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 (ЦЕНТРАЛЬНИЙ ОРТОГОНАЛЬНИЙ КОМПОЗИЦІЙНИЙ ПЛАН)»

Виконав:

студент групи ІВ-82

Захарчук Даниїл

Залікова книжка № 8210

Перевірив Регіда П. Г.

Київ 2020р.

**Мета:** провести трьохфакторний експеремент з урахуванням квадратичних членів, використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії адекватне об'єкту

**Виконання:** Варіант – 209



import numpy as np

import pandas as pd

from scipy.stats import t, f

import random

import sklearn.linear\_model as lm

from typing import Tuple

np.set\_printoptions(precision=3)

pd.set\_option('display.precision', 3)

class FactorExperiment:

'''

Lab5: Three-factor experiment with quadratic and interaction effects

Variables:

l: float {number} -- [the magnitude of the star shoulder]

x\_plan {[type]} -- [plan matrix for k = 3]

'''

l: float = 1.215

x\_plan = np.array(

[[-1, -1, -1],

[-1, -1, 1],

[-1, 1, -1],

[-1, 1, 1],

[1, -1, -1],

[1, -1, 1],

[1, 1, -1],

[1, 1, 1],

[-l, 0, 0],

[l, 0, 0],

[0, -l, 0],

[0, l, 0],

[0, 0, -l],

[0, 0, l],

[0, 0, 0]]

)

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

X1\_range: Tuple[int, int],

X2\_range: Tuple[int, int],

X3\_range: Tuple[int, int],

p: float = 0.95) -> 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 for statistical tests]

'''

self.p = p

self.N = 15

self.M = 3

X\_ranges = np.array([X1\_range, X2\_range, X3\_range])

self.x\_mins = X\_ranges[:, 0]

self.x\_maxs = X\_ranges[:, 1]

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\_naturalized\_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()

print('Stable variances!')

self.student\_criterion()

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

self.fisher\_criterion()

self.print\_regression()

def print\_regression(self) -> None:

'''

Prints regression with significance coefs

'''

print(f'\ny = {self.b[0]:.2f}', end='')

for i in range(1, len(self.b)):

if self.b[i]:

print(f' + {self.b[i]:.2f}\*{self.x.columns[i]}', end='')

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

'''

Creates x and y arrays for plan experiment

'''

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

dx = x0 - self.x\_mins

self.x = np.array(

[[self.x\_plan[j, i] \* dx[i] - x0[i]

if j > 8

else (self.x\_mins[i]

if self.x\_plan[j, i] == -1

else self.x\_maxs[i])

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

for j in range(self.N)]

)

self.x = pd.DataFrame(self.x, columns=['x1', 'x2', 'x3'])

self.x['b'] = 1

self.x = self.x.reindex(columns=['b', 'x1', 'x2', 'x3'])

self.x['x1\_x2'] = self.x.x1 \* self.x.x2

self.x['x1\_x3'] = self.x.x1 \* self.x.x3

self.x['x2\_x3'] = self.x.x2 \* self.x.x3

self.x['x1\_x2\_x3'] = self.x.x1 \* self.x.x2 \* self.x.x3

self.x['x1\_2'] = self.x.x1 \*\* 2

self.x['x2\_2'] = self.x.x2 \*\* 2

self.x['x3\_2'] = self.x.x3 \*\* 2

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('X values:')

print(self.x)

print('\nY values:')

print(self.y)

def find\_b(self) -> None:

'''

Finds b-coefs for regression

'''

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

regression = lm.LinearRegression(fit\_intercept=False)

regression.fit(self.x, self.y\_means)

self.b = regression.coef\_

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

print()

regression\_values = self.b[0] + self.x.drop('b', axis=1) @ self.b[1:].T

self.check\_regression(regression\_values)

def check\_regression(self, regression\_values: np.array) -> None:

'''

Compare regression results and true values

Arguments:

regression\_values {[type]} -- [regression values]

'''

for i, (y\_i, y\_i\_mean) in enumerate(zip(regression\_values, 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.variances \* self.x.T).T, 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.x.shape[1]):

print(f'Coef\_{i} is valuable: {valuable[i]}')

self.b \*= valuable

self.yh = self.b[0] + self.x.drop('b', axis=1) @ self.b[1:].T

print('\nValues for y with significant factors:\n')

self.check\_regression(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 = (-10, 9)

X2\_range = (0, 1)

X3\_range = (-3, 4)

# Init and run lab3 experiment

lab\_5 = FactorExperiment(X1\_range, X2\_range, X3\_range)

lab\_5.experiment()

**Висновок:**Отже, у ході виконання лабораторної роботи № 5 провели трьохфакторний експеримент при використанні рівняння з урахуванням квадратичних членів. Склали матрицю планування, знайшли коефіцієнти рівняння регресії, провели 3 статистичні перевірки. Була написана текстова програма, результати наведені вище. Кінцева мета роботи досягнута!