Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №3

з дисципліни МОПЕ

на тему:

# «ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ»

Виконав:

студент групи ІВ-82

Захарчук Д. С.

Залікова книжка № 8210

Варіант: 209

Перевірив:

Регіда П.Г.

Київ 2020

**Варіант:**

****

X1min = -20 X2min = -30 X3min = -30

X1max = 15 X2max = 45 X3max = -15

**Код програми:**

import numpy as np

from scipy.stats import t, f

import random

from typing import Tuple

np.set\_printoptions(precision=3)

class FactorExperiment:

'''

Lab3: Three-factor experiment with linear regression

'''

def \_\_init\_\_(self,

X1\_range: Tuple[float, float],

X2\_range: Tuple[float, float],

X3\_range: Tuple[float, float],

p: float) -> None:

'''

Init with variant constants

Arguments:

X1\_range {Tuple[float, float]} -- [x\_min\_1, x\_max\_1]

X2\_range {Tuple[float, float]} -- [x\_min\_2, x\_max\_2]

X3\_range {Tuple[float, float]} -- [x\_min\_3, x\_max\_3]

p {float} -- [probability]

'''

self.p = p

self.N = 4

self.M = 3

self.x\_mins = np.array(

[X\_range[0] for X\_range

in [X1\_range, X2\_range, X3\_range]])

self.x\_maxs = np.array(

[X\_range[1] for X\_range

in [X1\_range, X2\_range, X3\_range]])

self.x\_mean\_min = np.mean(self.x\_mins)

self.x\_mean\_max = np.mean(self.x\_maxs)

self.y\_min = 200 + self.x\_mean\_min

self.y\_max = 200 + self.x\_mean\_max

def experiment(self) -> None:

'''

Run all needed operations

'''

self.create\_plan\_matrix()

self.create\_norm\_matrix()

self.f1 = self.M - 1

self.f2 = self.N

self.f3 = self.f1 \* self.f2

self.find\_b()

print('\nCohren criterion:')

if not self.cohren\_criterion():

print(f'\nWrong!')

print('Change m={self.M} to m\'={self.M+1}\n')

self.M += 1

self.experiment()

self.student\_criterion()

self.f4 = self.N - self.d if self.N != self.d else 1

self.fisher\_criterion()

# Regression

print()

print(f'y = {self.b[0]:.2f} + {self.b[1]:.2f}\*x1 + ' +

f'{self.b[2]:.2f}\*x2 + {self.b[3]:.2f}\*x3')

def create\_plan\_matrix(self) -> None:

'''

Creates x and y arrays for plan experiment

'''

self.x = np.array(

[[random.random() \* (self.x\_maxs[i] - self.x\_mins[i]) + self.x\_mins[i]

for i in range(len(self.x\_mins))]

for j in range(self.N)]

)

self.y = np.array(

[[random.random() \* (self.y\_max - self.y\_min) + self.y\_min

for i in range(self.M)]

for j in range(self.N)]

)

print('Plan matrix:')

print('\nX values:')

print(self.x)

print('\nY values:')

print(self.y)

def create\_norm\_matrix(self) -> None:

'''

Normalize x array

'''

self.x\_norm = np.ndarray((self.x.shape[0], self.x.shape[1] + 1))

x0 = (self.x\_maxs + self.x\_mins) / 2

dx = x0 - self.x\_mins

for i in range(self.x.shape[0]):

for j in range(self.x.shape[1]):

self.x\_norm[i][1 + j] = (self.x[i][j] - x0[j]) / dx[j]

self.x\_norm[i][0] = 1

print('\nNorm matrix:\n', self.x\_norm)

def find\_b(self) -> None:

'''

Finds b-coefs for regression

'''

self.y\_means = self.y.mean(axis=1)

mx1 = np.mean(self.x.T[0])

mx2 = np.mean(self.x.T[1])

mx3 = np.mean(self.x.T[2])

my = np.mean(self.y\_means)

a1 = np.mean(self.x.T[0] \* self.y\_means)

a2 = np.mean(self.x.T[1] \* self.y\_means)

a3 = np.mean(self.x.T[2] \* self.y\_means)

a11 = np.mean(self.x.T[0] \*\* 2)

a22 = np.mean(self.x.T[1] \*\* 2)

a33 = np.mean(self.x.T[2] \*\* 2)

a12 = a21 = np.mean(self.x.T[0] \* self.x.T[1])

a13 = a31 = np.mean(self.x.T[0] \* self.x.T[2])

a23 = a32 = np.mean(self.x.T[1] \* self.x.T[2])

self.b = np.linalg.solve([[1, mx1, mx2, mx3],

[mx1, a11, a12, a13],

[mx2, a21, a22, a23],

[mx3, a31, a32, a33]],

[my, a1, a2, a3])

print(f'\nb coefs are:\n', self.b)

print()

regr = self.b[0] + self.x @ self.b[1:].T

for i, (y\_i, y\_i\_mean) in enumerate(zip(regr, self.y\_means)):

print(f'y{i+1} = {y\_i:.3f}, y{i+1} mean = {y\_i\_mean:.3f}')

def get\_cohren\_critical(self) -> float:

'''

Get table value of Cohren criterion

Returns:

float -- [criterion value]

'''

f\_crit = f.isf((1 - self.p) / self.f2,

self.f1,

(self.f2 - 1) \* self.f1)

return f\_crit / (f\_crit + self.f2 - 1)

def cohren\_criterion(self) -> bool:

'''

Checks Cohren's criterion

Returns:

bool -- [criterion result]

'''

self.variances = np.var(self.y, axis=1) \* self.M

Gp = max(self.variances) / sum(self.variances)

Gt = self.get\_cohren\_critical()

print(f'Gp: {Gp:.3f} Gt: {Gt:.3f}')

return Gp < Gt

def student\_criterion(self) -> None:

'''

Checks Student's criterion

'''

self.s2\_b = np.mean(self.variances) / (self.M \* self.N)

s\_b = np.sqrt(self.s2\_b)

b = np.abs(np.mean(self.x\_norm \* self.variances, axis=0))

t\_s = b / s\_b

t\_tabl = round(t.ppf((1 + self.p) / 2, self.f3), 3)

print('\nStudent criterion:')

print('Values for factors:\n', t\_s)

print('nFt:\n', t\_tabl)

print('\nValuables:')

valuable = t\_s > t\_tabl

self.d = sum(valuable)

for i in range(self.N):

print(f'X{i} is valuable: {valuable[i]}')

self.b \*= valuable

self.yh = self.b[0] + self.x @ self.b[1:].T

print('\nValues for y with significant factors:\n', self.yh)

def fisher\_criterion(self) -> None:

'''

Checks Fisher's criterion

'''

nd\_dif = self.N - self.d if self.N != self.d else 1

s2\_ad = sum([(self.yh[i] - self.y\_means[i]) \*\* 2 for i in range(self.N)]) \* self.M / nd\_dif

Fp = s2\_ad / self.s2\_b

Ft = f.ppf(self.p, self.f4, self.f3)

print('\nFisher criterion:')

if Fp > Ft:

print(f'OK with q = {1 - self.p:.2f}')

else:

print(f'Wrong with q = {1 - self.p:.2f}')

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

# My variant X ranges

X1\_range = [-20, 15]

X2\_range = [-30, 45]

X3\_range = [-30, -15]

# and probability

p = 0.95

# Init and run lab3 experiment

lab\_3 = FactorExperiment(X1\_range, X2\_range, X3\_range, p)

lab\_3.experiment()