

Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки Кафедра  
обчислювальної техніки

**Методи планування експерименту**

**Лабораторна робота №4**

«Проведення трьохфакторного експерименту при використанні рівняння  
регресії з урахуванням ефекту взаємодії»

**Виконав:**

студент II курсу ФІОТ

групи ІВ-92

Салун Кирило

номер у списку групи – 20

**Перевірив:**

ас. Регіда П. Г.

Київ – 2021

## Мета:

Провести повний трьохфакторний експеримент. Знайти рівняння регресії адекватне об'єкту.

1. Скласти матрицю планування для повного трьохфакторного експерименту.
2. Провести експеримент, повторивши N раз досліди у всіх точках факторного простору і знайти значення відгуку Y. Знайти значення Y шляхом моделювання випадкових чисел у певному діапазоні відповідно варіанту. Варіанти вибираються за номером в списку в журналі викладача.

$$y_{i\max} = 200 + x_{cp\max}$$

$$y_{i\min} = 200 + x_{cp\min}$$

$$\text{де } x_{cp\max} = \frac{x_{1\max} + x_{2\max} + x_{3\max}}{3}, \quad x_{cp\min} = \frac{x_{1\min} + x_{2\min} + x_{3\min}}{3}$$

3. Знайти коефіцієнти рівняння регресії і записати його.
4. Провести 3 статистичні перевірки – за критеріями Кохрена, Стюдента, Фішера.
5. Зробити висновки по адекватності регресії та значимості окремих коефіцієнтів і записати скореговане рівняння регресії.
6. Написати комп'ютерну програму, яка усе це моделює.

## Завдання:

Варіанти обираються по номеру в списку в журналі викладача.

## Варіант завдання:

№ <sub>варіанта</sub>	$x_1$		$x_2$		$x_3$	
	min	max	min	Max	min	max
220	15	45	-70	-10	15	30

## Лістинг програми:

```
for import random
from typing import Tuple

import numpy
import sklearn.linear_model as lm
```

```

from numpy.linalg import solve
from scipy.stats import f, t

counter = 0

class LaboratoryWorkN4:

    def __init__(self, n_number: int, m_number: int, *x_args: Tuple[int, int]):
        self.n, self.m = n_number, m_number
        self.x_range = [element for element in x_args]
        self.y_min = 200 + round(sum([element[0] for element in self.x_range]) / 3)
        self.y_max = 200 + round(sum([element[1] for element in self.x_range]) / 3)

    def run(self):
        if not self.linear():
            self.with_interaction_effect()

    @staticmethod
    def get_regression(x, b):
        return sum([x[i] * b[i] for i in range(len(x))])

    @staticmethod
    def find_coefficients(x_matrix, y_matrix, normalized=False):
        skm = lm.LinearRegression(fit_intercept=False)
        skm.fit(x_matrix, y_matrix)
        b = skm.coef_

        if normalized:
            print('Коефіцієнти рівняння регресії з нормованими X:')
        else:
            print('Коефіцієнти рівняння регресії:')
        b = [round(i, 3) for i in b]
        print(f'\t{b}')
        return b

    def get_dispersion(self, y_matrix, y_average):
        result = []
        for i in range(self.n):
            s = sum([(y_average[i] - y_matrix[i][j]) ** 2 for j in range(self.m)]) /
self.m
            result.append(round(s, 3))
        return result

    def planing_matrix_interaction_effect(self):
        x_normalized = [[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]]
        y = numpy.zeros(shape=(self.n, self.m), dtype=numpy.int64)
        for i in range(self.n):
            for j in range(self.m):
                y[i][j] = random.randint(self.y_min, self.y_max)

        for x in x_normalized:
            x.append(x[1] * x[2])
            x.append(x[1] * x[3])
            x.append(x[2] * x[3])
            x.append(x[1] * x[2] * x[3])

