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 №6**

З дисципліни «Методи оптимізації та планування»

**Загальні принципи організації експериментів з**

**довільними значеннями факторів**

ВИКОНАB: Губський Артур

Студент 2 курсу

ФІОТ гр. ІO-92

ПЕРЕВІРИВ:

асистент

Регіда П.Г.

Київ 2021 р.

**Мета:**

Провести трьохфакторний експеремент і отримати адекватну модель-рівняння регресії використовуючи рототабельний композиційний план.

**Варіант завдання:**

**Лістинг програми:**

from math import fabs, sqrt

import time

m = 2

p = 0.95

N = 15

x1\_min = 10

x1\_max = 40

x2\_min = -15

x2\_max = 35

x3\_min = -15

x3\_max = 5

x01 = (x1\_max + x1\_min) / 2

x02 = (x2\_max + x2\_min) / 2

x03 = (x3\_max + x3\_min) / 2

delta\_x1 = x1\_max - x01

delta\_x2 = x2\_max - x02

delta\_x3 = x3\_max - x03

average\_y = None

matrix = None

dispersion\_b2 = None

student\_lst = None

d = None

q = None

f3 = None

class Perevirku:

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

from \_pydecimal import Decimal

from scipy.stats import f

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\_\_()

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

from \_pydecimal import Decimal

from scipy.stats import t

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

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

from \_pydecimal import Decimal

from scipy.stats import f

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

def generate\_matrix():

def f(X1, X2, X3):

from random import randrange

y = 9.9 + 2.7 \* X1 + 0.3 \* X2 + 1.4 \* X3 + 6.8 \* X1 \* X1 + 0.6 \* X2 \* X2 + 3.9 \* X3 \* X3 + 1.6 \* X1 \* X2 + \

0.1 \* X1 \* X3 + 1.7 \* X2 \* X3 + 9.9 \* X1 \* X2 \* X3 + randrange(0, 10) - 5

return y

matrix\_with\_y = [[f(matrix\_x[j][0], matrix\_x[j][1], matrix\_x[j][2]) for i in range(m)] for j in range(N)]

return matrix\_with\_y

def x(l1, l2, l3):

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 find\_average(lst, orientation):

average = []

if orientation == 1:

for rows in range(len(lst)):

average.append(sum(lst[rows]) / len(lst[rows]))

else:

for column in range(len(lst[0])):

number\_lst = []

for rows in range(len(lst)):

number\_lst.append(lst[rows][column])

average.append(sum(number\_lst) / len(number\_lst))

return average

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 += average\_y[j] \* matrix\_x[j][number - 1] / 15

return need\_a

def solve(lst\_1, lst\_2):

from numpy.linalg import solve

solver = solve(lst\_1, lst\_2)

return solver

def check\_result(b\_lst, k):

y\_i = b\_lst[0] + b\_lst[1] \* matrix[k][0] + b\_lst[2] \* matrix[k][1] + b\_lst[3] \* matrix[k][2] + \

b\_lst[4] \* matrix[k][3] + b\_lst[5] \* matrix[k][4] + b\_lst[6] \* matrix[k][5] + b\_lst[7] \* matrix[k][6] + \

b\_lst[8] \* matrix[k][7] + b\_lst[9] \* matrix[k][8] + b\_lst[10] \* matrix[k][9]

return y\_i

def student\_test(b\_lst, number\_x=10):

dispersion\_b = sqrt(dispersion\_b2)

for column in range(number\_x + 1):

t\_practice = 0

t\_theoretical = Perevirku.get\_student\_value(f3, q)

for row in range(N):

if column == 0:

t\_practice += average\_y[row] / N

else:

t\_practice += average\_y[row] \* matrix\_pfe[row][column - 1]

if fabs(t\_practice / dispersion\_b) < t\_theoretical:

b\_lst[column] = 0

return b\_lst

def fisher\_test():

dispersion\_ad = 0

f4 = N - d

for row in range(len(average\_y)):

dispersion\_ad += (m \* (average\_y[row] - check\_result(student\_lst, row))) / (N - d)

