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 трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів
(центральний ортогональний композиційний план)»

Виконала:
студентка групи ІО-92
Шолотюк Ганна Сергіївна
Номер залікової книжки № 9229
Номер у списку – 21

Перевірив:
Регіда Павло Геннадійович

Київ 2021

Мета: Провести трьохфакторний експеримент з урахуванням квадратичних членів, використовуючи центральний ортогональний композиційний план. Знайти рівняння регресії, яке буде адекватним для опису об'єкту.

Завдання на лабораторну роботу

1. Взяти рівняння з урахуванням квадратичних членів.

2. Скласти матрицю планування для ОЦКП

3. Провести експеримент у всіх точках факторного простору (знайти значення функції відгуку Y). Значення функції відгуку

знайти у відповідності з варіантом діапазону, зазначеного далі. Варіанти вибираються по номеру в списку в журналі

викладача.

4. Розрахувати коефіцієнти рівняння регресії і записати його.

5. Провести 3 статистичні перевірки.

Порядок виконання лабораторної роботи

1. Записати рівняння регресії з урахуванням квадратичних членів::

2. Вибрати з таблиці варіантів і записати в протокол інтервали значень x1, x2, x3.

3. Скласти матрицю планування для ОЦКП і заповнити нормованими значеннями. Початкова кількість дослідів m = 3.

4. Провести першу статистичну перевірку - перевірку однорідності дисперсії за критерієм Кохрена (якщо дисперсія не

однорідна, то збільшити m і почати з п.3).

5. Розрахувати коефіцієнти рівняння регресії, розв`язавши матричні рівняння. При розрахунку використовувати

натуральні значення x1, x2 і x3.

6. Провести другу статистичну перевірку і скорегувати рівняння регресії.

7. Провести третю статистичну перевірку.

8. Зробити висновки щодо перевірок 3-х критеріїв.

