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 №4
## «Проведення трьох факторного експерименту при використанні рівняння регресії з урахуванням ефекту взаємодії»

ВИКОНАВ:
студент ІІ курсу ФІОТ
групи ІО-91
Сниченков Д. А.
варіант – 21

ПЕРЕВІРИВ:
Регіда П. Г.

Київ – 2021

## Текст програми

```
from numpy.linalg import solve
from scipy.stats import f, t
import numpy as np
import random
import sklearn.linear_model as lm


# Main functions
def planning_matrix_linear(n, m, range_x):
    x_normalized = 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],
                             [1, 1, -1, -1],
                             [1, 1, 1, 1]])
    y = np.zeros(shape=(n, m))
    for i in range(n):
        for j in range(m):
            y[i][j] = random.randint(y_min, y_max)

    x_normalized = x_normalized[:len(y)]

    x = np.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[i][j] = range_x[j - 1][0]
            else:
                x[i][j] = range_x[j - 1][1]

    print('\nМатриця планування:')
    print('\n    X0  X1   X2   X3   Y1   Y2   Y3  ')
    print(np.concatenate((x, y), axis=1))

    return x, y, x_normalized


def planing_matrix_interaction_effect(n, m):
    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 = np.zeros(shape=(n, m), dtype=np.int64)
    for i in range(n):
        for j in range(m):
            y[i][j] = random.randint(y_min, 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 = np.array(x_normalized[:len(y)])
    x = np.ones(shape=(len(x_normalized), len(x_normalized[0])),
```

```
dtype=np.int64)

    for i in range(len(x_normalized)):
        for j in range(1, 4):
            if x_normalized[i][j] == -1:
                x[i][j] = x_range[j - 1][0]
            else:
                x[i][j] = 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 = {n}, m = {m}:')
    print('\nЗ кодованими значеннями факторів:')
    print('\n     X0    X1    X2    X3  X1X2  X1X3  X2X3 X1X2X3   Y1    Y2
Y3')
    print(np.concatenate((x, y), axis=1))
    print('\nНормовані значення факторів:\n')
    print(x_normalized)

    return x, y, x_normalized


def regression_equation(x, y, n):
    y_average = [round(sum(i) / len(i), 2) for i in y]

    mx1 = sum(x[:, 1]) / n
    mx2 = sum(x[:, 2]) / n
    mx3 = sum(x[:, 3]) / n

    my = sum(y_average) / n

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

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

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

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

    print('\nРівняння регресії:')
    print(f'y = {b[0]} + {b[1]}*x1 + {b[2]}*x2 + {b[3]}*x3')

    return y_average, b


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

```
    x, y, x_norm = planning_matrix_linear(n, m, x_range)

    y_average, b = regression_equation(x, y, n)

    dispersion_arr = dispersion(y, y_average, n, m)

    temp_cohren = f.ppf(q=(1 - q / f1), dfn=f2, dfd=(f1 - 1) * f2)
    cohren_cr_table = temp_cohren / (temp_cohren + f1 - 1)
    g_p = max(dispersion_arr) / sum(dispersion_arr)

    print('\nПеревірка за критерієм Кохрена:\n')
    print(f'Розрахункове значення: Gp = {g_p}'
          f'\nТабличне значення: Gt = {cohren_cr_table}')
    if g_p < cohren_cr_table:
        print(f'З ймовірністю {1-q} дисперсії однорідні.')
    else:
        print("Необхідно збільшити ксть дослідів")
        m += 1
        linear(n, m)

    qq = (1 + 0.95) / 2
    student_cr_table = t.ppf(df=f3, q=qq)
    student_t = student(x_norm[:, 1:], y_average, n, m, dispersion_arr)

    print('\nТабличне значення критерій Стьюдента:\n', student_cr_table)
    print('Розрахункове значення критерій Стьюдента:\n', student_t)
    res_student_t = [temp for temp in student_t if temp > student_cr_table]
    final_coefficients = [b[student_t.index(i)] for i in student_t if i in
res_student_t]
    print('Коефіцієнти {} статистично незначущі.'.
          format([i for i in b if i not in final_coefficients]))

    y_new = []
    for j in range(n):
        y_new.append(regression([x[j][student_t.index(i)] for i in student_t
if i in res_student_t],
                                final_coefficients))

    print(f'\nОтримаємо значення рівння регресії для {m} дослідів: ')
    print(y_new)

    d = len(res_student_t)
    f4 = n - d
    f_p = fisher(y, y_average, y_new, n, m, d, dispersion_arr)
    f_t = f.ppf(dfn=f4, dfd=f3, q=1 - 0.05)

    print('\nПеревірка адекватності за критерієм Фішера:\n')
    print('Розрахункове значення критерія Фішера: Fp =', f_p)
    print('Табличне значення критерія Фішера: Ft =', f_t)
    if f_p < f_t:
        print('Математична модель адекватна експериментальним даним')
        return True
    else:
        print('Математична модель не адекватна експериментальним даним')
        return False


def calculate_coefficient(x, y, norm=False):
    skm = lm.LinearRegression(fit_intercept=False)
    skm.fit(x, y)
    b = skm.coef_

    if norm == 1:
        print('\nКоефіцієнти рівняння регресії з нормованими X:')
```

