Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Методи оптимізації та планування експерименту**

Лабораторна робота №5:

«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів»

Виконав:

студент групи ІО-83

Бурлака Данило Сергійович

Залікова книжка № 8305

Перевірив Регіда П. Г.

Київ 2020р.

**Лабораторна робота №5**

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

(центральний ортогональний композиційний план)

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

**Виконання:**

Варіант – 305.





1. Лістинг програми:

import random

import math

import numpy as np

from \_pydecimal import Decimal

from scipy.stats import f, t

from functools import reduce

from itertools import compress

x1min, x2min, x3min = -10, -2, -1

x1max, x2max, x3max = 10, 2, 4

x\_average\_min = (x1min + x2min + x3min) / 3

x\_average\_max = (x1max + x2max + x3max) / 3

x0\_i = [(x1max+x1min)/2, (x2max+x2min)/2, (x3max+x3min)/2]

det\_x\_i = [(x1min - x0\_i[0]), (x2min-x0\_i[1]), (x3min-x0\_i[2])]

l = 1.215

m = 3

N = 15

ymin = round(200 + x\_average\_min)

ymax = round(200 + x\_average\_max)

def getStudentValue(f3, q):

return Decimal(abs(t.ppf(q/2,f3))).quantize(Decimal('.0001'))

def getFisherValue(f3,f4, q):

return Decimal(abs(f.isf(q,f4,f3))).quantize(Decimal('.0001'))

def criteriaStudent(m, N, y\_table, beta\_coefficients):

print("\nValidation of regression coefficients using Student's criteria: ".format(m, N))

average\_variation = np.average(list(map(np.var, y\_table)))

variation\_beta\_s = average\_variation/N/m

standard\_deviation\_beta\_s = math.sqrt(variation\_beta\_s)

t\_i = np.array([abs(beta\_coefficients[i])/standard\_deviation\_beta\_s for i in range(len(beta\_coefficients))])

f3 = (m-1)\*N

q = 0.05

t = getStudentValue(f3, q)

importance = [True if el > t else False for el in list(t\_i)]

print("Coefficient estimates βs: " + ", ".join(list(map(lambda x: str(round(float(x), 3)), beta\_coefficients))))

print("Coefficient ts: " + ", ".join(list(map(lambda i: "{:.2f}".format(i), t\_i))))

print("f3 = {}; q = {}; table = {}".format(f3, q, t))

beta\_i = ["β0", "β1", "β2", "β3", "β12", "β13", "β23", "β123", "β11", "β22", "β33"]

importance\_to\_print = ["important" if i else "unimportant" for i in importance]

to\_print = map(lambda x: x[0] + " " + x[1], zip(beta\_i, importance\_to\_print))

x\_i\_names = list(compress(["", "x1", "x2", "x3", "x12", "x13", "x23", "x123", "x1^2", "x2^2", "x3^2"], importance))

betas\_to\_print = list(compress(beta\_coefficients, importance))

print(\*to\_print, sep="; ")

equation = " ".join(["".join(i) for i in zip(list(map(lambda x: "{:+.2f}".format(x), betas\_to\_print)),x\_i\_names)])

print("The regression equation without insignificant members: y = " + equation)

return importance

def createFactorTable(raw\_array):

return [row + [row[0] \* row[1], row[0] \* row[2], row[1] \* row[2], row[0] \* row[1] \* row[2]]

+ list(map(lambda x: round(x \*\* 2, 5), row))

for row in raw\_array]

def m\_ij(\*arrays):

return np.average(reduce(lambda accum, el: accum\*el, arrays))

def x\_i(i):

try:

assert i <= 10

except:

raise AssertionError("it should be smaller or equal 10")

with\_null\_factor = list(map(lambda x: [1] + x, createFactorTable(raw\_factors\_table)))

res = [row[i] for row in with\_null\_factor]

return np.array(res)

def cochranValue(f1, f2, q):

partResult1 = q / f2 # (f2 - 1)

params = [partResult1, f1, (f2 - 1) \* f1]

fisher = f.isf(\*params)

result = fisher/(fisher + (f2 - 1))

return Decimal(result).quantize(Decimal('.0001'))

def cochranCriteria(m, N, y\_table):

print("\nChecking the uniformity of dispersions by Cochran criteria: ".format(m, N))

y\_variations = [np.var(i) for i in y\_table]

max\_y\_variation = max(y\_variations)

gp = max\_y\_variation/sum(y\_variations)

f1 = m - 1

f2 = N

p = 0.95

q = 1-p

gt = cochranValue(f1,f2, q)

print("Gp = {}; Gt = {}; f1 = {}; f2 = {}; q = {:.2f}".format(gp, gt, f1, f2, q))

if gp < gt:

print("Gp < Gt => dispersions are uniform ")

return True

else:

print("Gp > Gt => dispersions are uneven ")

return False

def calculateTheoretical\_Y(x\_table, b\_coefficients, importance):

x\_table = [list(compress(row, importance)) for row in x\_table]

b\_coefficients = list(compress(b\_coefficients, importance))

y\_vals = np.array([sum(map(lambda x, b: x\*b, row, b\_coefficients)) for row in x\_table])

return y\_vals

def fisherCriteria(m, N, d, naturalized\_x\_table, y\_table, b\_coefficients, importance):

