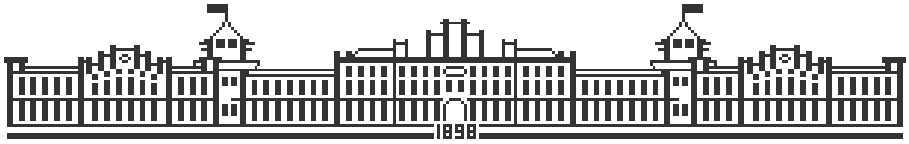

Міністерство освіти та науки України

Національний технічний університет України “Київський політехнічний інститут”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №5
з дисципліни «Методи оптимізації та планування»
на тему: «Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (центральний ортогональний композиційний план)»

Виконав:
студент 2-го курсу ФІОТ
групи ІВ-71
Мазан Я. В.

Перевірив:
асистент
Регіда П. Г.

Київ – 2019

Варіант:

№ 109

$x_{1min} = -3;$ $x_{2min} = 0;$ $x_{3min} = -8;$

$x_{1max} = 7;$ $x_{2max} = 9;$ $x_{3max} = 10;$

$x_{cpmax} = 8.667 \approx 9$

$x_{cpmin} = -3.667 \approx -4$

$y_{imax} = 209$

$y_{imin} = 196$

Код програми:

```
import random

from functions import *
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 = 209
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)
```

```
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 = [[-3, 0, -8],
                                 [-3, 9, 10],
                                 [+7, 0, 10],
                                 [+7, 9, -8],
                                 [-3, 0, 10],
                                 [-3, 9, -8],
                                 [+7, 0, -8],
                                 [+7, 9, 10],
                                 [-4.075, +4.5,     +1],
                                 [+8.075, +4.5,     +1],
                                 [2,   -0.9675,     +1],
                                 [2,   +9.9675,     +1],
                                 [2,      +4.5, -9.935],
                                 [2,      +4.5, 11.935],
                                 [2,      +4.5,     +1]]
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__()
```

Результати виконання програми:

```
/home/yan/PycharmProjects/Optimisation&PlanningLab5/venv/bin/python
  /home/yan/PycharmProjects/Optimisation&PlanningLab5/main.py
[-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 = 0.13698630136986298; Gt = 0.3346; f1 = 2; f2 = 15; q = 0.05
Gp < Gt => дисперсії рівномірні - все правильно
[202, 0, 0, 0, 0, 0, -1, 0, 3, -1, 0]

Перевірка значимості коефіцієнтів регресії за критерієм Стьюдента: m = 3, N = 15 для таблиці y_table та
  нормалізованих факторів
Оцінки коефіцієнтів βs: 202.538, 0.45, -0.274, 0.761, 0.583, 0.417, -1.583, -0.0, 3.39, -1.465, -0.562
Коефіцієнти ts:         413.15, 0.92, 0.56, 1.55, 1.19, 0.85, 3.23, 0.00, 6.91, 2.99, 1.15
f3 = 30; q = 0.05; tтабл = 2.0423
β0 важливий; β1 неважливий; β2 неважливий; β3 неважливий; β12 неважливий; β13 неважливий; β23 важливий; β123
  неважливий; β11 важливий; β22 важливий; β33 неважливий
Рівняння регресії без незначимих членів: y = +202.54 -1.58x23 +3.39x1^2 -1.46x2^2

Перевірка адекватності моделі за критерієм Фішера: m = 3, N = 15 для таблиці y_table
Теоретичні значення y для різних комбінацій факторів:
x1 = 0, x2 = -8, x3 = 0: y = -701.3737717931701
x1 = 9, x2 = 10, x3 = -27: y = -52.0460650035574
x1 = 0, x2 = 10, x3 = 0: y = 1271.2682373753403
x1 = 9, x2 = -8, x3 = 63: y = 2396.575144312925
x1 = 0, x2 = 10, x3 = 0: y = -754.1133718671568
x1 = 9, x2 = -8, x3 = -27: y = -768.8064649295684
x1 = 0, x2 = -8, x3 = 0: y = 1324.007837449327
x1 = 9, x2 = 10, x3 = 63: y = 548.3355442389441
x1 = 4.5, x2 = 1, x3 = -18.337500000000002: y = -729.1317929413616
x1 = 4.5, x2 = 1, x3 = 36.3375: y = 1645.1381122882726
x1 = -0.9675, x2 = 1, x3 = -1.935: y = 409.8480641089472
x1 = 9.9675, x2 = 1, x3 = 19.935: y = 708.8198419482222
x1 = 4.5, x2 = -9.935, x3 = 9.0: y = 470.6912981252101
x1 = 4.5, x2 = 11.935, x3 = 9.0: y = 94.96518403531712
x1 = 4.5, x2 = 1, x3 = 9.0: y = 458.0031596734556
Fp = 339376.66163822147, Ft = 2.1256
Fp > Ft => модель неадекватна

Process finished with exit code 0
```