```

```

x_normalized = numpy.array(x_normalized[:len(y)])
x = numpy.ones(shape=(len(x_normalized), len(x_normalized[0])),
dtype=numpy.int64)

for i in range(len(x_normalized)):
    for j in range(1, 4):
        if x_normalized[i][j] == -1:
            x[i][j] = self.x_range[j - 1][0]
        else:
            x[i][j] = self.x_range[j - 1][1]

for i in range(len(x)):
    x[i][4] = x[i][1] * x[i][2]
    x[i][5] = x[i][1] * x[i][3]
    x[i][6] = x[i][2] * x[i][3]
    x[i][7] = x[i][1] * x[i][3] * x[i][2]

print(f'\nМатриця планування для n = {self.n}, m = {self.m}:')
print('3 кодованими значеннями факторів:')
print('\t\t\t X0\t\t X1\t\t X2\t\t X3\t\t X1X2\t\t X1X3\t\t X2X3\t\t X1X2X3\t\t Y1\t\t Y2\t\t Y3')
print(numpy.concatenate((x, y), axis=1))
print(f'Нормовані значення факторів:\n{x_normalized}')
return x, y, x_normalized

def bs(self, x, y_average):
    res = [sum(1 * y for y in y_average) / self.n]
    for i in range(7):
        b = sum(j[0] * j[1] for j in zip(x[:, i], y_average)) / self.n
        res.append(b)
    return res

def student_criterion(self, x, y_average, dispersion):
    dispersion_average = sum(dispersion) / self.n
    s_beta_s = (dispersion_average / self.n / self.m) ** 0.5

    beta = [sum(1 * y for y in y_average) / self.n]
    for i in range(3):
        b = sum(j[0] * j[1] for j in zip(x[:, i], y_average)) / self.n
        beta.append(b)

    return [round(abs(b) / s_beta_s, 3) for b in beta]

def student_criterion_alt(self, x, y, y_average):
    S_kv = self.get_dispersion(y, y_average)
    s_kv_average = sum(S_kv) / self.n
    s_bs = (s_kv_average / self.n / self.m) ** 0.5
    bs_value = self.bs(x, y_average)
    return [round(abs(b) / s_bs, 3) for b in bs_value]

def fisher_criterion(self, y, y_average, y_new, d, dispersion):
    S_ad = self.m / (self.n - d) * sum([(y_new[i] - y_average[i]) ** 2 for i in
range(len(y))])
    dispersion_average = sum(dispersion) / self.n
    return S_ad / dispersion_average

def check(self, x_matrix, y_matrix, b):
    f1, f2 = self.m - 1, self.n
    f3 = f1 * f2
    q = 0.05

    y_average = [round(sum(i) / len(i), 3) for i in y_matrix]
    print(f'Середнє значення y:\n\t{y_average}')