f3 = (m - 1) \* N

f4 = N - d

q = 0.05

theoretical\_y = calculateTheoretical\_Y(naturalized\_x\_table, b\_coefficients, importance)

theoretical\_values\_to\_print = list(zip(map(lambda x: "x1 = {0[1]}, x2 = {0[2]}, x3 = {0[3]}".format(x),

naturalized\_x\_table), theoretical\_y))

y\_averages = np.array(list(map(np.average, y\_table)))

s\_ad = m/(N-d)\*(sum((theoretical\_y-y\_averages)\*\*2))

y\_variations = np.array(list(map(np.var, y\_table)))

s\_v = np.average(y\_variations)

f\_p = float(s\_ad/s\_v)

f\_t = getFisherValue(f3, f4, q)

print("\nFisher's model adequacy check:".format(m, N))

print("Theoretical values of Y for different combinations of factors:")

print("\n".join(["{arr[0]}: y = {arr[1]}".format(arr=el) for el in theoretical\_values\_to\_print]))

print("Fp = {}, Ft = {}".format(f\_p, f\_t))

print("Fp < Ft => the model is adequate" if f\_p < f\_t else "Fp > Ft => the model is inadequate")

return True if f\_p < f\_t else False

rawNaturalizedFactorsTable = [[x1min, x2min, x3min],

[x1min, x2max, x3max],

[x1max, x2min, x3max],

[x1max, x2max, x3min],

[x1min, x2min, x3max],

[x1min, x2max, x3min],

[x1max, x2min, x3min],

[x1max, x2max, x3max],

[-l\*det\_x\_i[0]+x0\_i[0], x0\_i[1], x0\_i[2]],

[l\*det\_x\_i[0]+x0\_i[0], x0\_i[1], x0\_i[2]],

[x0\_i[0], -l\*det\_x\_i[1]+x0\_i[1], x0\_i[2]],

[x0\_i[0], l\*det\_x\_i[1]+x0\_i[1], x0\_i[2]],

[x0\_i[0], x0\_i[1], -l\*det\_x\_i[2]+x0\_i[2]],

[x0\_i[0], x0\_i[1], l\*det\_x\_i[2]+x0\_i[2]],

[x0\_i[0], x0\_i[1], x0\_i[2]]]

raw\_factors\_table = [[-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],

[+1.215, 0, 0],

[0, -1.215, 0],

[0, +1.215, 0],

[0, 0, -1.215],

[0, 0, +1.215],

[0, 0, 0]]

factorsTable = createFactorTable(raw\_factors\_table)

for row in factorsTable:

print(row)

naturalizedFactorsTable = createFactorTable(rawNaturalizedFactorsTable)

withNullFactor = list(map(lambda x: [1] + x, naturalizedFactorsTable))

y\_arr = [[random.randint(ymin, ymax) for \_ in range(m)] for \_ in range(N)]

while not cochranCriteria(m, N, y\_arr):

m += 1

y\_arr = [[random.randint(ymin, ymax) for \_ in range(m)] for \_ in range(N)]

y\_i = np.array([np.average(row) for row in y\_arr])

coefficients = [[m\_ij(x\_i(column)\*x\_i(row)) for column in range(11)] for row in range(11)]

freeValues = [m\_ij(y\_i, x\_i(i)) for i in range(11)]

betaCoefficients = np.linalg.solve(coefficients, freeValues)

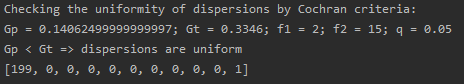
print(list(map(int,betaCoefficients)))

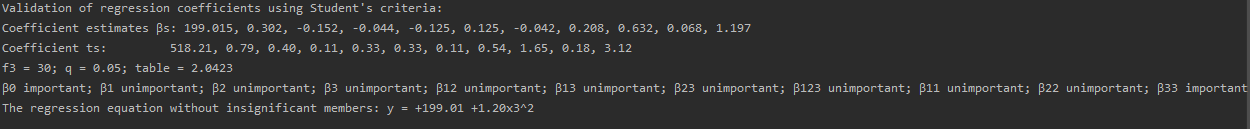
importance = criteriaStudent(m, N, y\_arr, betaCoefficients)

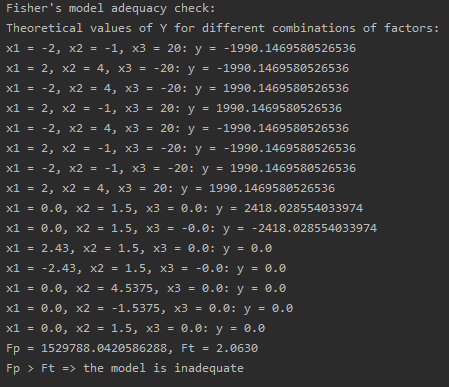
d = len(list(filter(None, importance)))

fisherCriteria(m, N, d, naturalizedFactorsTable, y\_arr, betaCoefficients, importance)

1. Результат виконання роботи програми:







**Висновок:** Отже, у ході виконання лабораторної роботи №5 провели трьохфакторний експеримент з урахуванням квадратичних членів ,використовуючи центральний ортогональний композиційний план. Знайшли рівняння регресії, яке буде адекватним для опису об'єкту, результати наведені вище. Кінцева мета роботи досягнута!