221	-10	9	0	1	-3	4

Код програми

```
import random
import math
from _pydecimal import Decimal
from scipy.stats import f, t, ttest_ind, norm
from functools import reduce
from itertools import compress
import numpy as np


raw_naturalized_factors_table = [[-10, 0, -3],
                                 [-10, 1, 4],
                                 [+1, 0, 4],
                                 [+1, 1, -3],

                                 [-10, 0, 4],
                                 [-10, 1, -3],
                                 [+1, 0, -3],
                                 [+1, 1, 4],

                                 [11.0425, +0.5,      +0.5],
                                 [-12.0425, +0.5,      +0.5],
                                 [-0.5,    +1.1075,      +0.5],
                                 [-0.5,    -0.1075,      +0.5],
                                 [-0.5,       +0.5, 4.7525],
                                 [-0.5,       +0.5, -3.7525],

                                 [-0.5,       +0.5,      +0.5]]

raw_factors_table = [[-1, -1, -1],
                     [-1, +1, +1],
                     [+1, -1, +1],
                     [+1, +1, -1],

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

                     [-1.215, 0, 0],
                     [+1.215, 0, 0],
                     [0, -1.215, 0],
                     [0, +1.215, 0],
                     [0, 0, -1.215],
                     [0, 0, +1.215],

                     [0, 0, 0]]


def generate_factors_table(raw_array):
    return [row + [row[0] * row[1], row[0] * row[2], row[1] * row[2], row[0] *
row[1] * row[2]]
     + list(map(lambda x: round(x ** 2, 5), row))
     for row in raw_array]


def x_i(i):
    try:
        assert i <= 10
    except:
        raise AssertionError("i must be smaller or equal 10")
```

```
    with_null_factor = list(map(lambda x: [1] + x,
generate_factors_table(raw_factors_table)))
    res = [row[i] for row in with_null_factor]
    return np.array(res)


def cochran_criteria(m, N, y_table):
    print("Перевірка рівномірності дисперсій за критерієм Кохрена: m = {}, N = {}
для таблиці y_table".format(m, N))
    y_variations = [np.var(i) for i in y_table]
    max_y_variation = max(y_variations)
    gp = max_y_variation/sum(y_variations)
    f1 = m - 1
    f2 = N
    p = 0.95
    q = 1-p
    gt = get_cochran_value(f1,f2, q)
    print("Gp = {}; Gt = {}; f1 = {}; f2 = {}; q = {:.2f}".format(gp, gt, f1, f2,
q))
    if gp < gt:
        print("Gp < Gt => дисперсії рівномірні - все правильно")
        return True
    else:
        print("Gp > Gt => дисперсії нерівномірні - треба ще експериментів")
        return False


def student_criteria(m, N, y_table, beta_coefficients):
    print("\nПеревірка значимості коефіцієнтів регресії за критерієм Стьюдента: m =
{}, N = {} "
          "для таблиці y_table та нормалізованих факторів".format(m, N))
    average_variation = np.average(list(map(np.var, y_table)))

    y_averages = np.array(list(map(np.average, y_table)))
    variation_beta_s = average_variation/N/m
    standard_deviation_beta_s = math.sqrt(variation_beta_s)
    x_vals = [x_i(i) for i in range(11)]
    # coefficients_beta_s = np.array([round(np.average(y_averages*x_vals[i]),3) for
i in range(len(x_vals))])
    t_i = np.array([abs(beta_coefficients[i])/standard_deviation_beta_s for i in
range(len(beta_coefficients))])
    f3 = (m-1)*N
    q = 0.05

    t = get_student_value(f3, q)
    importance = [True if el > t else False for el in list(t_i)]

    # print result data
    print("Оцінки коефіцієнтів βs: " + ", ".join(list(map(lambda x:
str(round(float(x), 3)), beta_coefficients))))
    print("Коефіцієнти ts:         " + ", ".join(list(map(lambda i:
"{:.2f}".format(i), t_i))))
    print("f3 = {}; q = {}; tтабл = {}".format(f3, q, t))
    beta_i = ["β0", "β1", "β2", "β3", "β12", "β13", "β23", "β123", "β11", "β22",
"β33"]
    importance_to_print = ["важливий" if i else "неважливий" for i in importance]
    to_print = map(lambda x: x[0] + " " + x[1], zip(beta_i, importance_to_print))
    x_i_names = list(compress(["", "x1", "x2", "x3", "x12", "x13", "x23", "x123",
"x1^2", "x2^2", "x3^2"], importance))
    betas_to_print = list(compress(beta_coefficients, importance))
    print(*to_print, sep="; ")
    equation = " ".join(["".join(i) for i in zip(list(map(lambda x:
```

```
"{:+.2f}".format(x), betas_to_print)),x_i_names)])
    print("Рівняння регресії без незначимих членів: y = " + equation)
    return importance


def calculate_theoretical_y(x_table, b_coefficients, importance):
    x_table = [list(compress(row, importance)) for row in x_table]
    b_coefficients = list(compress(b_coefficients, importance))
    y_vals = np.array([sum(map(lambda x, b: x*b, row, b_coefficients)) for row in
x_table])
    return y_vals


def fisher_criteria(m, N, d, naturalized_x_table, y_table, b_coefficients,
importance):
    f3 = (m - 1) * N
    f4 = N - d
    q = 0.05

    theoretical_y = calculate_theoretical_y(naturalized_x_table, b_coefficients,
importance)
    theoretical_values_to_print = list(zip(map(lambda x: "x1 = {0[1]}, x2 = {0[2]},
x3 = {0[3]}".format(x),naturalized_x_table),theoretical_y))

    y_averages = np.array(list(map(np.average, y_table)))
    s_ad = m/(N-d)*(sum((theoretical_y-y_averages)**2))
    y_variations = np.array(list(map(np.var, y_table)))
    s_v = np.average(y_variations)
    f_p = float(s_ad/s_v)
    f_t = get_fisher_value(f3, f4, q)

    print("\nПеревірка адекватності моделі за критерієм Фішера: m = {}, "
          "N = {} для таблиці y_table".format(m, N))
    print("Теоретичні значення y для різних комбінацій факторів:")
    print("\n".join(["{arr[0]}: y = {arr[1]}".format(arr=el) for el in
theoretical_values_to_print]))
    print("Fp = {}, Ft = {}".format(f_p, f_t))
    print("Fp < Ft => модель адекватна" if f_p < f_t else "Fp > Ft => модель
неадекватна")
    return True if f_p < f_t else False


def m_ij(*arrays):
    return np.average(reduce(lambda accum, el: accum*el, arrays))

def get_cochran_value(f1, f2, q):
    partResult1 = q / f2 # (f2 - 1)
    params = [partResult1, f1, (f2 - 1) * f1]
    fisher = f.isf(*params)
    result = fisher/(fisher + (f2 - 1))
    return Decimal(result).quantize(Decimal('.0001')).__float__()


def get_student_value(f3, q):
    return Decimal(abs(t.ppf(q/2,f3))).quantize(Decimal('.0001')).__float__()


def get_fisher_value(f3,f4, q):
    return Decimal(abs(f.isf(q,f4,f3))).quantize(Decimal('.0001')).__float__()
```

```
factors_table = generate_factors_table(raw_factors_table)
for row in factors_table:
    print(row)
naturalized_factors_table = generate_factors_table(raw_naturalized_factors_table)
with_null_factor = list(map(lambda x: [1] + x, naturalized_factors_table))

m = 3
N = 15
ymin = 196
ymax = 205
y_arr = [[random.randint(ymin, ymax) for _ in range(m)] for _ in range(N)]
while not cochran_criteria(m, N, y_arr):
    m+=1
    y_arr = [[random.randint(ymin, ymax) for _ in range(m)] for _ in range(N)]

y_i = np.array([np.average(row) for row in y_arr])

coefficients = [[m_ij(x_i(column)*x_i(row)) for column in range(11)] for row in
range(11)]

free_values = [m_ij(y_i, x_i(i)) for i in range(11)]

beta_coefficients = np.linalg.solve(coefficients, free_values)
print(list(map(int,beta_coefficients)))

importance = student_criteria(m, N, y_arr, beta_coefficients)
d = len(list(filter(None, importance)))
fisher_criteria(m, N, d, naturalized_factors_table, y_arr, beta_coefficients,
importance)
```