```
    else:
        print('\nКоефіцієнти рівняння регресії:')
    b = [round(i, 3) for i in b]
    print(b)
    return b


def check(x, y, b, n, m):
    f1 = m - 1
    f2 = n
    f3 = f1 * f2
    q = 0.05

    y_aver = [round(sum(i) / len(i), 3) for i in y]
    print('\nСереднє значення y:', y_aver)

    dispersion_arr = dispersion(y, y_aver, n, m)

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

    ts = student_2(x[:, 1:], y, y_aver, n, m)

    temp_cohren = f.ppf(q=(1 - q / f1), dfn=f2, dfd=(f1 - 1) * f2)
    cohren_cr_table = temp_cohren / (temp_cohren + f1 - 1)
    g_p = max(dispersion_arr) / sum(dispersion_arr)

    print('Дисперсія y:', dispersion_arr)

    print(f'Gp = {g_p}')
    if g_p < cohren_cr_table:
        print(f'З ймовірністю {1-q} дисперсії однорідні.')
    else:
        print("Необхідно збільшити кількість дослідів")
        m += 1
        with_interaction_effect(n, m)

    print('\nКритерій Стьюдента:\n', ts)
    res = [te for te in ts if te > student_cr_table]
    final_k = [b[i] for i in range(len(ts)) if ts[i] in res]
    print('\nКоефіцієнти {} статистично незначущі, тому ми виключаємо їх з
рівняння.'.format(
        [round(i, 3) for i in b if i not in final_k]))

    y_new = []
    for j in range(n):
        y_new.append(regression([x[j][i] for i in range(len(ts)) if ts[i] in
res], final_k))

    print(f'\nЗначення "y" з коефіцієнтами {final_k}')
    print(y_new)

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

    f_p = fisher(y, y_aver, y_new, n, m, d, dispersion_arr)

    f_t = f.ppf(dfn=f4, dfd=f3, q=1 - 0.05)

    print('\nПеревірка адекватності за критерієм Фішера')
```

```
    print('Fp =', f_p)
    print('Ft =', f_t)
    if f_p < f_t:
        print('Математична модель адекватна експериментальним даним')
        return True
    else:
        print('Математична модель не адекватна експериментальним даним')
        return False


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


def dispersion(y, y_aver, n, m):
    result = []
    for i in range(n):
        s = sum([(y_aver[i] - y[i][j]) ** 2 for j in range(m)]) / m
        result.append(round(s, 3))
    return result


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


# Criteria
def student(x, y_average, n, m, dispersion):
    dispersion_average = sum(dispersion) / n
    s_beta_s = (dispersion_average / n / m) ** 0.5

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

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

    return te


def student_2(x, y, y_aver, n, m):
    s_kv = dispersion(y, y_aver, n, m)
    s_kv_aver = sum(s_kv) / n
    s_bs = (s_kv_aver / n / m) ** 0.5
    bss = bs(x, y_aver, n)
    ts = [round(abs(b) / s_bs, 3) for b in bss]

    return ts


def fisher(y, y_average, y_new, n, m, d, dispersion):
    s_ad = m / (n - d) * sum([(y_new[i] - y_average[i]) ** 2 for i in
range(len(y))])
    dispersion_average = sum(dispersion) / n

    return s_ad / dispersion_average


# Main
```

```
def with_interaction_effect(n, m):
    x, y, x_norm = planing_matrix_interaction_effect(n, m)

    y_aver = [round(sum(i) / len(i), 3) for i in y]

    b_norm = calculate_coefficient(x_norm, y_aver, norm=True)

    return check(x_norm, y, b_norm, n, m)


def main(n, m):
    if not linear(n, m):
        with_interaction_effect(n, m)


if __name__ == '__main__':
    x_range = ((-30, 20), (-70, -10), (-70, -40))

    y_max = 200 + int(sum([x[1] for x in x_range]) / 3)
    y_min = 200 + int(sum([x[0] for x in x_range]) / 3)

    main(8, 3)
```

## Результат виконання програми

```
Матриця планування:

    X0  X1   X2   X3   Y1   Y2   Y3
[[  1. -30. -70. -70. 179. 151. 152.]
 [  1. -30. -10. -40. 162. 176. 176.]
 [  1.  20. -70. -40. 168. 186. 183.]
 [  1.  20. -10. -70. 157. 179. 187.]
 [  1. -30. -70. -40. 146. 158. 150.]
 [  1. -30. -10. -70. 181. 173. 174.]
 [  1.  20. -70. -70. 152. 157. 186.]
 [  1.  20. -10. -40. 174. 179. 149.]]

Рівняння регресії:
y = 171.07 + 0.13*x1 + 0.14*x2 + -0.06*x3

Перевірка за критерієм Кохрена:

Розрахункове значення: Gp = 0.2585017811283241
Табличне значення: Gt = 0.815948432359917
З ймовірністю 0.95 дисперсії однорідні.

Табличне значення критерій Стьюдента:
 2.119905299221011
Розрахункове значення критерій Стьюдента:
 [79.021, 1.547, 1.938, 0.412]
Коефіцієнти [0.13, 0.14, -0.06] статистично незначущі.

Отримаємо значення рівння регресії для 3 дослідів:
[171.07, 171.07, 171.07, 171.07, 171.07, 171.07, 171.07, 171.07]

Перевірка адекватності за критерієм Фішера:

Розрахункове значення критерія Фішера: Fp = 2.550564319837783
Табличне значення критерія Фішера: Ft = 2.6571966002210865
Математична модель адекватна експериментальним даним

Process finished with exit code 0
```