Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №6

з дисципліни МОПЕ

на тему:

# «Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

Виконав:

студент групи ІВ-82

Захарчук Даниїл Сергійович

Залікова книжка № 8210

Варіант: 209

Перевірив:

Регіда П.Г.

Київ – 2020

****

**Код програми:**

import numpy as np

import pandas as pd

from scipy.stats import t, f

import sklearn.linear\_model as lm

from typing import Tuple, TypeVar, Callable, Any, Union

from functools import wraps

from time import time

np.set\_printoptions(precision=3)

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

RT = TypeVar('RT') # generic return type

X = Union[np.array, pd.Series] # generic args type for variant function

Y = Union[np.array, pd.Series]

class FactorExperimentDecorators:

'''

Helper decorators for FactorExperiment

'''

@classmethod

def timeit(cls, func: Callable[..., RT]) -> Callable[..., RT]:

'''

Measure time for statistical checks

Decorators:

wraps - for debugging things

Arguments:

func {Callable[..., RT]} -- function to measure

Returns:

{Callable[..., RT]} -- wrappped function

'''

@wraps(func)

def \_wrapper(\*args: Any, \*\*kwargs: Any) -> RT:

'''

Return result of function and prints time of execution

Arguments:

\*args {Any} -- args of function

\*\*kwargs {Any} -- kwargs of function

Returns:

RT -- result of function

'''

start = time()

try:

return func(\*args, \*\*kwargs)

finally:

end = (time() - start) \* 1000

print(f'Total execution time: {end:.3f} ms')

return \_wrapper

class FactorExperiment:

'''

Lab6: Three-factor experiment with quadratic and interaction effects

Variables:

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

x\_plan {np.array} -- plan matrix for k = 3

'''

l: float = 1.73

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],

func: Callable[[X, X, X], Y],

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]

func {Callable[[X, X, X], Y]} -- variant function for y generation

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.func = func

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()

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()

self.print\_regression()

print('\nEnd of factor experiment')

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.generate\_y()

print('Plan matrix:')

print('X values:\n', self.x)

print('\nY values:\n', self.y)

def generate\_y(self) -> None:

'''

Using variant function generate y

'''

self.y = self.func(self.x.x1, self.x.x2, self.x.x3)

self.y += np.random.uniform(0, 10, size=self.y.shape) - 5

def find\_b(self) -> None:

'''

Finds b-coefs for regression

'''

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

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

self.b = regression.coef\_

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

self.check\_regression()

def check\_regression(self) -> None:

'''

Compare regression results and true values

'''

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

self.results = pd.DataFrame({

'y\_pred': self.yh,

'y\_true': self.y

})

print('Results:\n', self.results)

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)

@FactorExperimentDecorators.timeit

def cohren\_criterion(self) -> bool:

'''

Checks Cohren's criterion

Returns:

bool -- criterion result

'''

print('\nCohren criterion:')

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

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

Gt = self.get\_cohren\_critical()

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

return Gp < Gt

@FactorExperimentDecorators.timeit

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 = t.ppf((1 - self.p) / 2, self.f3)

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

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

if not all(valuable):

self.check\_regression()

@FactorExperimentDecorators.timeit

def fisher\_criterion(self) -> None:

'''

Checks Fisher's criterion

'''

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

s2\_ad = ((self.yh - self.y) \*\* 2).sum() \* self.M / nd\_dif

Fp = s2\_ad / self.s2\_b

Ft = f.isf(1 - self.p, self.f4, self.f3)

print('\nFisher criterion:')

print(f'Fp: {Fp}, Ft: {Fp}')

if Fp > Ft or (Fp - Ft) < 0.00000000001:

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 function

func = lambda x1, x2, x3: (5.4 + 3.4 \* x1 + 9.6 \* x2 + 6.8 \* x3

+ 0.8 \* x1 \* x2 + 0.8 \* x1 \* x3 + 9.9 \* x2 \* x3 + 4.5 \* x1 \* x2 \* x3

+ 3.1 \* x1\*\*2 + 0.1 \* x2\*\*2 + 1.2 \* x3\*\*2)

# Init and run lab6 experiment

lab\_6 = FactorExperiment(X1\_range, X2\_range, X3\_range, func)

lab\_6.experiment()