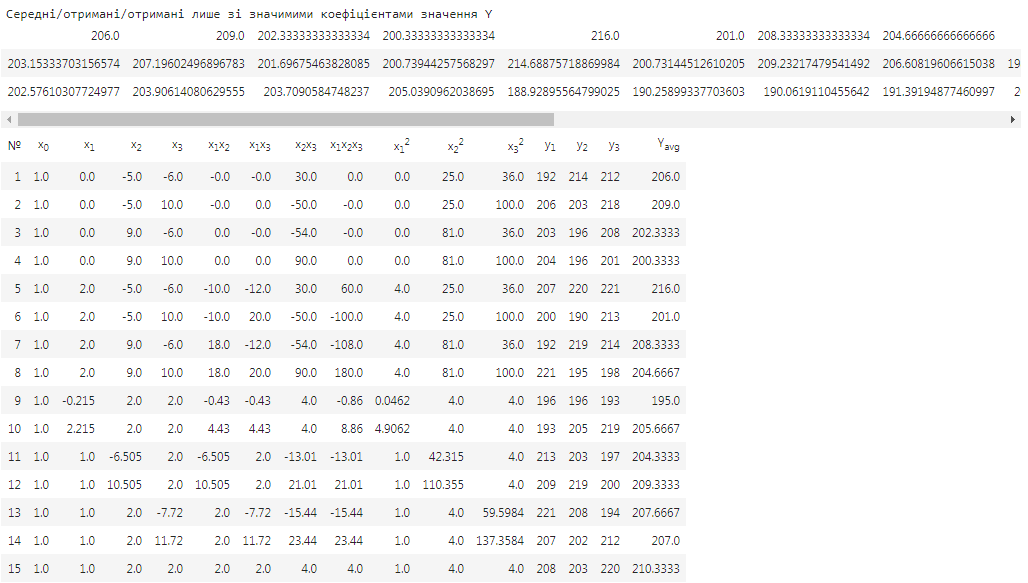
'</tr><tr>'.join(

'<td>{}</td>'.format('</td><td>'.join(str(\_) for \_ in row)) for row in data)

)

))  
else:  
 print("\n\nДисперсія неоднорідна!")

# Результат роботи програми:

**Висновок:**

Я провів повний трьохфакторний експеримент з урахуванням квадратичних членів, використовуючи центральний ортогональний композиційний план. Знайшов рівняння, яке буде адекватним для опису об’єкту