

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

Лабораторна робота 2  
з дисципліни «Методи оптимізації та планування експерименту»

Виконав:  
Студент 2 курсу ФІОТ  
групи ІО-91  
Герейханов Тимур

Перевірив:  
асистент  
Регіда П.Г.

Київ 2021

**Мета:** провести двофакторний експеримент, перевірити однорідність дисперсії за критерієм Романовського, отримати коефіцієнти рівняння регресії, провести натуралізацію рівняння регресії.

$$y_{\max} = (30 - N_{\text{варіанту}}) * 10 = 250,$$

$$y_{\min} = (20 - N_{\text{варіанту}}) * 10 = 150.$$

Варіант:

№ <sub>варіанта</sub>	X <sub>1</sub>		X <sub>2</sub>	
	min	max	min	max
105	-30	20	15	50

### Роздруківка програми:

```
import numpy as np
from typing import List
np.set_printoptions(precision=3)

class Two_factor:
    def __init__(self, X1_range: List[int], X2_range: List[int], Y_range: List[int], m: int) -> None:
        self.Rcritical = {5:2, 6:2.16, 7:2.3, 8:2.43, 9:2.5, 10:2.62}
        self.X1_range = X1_range
        self.X2_range = X2_range
        self.Y_range = Y_range
        self.plan_matrix = np.array(
            [np.random.randint(*self.X1_range, size=3),
             np.random.randint(*self.X2_range, size=3)]).T
        self.x0 = [np.mean(self.X1_range), np.mean(self.X2_range)]
        self.norm_matrix = self.make_norm_plan_matrix()
        self.m = m
        self.experiment()
        self.b = self.find_b()
        self.a = self.find_a()
        self.check_b = self.check_b_koefs()
        self.check_a = self.check_a_koefs()

    def experiment(self):
        self.y_matrix = np.random.randint(*self.Y_range, size=(3, self.m))
        self.y_mean = np.mean(self.y_matrix, axis=1)

        self.y_var = np.var(self.y_matrix, axis=1)
        self.sigma = np.sqrt((2 * (2 * self.m - 2)) / (self.m * (self.m - 4)))

        if not self.check_r():
            print(f'\n Дісперсія неоднорідна! Змінимо m={self.m} to m={self.m+1}\n')
            self.m += 1
```

```

self.experiment()

def make_norm_plan_matrix(self) -> np.array:
    self.N = self.plan_matrix.shape[0]
    self.k = self.plan_matrix.shape[1]

    interval_of_change = [self.X1_range[1] - self.x0[0],
                          self.X2_range[1] - self.x0[1]]
    X_norm = [
        [(self.plan_matrix[i, j] - self.x0[j]) / interval_of_change[j]
         for j in range(self.k)]
        for i in range(self.N)
    ]
    return np.array(X_norm)

def check_r(self) -> bool:
    for i in range(len(self.y_var)):
        for j in range(len(self.y_var)):
            if i > j:
                if self.y_var[i] >= self.y_var[j]:
                    R = (abs((self.m - 2) * self.y_var[i] /
                             (self.m * self.y_var[j]) - 1) / self.sigma)
                else:
                    R = (abs((self.m - 2) * self.y_var[j] /
                             (self.m * self.y_var[i]) - 1) / self.sigma)
                if R > self.Rcritical[self.m]:
                    print('Variance isn\'t stable!')
                    return False
    return True

def find_b(self) -> np.array:
    mx1 = np.mean(self.norm_matrix[:, 0])
    mx2 = np.mean(self.norm_matrix[:, 1])

    a1 = np.mean(self.norm_matrix[:, 0] ** 2)
    a2 = np.mean(self.norm_matrix[:, 0] * self.norm_matrix[:, 1])
    a3 = np.mean(self.norm_matrix[:, 1] ** 2)

    my = np.mean(self.y_mean)
    a11 = np.mean(self.norm_matrix[:, 0] * self.y_mean)
    a22 = np.mean(self.norm_matrix[:, 1] * self.y_mean)

    b = np.linalg.solve([[1, mx1, mx2],
                        [mx1, a1, a2],
                        [mx2, a2, a3]],
                        [my, a11, a22])
    return b

```

```

def find_a(self) -> np.array:
    delta_x = [abs(self.X1_range[1] - self.X1_range[0]) / 2,
               abs(self.X2_range[1] - self.X2_range[0]) / 2]
    a = [(self.b[0] - self.b[1] * self.x0[0] / delta_x[0] -
          self.b[2] * self.x0[1] / delta_x[1]),
          self.b[1] / delta_x[0],
          self.b[2] / delta_x[1]]
    return np.array(a)

def check_b_koefs(self) -> np.array:
    return np.array([
        (self.b[0] + np.sum(self.b[1:3] * self.norm_matrix[i]))
        for i in range(self.N)])

def check_a_koefs(self) -> np.array:
    return np.array([
        (self.a[0] + np.sum(self.a[1:3] * self.plan_matrix[i]))
        for i in range(self.N)])

def check_results(self) -> None:
    print('Матриця планування:\n', self.plan_matrix)
    print('Нормована матриця:\n', self.norm_matrix)
    print('Матриця Y:\n', self.y_matrix)
    print('\nНормовані коефіцієнти: ', self.b)
    print('Натуралізовані коефіцієнти:', self.a)
    print('\nY середнє: ', self.y_mean)
    print('Перевірка нормованих коефіцієнтів: ', self.check_b)
    print('Перевірка натуралізованих коефіцієнтів: ', self.check_a)

if __name__ == '__main__':
    Y_max = 250
    Y_min = 150
    m = 5
    X1_range = [-30, 20]
    X2_range = [15, 50]
    Y_range = [Y_min, Y_max]
    res = Two_factor(X1_range, X2_range, Y_range, m)
    res.check_results()

```

**Результати роботи програми:**

Матриця планування:

```
[[ 8 35]
 [-1 31]
 [ 8 34]]
```

Нормована матриця:

```
[[ 0.52  0.143]
 [ 0.16 -0.086]
 [ 0.52  0.086]]
```

Матриця Y:

```
[[220 214 163 186 182]
 [201 180 206 216 174]
 [177 169 175 224 238]]
```

Нормовані коефіцієнти: [184.667 33.333 -63. ]

Натуралізовані коефіцієнти: [308.333 1.333 -3.6 ]

Y середнє: [193. 195.4 196.6]

Перевірка нормованих коефіцієнтів: [193. 195.4 196.6]

Перевірка натуралізованих коефіцієнтів: [193. 195.4 196.6]