F\_practice = dispersion\_ad / dispersion\_b2

F\_theoretical = Perevirku.get\_fisher\_value(f3, f4, q)

return F\_practice < F\_theoretical

matrix\_pfe = [

[-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],

[-1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],

[+1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0, 0],

[0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],

[0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929, 0],

[0, 0, -1.73, 0, 0, 0, 0, 0, 0, 2.9929],

[0, 0, +1.73, 0, 0, 0, 0, 0, 0, 2.9929],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

]

matrix\_x = [[] for x in range(N)]

for i in range(len(matrix\_x)):

if i < 8:

x\_1 = x1\_min if matrix\_pfe[i][0] == -1 else x1\_max

x\_2 = x2\_min if matrix\_pfe[i][1] == -1 else x2\_max

x\_3 = x3\_min if matrix\_pfe[i][2] == -1 else x3\_max

else:

x\_lst = x(matrix\_pfe[i][0], matrix\_pfe[i][1], matrix\_pfe[i][2])

x\_1, x\_2, x\_3 = x\_lst

matrix\_x[i] = [x\_1, x\_2, x\_3, x\_1 \* x\_2, x\_1 \* x\_3, x\_2 \* x\_3, x\_1 \* x\_2 \* x\_3, x\_1 \*\* 2, x\_2 \*\* 2, x\_3 \*\* 2]

def run\_experiment():

adekvat = False

odnorid = False

global average\_y

global matrix

global dispersion\_b2

global student\_lst

global d

global q

global m

global f3

while not adekvat:

matrix\_y = generate\_matrix()

average\_x = find\_average(matrix\_x, 0)

average\_y = find\_average(matrix\_y, 1)

matrix = [(matrix\_x[i] + matrix\_y[i]) for i in range(N)]

mx\_i = average\_x

my = sum(average\_y) / 15

unknown = [

[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 = [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(unknown, known)

print("Отримане рівняння регресії")

print("{:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"

"+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11^2 + {:.3f} \* X22^2 + {:.3f} \* X33^2 = ŷ\n\tПеревірка"

.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("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check\_result(beta, i), average\_y[i]))

while not odnorid:

print("Матриця планування експеременту:")

print(" X1 X2 X3 X1X2 X1X3 X2X3 X1X2X3 X1X1"

" X2X2 X3X3 Yi ->")

for row in range(N):

print(end=' ')

for column in range(len(matrix[0])):

print("{:^12.3f}".format(matrix[row][column]), end=' ')

print("")

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] - average\_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)

print("Критерій Кохрена:")

Gt = Perevirku.get\_cohren\_value(f2, f1, q)

if Gt > Gp:

print("Дисперсія однорідна при рівні значимості {:.2f}.".format(q))

odnorid = True

else:

print("Дисперсія не однорідна при рівні значимості {:.2f}! Збільшуємо m.".format(q))

m += 1

dispersion\_b2 = sum(dispersion\_y) / (N \* N \* m)

student\_lst = list(student\_test(beta))

print("Отримане рівняння регресії з урахуванням критерія Стьюдента")

print("{:.3f} + {:.3f} \* X1 + {:.3f} \* X2 + {:.3f} \* X3 + {:.3f} \* Х1X2 + {:.3f} \* Х1X3 + {:.3f} \* Х2X3"

"+ {:.3f} \* Х1Х2X3 + {:.3f} \* X11^2 + {:.3f} \* X22^2 + {:.3f} \* X33^2 = ŷ\n\tПеревірка"

.format(student\_lst[0], student\_lst[1], student\_lst[2], student\_lst[3], student\_lst[4], student\_lst[5],

student\_lst[6], student\_lst[7], student\_lst[8], student\_lst[9], student\_lst[10]))

for i in range(N):

print("ŷ{} = {:.3f} ≈ {:.3f}".format((i + 1), check\_result(student\_lst, i), average\_y[i]))

print("Критерій Фішера")

d = 11 - student\_lst.count(0)

if fisher\_test():

print("Рівняння регресії адекватне оригіналу")

adekvat = True

else:

print("Рівняння регресії неадекватне оригіналу\n\t Проводимо експеремент повторно")

return adekvat

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

cnt = 0

adekvat = 0

while (time.time() - start) <= 10:

cnt += 1

try:

adekvat += run\_experiment()

except Exception:

continue

print(f'За 10 секунд експеремент був адекватним {adekvat} разів з {cnt}')

**Результат виконання роботи:**