Результати виконання програми

```
[-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, -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, 1, 1, 1, 1, 1, 1, 1, 1]
[-1.215, 0, 0, -0.0, -0.0, 0, -0.0, 1.47623, 0, 0]
[1.215, 0, 0, 0.0, 0.0, 0, 0.0, 1.47623, 0, 0]
[0, -1.215, 0, -0.0, 0, -0.0, -0.0, 0, 1.47623, 0]
[0, 1.215, 0, 0.0, 0, 0.0, 0.0, 0, 1.47623, 0]
[0, 0, -1.215, 0, -0.0, -0.0, -0.0, 0, 0, 1.47623]
[0, 0, 1.215, 0, 0.0, 0.0, 0.0, 0, 0, 1.47623]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Перевірка рівномірності дисперсій за критерієм Кохрена: m = 3, N = 15 для таблиці y_table
Gp = 0.17539267015706803; Gt = 0.3346; f1 = 2; f2 = 15; q = 0.05
Gp < Gt => дисперсії рівномірні - все правильно
[199, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Перевірка значимості коефіцієнтів регресії за критерієм Стьюдента:
 m = 3, N = 15 для таблиці y_table та нормалізованих факторів
```

```
Оцінки коефіцієнтів βs: 199.814, 0.133, 0.002, -0.296, 0.5, 0.583, -0.833, 0.25, -0.694, 1.339, 0.097
Коефіцієнти ts:         563.45, 0.37, 0.01, 0.83, 1.41, 1.64, 2.35, 0.70, 1.96, 3.77, 0.27
f3 = 30; q = 0.05; ttабл = 2.0423
β0 важливий
β1 неважливий
β2 неважливий
β3 неважливий
β12 неважливий
β13 неважливий
β23 важливий
β123 неважливий
β11 неважливий
β22 важливий
β33 неважливий

Рівняння регресії без незначимих членів: y = +199.81 -0.83x23 +1.34x2^2

Перевірка адекватності моделі за критерієм Фішера: m = 3, N = 15 для таблиці y_table
Теоретичні значення y для різних комбінацій факторів:
x1 = 0, x2 = -3, x3 = 0: y = -1986.0949524488165
x1 = 1, x2 = 4, x3 = -10: y = -1943.3919582556462
x1 = 0, x2 = 4, x3 = 0: y = 221.23053360648848
x1 = 1, x2 = -3, x3 = 1: y = 214.36087274665132
x1 = 0, x2 = 4, x3 = 0: y = -1976.7252915889794
x1 = 1, x2 = -3, x3 = -10: y = -2011.0949524488162
x1 = 0, x2 = -3, x3 = 0: y = 211.86087274665132
x1 = 1, x2 = 4, x3 = 1: y = 217.89720027315516
x1 = 0.5, x2 = 0.5, x3 = 5.52125: y = 2204.4820371590204
x1 = 0.5, x2 = 0.5, x3 = -6.02125: y = -2403.4186082625592
x1 = 1.1075, x2 = 0.5, x3 = -0.55375: y = -99.3417230517695
x1 = -0.1075, x2 = 0.5, x3 = 0.05375: y = -99.5948480517695
x1 = 0.5, x2 = 4.7525, x3 = -0.25: y = -68.68475075503434
x1 = 0.5, x2 = -3.7525, x3 = -0.25: y = -81.84076369973647
x1 = 0.5, x2 = 0.5, x3 = -0.25: y = -99.4682855517695
Fp = 1334797.2728065862, Ft = 2.0921
Fp > Ft => модель неадекватна

Process finished with exit code 0
```