```

```

dispersion_list = self.get_dispersion(y_matrix, y_average)

qq = (1 + 0.95) / 2
student_cr_table = t.ppf(df=f3, q=qq)

ts = self.student_criterion_alt(x_matrix[:, 1:], y_matrix, y_average)

temp_cochrane = f.ppf(q=(1 - q / f1), dfn=f2, dfd=(f1 - 1) * f2)
cochrane_cr_table = temp_cochrane / (temp_cochrane + f1 - 1)
gp = max(dispersion_list) / sum(dispersion_list)

print(f'Дисперсія y:\n\t{dispersion_list}')

print(f'Gp = {gp}')
if gp < cochrane_cr_table:
    print(f'З ймовірністю {1 - q} дисперсії є однорідними.')
else:
    print('Необхідно збільшити кількість дослідів!')
    self.m += 1
    self.with_interaction_effect()

print(f'Критерій Стюдента:\n\t{ts}')
res = [element for element in ts if element > student_cr_table]
final_k = [b[i] for i in range(len(ts)) if ts[i] in res]
print(f'Коефіцієнти {[round(i, 3) for i in b if i not in final_k]} '
      f'статистично незначущі, тому ми виключаємо їх з порівняння.')

y_new = []
for j in range(n):
    y_new.append(self.get_regression([x_matrix[j][i] for i in range(len(ts)) if
ts[i] in res], final_k))
print(f'Значення y з коефіцієнтами {final_k}:\n\t{y_new}')

d = len(res)
if d >= self.n:
    print('\nF4 <= 0')
    print('')
    return
f4 = self.n - d

fp = self.fisher_criterion(y_matrix, y_average, y_new, d, dispersion_list)
ft = f.ppf(dfn=f4, dfd=f3, q=1 - 0.05)
print('Перевірка адекватності за критерієм Фішера')
print(f'\tFp = {fp}')
print(f'\tFt = {ft}')
if fp < ft:
    print('\tМатематична модель адекватна експериментальним даним')
    return True
else:
    print('\tМатематична модель не адекватна експериментальним даним')
    return False

def with_interaction_effect(self):
    x, y, x_normalized = self.planing_matrix_interaction_effect()
    y_average = [round(sum(i) / len(i), 3) for i in y]
    b_normalized = self.find_coefficients(x_normalized, y_average, normalized=True)
    return self.check(x_normalized, y, b_normalized)

def get_planning_matrix(self):
    x_normalized = numpy.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],
        [1, -1, 1, -1],
        [1, 1, -1, -1],
        [1, 1, 1, 1]])

y_matrix = numpy.zeros(shape=(self.n, self.m))
for i in range(self.n):
    for j in range(self.m):
        y_matrix[i][j] = random.randint(self.y_min, self.y_max)

x_normalized = x_normalized[:len(y_matrix)]
x_matrix = numpy.ones(shape=(len(x_normalized), len(x_normalized[0])))
for i in range(len(x_normalized)):
    for j in range(1, len(x_normalized[i])):
        if x_normalized[i][j] == -1:
            x_matrix[i][j] = self.x_range[j - 1][0]
        else:
            x_matrix[i][j] = self.x_range[j - 1][1]

print('Матриця планування:')
print('\tx0  x1  x2  x3  y1  y2  y3  ')
print(numpy.concatenate((x_matrix, y_matrix), axis=1))
return x_matrix, y_matrix, x_normalized

def regression_equation(self, x_matrix, y_matrix):
    y_average = [round(sum(i) / len(i), 2) for i in y_matrix]

    mx1 = sum(x_matrix[:, 1]) / self.n
    mx2 = sum(x_matrix[:, 2]) / self.n
    mx3 = sum(x_matrix[:, 3]) / self.n
    m_average = sum(y_average) / self.n

    a1 = sum([y_average[i] * x_matrix[i][1] for i in range(len(x_matrix))]) / self.n
    a2 = sum([y_average[i] * x_matrix[i][2] for i in range(len(x_matrix))]) / self.n
    a3 = sum([y_average[i] * x_matrix[i][3] for i in range(len(x_matrix))]) / self.n

    a12 = sum([x_matrix[i][1] * x_matrix[i][2] for i in range(len(x_matrix))]) /
self.n
    a13 = sum([x_matrix[i][1] * x_matrix[i][3] for i in range(len(x_matrix))]) /
self.n
    a23 = sum([x_matrix[i][2] * x_matrix[i][3] for i in range(len(x_matrix))]) /
self.n

    a11 = sum([i ** 2 for i in x_matrix[:, 1]]) / self.n
    a22 = sum([i ** 2 for i in x_matrix[:, 2]]) / self.n
    a33 = sum([i ** 2 for i in x_matrix[:, 3]]) / self.n

    x = [[1, mx1, mx2, mx3], [mx1, a11, a12, a13], [mx2, a12, a22, a23], [mx3, a13,
a23, a33]]
    y = [m_average, a1, a2, a3]
    b = [round(i, 2) for i in solve(x, y)]

    print('Рівняння регресії:')
    print(f'\ty = {b[0]} + {b[1]}x1 + {b[2]}x2 + {b[3]}x3')
    return y_average, b

def linear(self):
    f1, f2 = self.m - 1, self.n
    f3 = f1 * f2
    q = 0.05

    x, y, x_normalized = self.get_planning_matrix()

```

```

y_average, b = self.regression_equation(x, y)
dispersion_list = self.get_dispersion(y, y_average)

temp_cochrane = f.ppf(q=(1 - q / f1), dfn=f2, dfd=(f1 - 1) * f2)
cochrane_cr_table = temp_cochrane / (temp_cochrane + f1 - 1)
gp = max(dispersion_list) / sum(dispersion_list)

print('Перевірка за критерієм Кохрена:')
print(f'\tРозрахункове значення: Gp = {gp}')
print(f'\tТабличне значення: Gt = {cochrane_cr_table}')
if gp < cochrane_cr_table:
    print(f'\tЗ імовірністю {1 - q} дисперсії є однорідними.')
else:
    print("\tНеобхідно збільшити кількість дослідів!")
    self.m += 1
    self.linear()

qq = (1 + 0.95) / 2
student_cr_table = t.ppf(df=f3, q=qq)
student_value = self.student_criterion(x_normalized[:, 1:], y_average,
dispersion_list)

print('Критерій Стюдента:')
print('\tТабличне значення критерій Стюдента:', student_cr_table)
print('\tРозрахункове значення критерій Стюдента:', student_value)
res_student_t = [temp for temp in student_value if temp > student_cr_table]
final_coefficients = [b[student_value.index(element)] for element in
student_value if element in res_student_t]
print(f'\tКоефіцієнти {[element for element in b if element not in
final_coefficients]} статистично незначущі.')

y_new = []
for j in range(n):
    y_new.append(self.get_regression(
        [x[j][student_value.index(i)] for i in student_value if i in
res_student_t], final_coefficients))
    print(f'Отримуємо значення рівня регресії для {self.m} дослідів:\n\t{y_new}')

d = len(res_student_t)
f4 = self.n - d
fp = self.fisher_criterion(y, y_average, y_new, d, dispersion_list)
ft = f.ppf(df=f4, dfd=f3, q=1 - 0.05)
counter = 0

print('Перевірка адекватності за критерієм Фішера:')
print(f'\tРозрахункове значення критерія Фішера: Fp = {fp}')
print(f'\tТабличне значення критерія Фішера: Ft = {ft}')
if fp < ft:
    counter += 1
    print('\tМатематична модель адекватна експериментальним даним')
    return True
else:
    print('\tМатематична модель не адекватна експериментальним даним')
    return False

# Точка входу в програму
for k in range(1, 100):
    n, m = 8, 3
    x1, x2, x3 = (-15, 45), (-70, -10), (15, 30)
    worker = LaboratoryWorkN4(n, m, x1, x2, x3)
    worker.run()
    print(counter)

```

## Результати виконання роботи:

Матриця планування:

	X0	X1	X2	X3	Y1	Y2	Y3
[[	1.	-15.	25.	-15.	210.	223.	217.]
[	1.	-15.	65.	5.	233.	228.	225.]
[	1.	30.	25.	5.	199.	203.	226.]
[	1.	30.	65.	-15.	226.	205.	225.]
[	1.	-15.	25.	5.	222.	205.	218.]
[	1.	-15.	65.	-15.	233.	202.	213.]
[	1.	30.	25.	-15.	226.	203.	220.]
[	1.	30.	65.	5.	231.	203.	206.]]

Рівняння регресії:

$$y = 212.01 + -0.1x_1 + 0.12x_2 + -0.02x_3$$

Перевірка за критерієм Кохрена:

Розрахункове значення:  $G_p = 0.22132601794081197$

Табличне значення:  $G_t = 0.815948432359917$

З імовірністю 0.95 дисперсії є однорідними.

Критерій Стюдента:

Табличне значення критерій Стюдента: 2.119905299221011

Розрахункове значення критерій Стюдента: [110.109, 1.186, 1.228, 0.085]

Коефіцієнти [-0.1, 0.12, -0.02] статистично незначущі.

Отримаємо значення рівня регресії для 3 дослідів:

[212.01, 212.01, 212.01, 212.01, 212.01, 212.01, 212.01, 212.01]

Перевірка адекватності за критерієм Фішера:

Розрахункове значення критерія Фішера:  $F_p = 1.8252218216003255$

Табличне значення критерія Фішера:  $F_t = 2.6571966002210865$

Математична модель адекватна експериментальним даним

Process finished with exit code 0

## Висновок:

У ході виконання лабораторної роботи проведено повний трьохфакторний експеримент. В ході дослідження було розроблено відповідну програму мовою програмування Python, яка моделює проведення повного трьохфакторного експерименту. Складено матрицю планування, знайдено коефіцієнти рівняння регресії, проведено 3 статистичні перевірки (критерії Кохрена, Стюдента та Фішера). Результати роботи, наведені у протоколі, підтверджують правильність виконання – кінцеву мету роботи було досягнуто.