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

Лабораторна робота №6

З дисципліни «Методи оптимізації та планування»

Тема: Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами

ВИКОНАВ:
Студент ІІ курсу ФІОТ
Групи ІО-92
Рожко М.М.

ПЕРЕВІРИВ:
асистент
Регіда П.Г.

Київ 2021 р.

Мета:

Провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.

Завдання до лабораторної роботи:

1. Ознайомитися з теоретичними відомостями.
2. Вибрати з таблиці варіантів і записати в протокол інтервали значень x_1, x_2, x_3. Обчислити і записати значення, відповідні кодованим значенням факторів +1; -1;+ l; - l; 0 для $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$.
3. Значення функції відгуку знайти за допомогою підстановки в формулу:

$$y_i = f(x_1, x_2, x_3) + \text{random}(10)-5,$$

де $f(x_1, x_2, x_3)$ вибирається по номеру в списку в журналі викладача.

4. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках використовувати натуральні значення факторів.
5. Зробити висновки по виконаній роботі.

Варіант завдання:

N	X1		X2		X3		f(x1,x2,x4)
	min	max	min	max	min	max	
217	20	70	5	40	20	45	3,1+6,3*x1+9,8*x2+5,5*x3+2,5*x1*x1+0,4*x2*x2+1,0*x3*x3+3,5*x1*x2+0,7*x1*x3+7,9*x2*x3+8,7*x1*x2*x3

Розруківка коду програми:

```
from _decimal import Decimal
from scipy.stats import f, t
from itertools import compress
from functools import reduce
import numpy
import math
import random

xmin = [20, 5, 20]
xmax = [70, 40, 45]

norm_plan_raw = [[-1, -1, -1],
                 [-1, +1, +1],
                 [+1, -1, +1],
                 [+1, +1, -1],
                 [-1, -1, +1],
                 [-1, +1, -1],
                 [+1, -1, -1],
                 [+1, +1, +1],
                 [-1.73, 0, 0],
                 [+1.73, 0, 0],
                 [0, -1.73, 0],
                 [0, +1.73, 0],
                 [0, 0, -1.73],
                 [0, 0, +1.73]]

x0 = [(xmax[_] + xmin[_])/2 for _ in range(3)]
dx = [xmax[_] - x0[_] for _ in range(3)]

natur_plan_raw = [[xmin[0],           xmin[1],           xmin[2]],
                  [xmin[0],           xmin[1],           xmax[2]],
                  [xmin[0],           xmax[1],           xmin[2]],
                  [xmin[0],           xmax[1],           xmax[2]],
                  [xmax[0],           xmin[1],           xmin[2]],
                  [xmax[0],           xmin[1],           xmax[2]],
                  [xmax[0],           xmax[1],           xmin[2]],
                  [xmax[0],           xmax[1],           xmax[2]],
                  [-1.73*dx[0]+x0[0], x0[1],             x0[2]],
                  [1.73*dx[0]+x0[0],  x0[1],             x0[2]],
                  [x0[0],             -1.73*dx[1]+x0[1], x0[2]],
                  [x0[0],             1.73*dx[1]+x0[1],  x0[2]],
                  [x0[0],             x0[1],             -1.73*dx[2]+x0[2]],
                  [x0[0],             x0[1],             1.73*dx[2]+x0[2]],
                  [x0[0],             x0[1],             x0[2]]]


def equation_of_regression(x1, x2, x3, cef, importance=[] * 11):
    factors_array = [1, x1, x2, x3, x1 * x2, x1 * x3, x2 * x3, x1 * x2 * x3,
x1 ** 2, x2 ** 2, x3 ** 2]
    return sum([el[0] * el[1] for el in compress(zip(cef, factors_array),
importance)])


def func(x1, x2, x3):
    coeffs = [3.1, 6.3, 9.8, 5.5, 2.5, 0.4, 1, 3.5, 0.7, 7.9, 8.7]
    return equation_of_regression(x1, x2, x3, coeffs)


def generate_factors_table(raw_array):
    raw_list = [row + [row[0] * row[1], row[0] * row[2], row[1] * row[2],
row[0] * row[1] * row[2]] + list(
```

```
        map(lambda x: x ** 2, row)) for row in raw_array]
    return list(map(lambda row: list(map(lambda el: round(el, 3), row)),
raw_list))


def generate_y(m, factors_table):
    return [[round(func(row[0], row[1], row[2]) + random.randint(-5, 5), 3)
for _ in range(m)] for row in factors_table]


def print_matrix(m, N, factors, y_vals, additional_text=":"):
    labels_table = list(map(lambda x: x.ljust(10),
                            ["x1", "x2", "x3", "x12", "x13", "x23", "x123",
"x1^2", "x2^2", "x3^2"] + [
                                "y{}".format(i + 1) for i in range(m)]))
    rows_table = [list(factors[i]) + list(y_vals[i]) for i in range(N)]
    print("\nМатриця планування" + additional_text)
    print(" ".join(labels_table))
    print("\n".join([" ".join(map(lambda j: "{:<+10}".format(j),
rows_table[i])) for i in range(len(rows_table))]))
    print("\t")


def print_equation(coeffs, importance=[True] * 11):
    x_i_names = list(compress(["", "x1", "x2", "x3", "x12", "x13", "x23",
"x123", "x1^2", "x2^2", "x3^2"], importance))
    coefficients_to_print = list(compress(coeffs, importance))
    equation = " ".join(
        ["".join(i) for i in zip(list(map(lambda x: "{:+.2f}".format(x),
coefficients_to_print)), x_i_names)])
    print("Рівняння регресії: y = " + equation)


def set_factors_table(factors_table):
    def x_i(i):
        with_null_factor = list(map(lambda x: [1] + x,
generate_factors_table(factors_table)))
        res = [row[i] for row in with_null_factor]
        return numpy.array(res)

    return x_i


def m_ij(*arrays):
    return numpy.average(reduce(lambda accum, el: accum * el, list(map(lambda
el: numpy.array(el), arrays))))


def find_coefficients(factors, y_vals):
    x_i = set_factors_table(factors)
    coeffs = [[m_ij(x_i(column), x_i(row)) for column in range(11)] for row
in range(11)]
    y_numpy = list(map(lambda row: numpy.average(row), y_vals))
    free_values = [m_ij(y_numpy, x_i(i)) for i in range(11)]
    beta_coefficients = numpy.linalg.solve(coeffs, free_values)
    return list(beta_coefficients)


def cochran_criteria(m, N, y_table):
    def get_cochran_value(f1, f2, q):
        partResult1 = q / f2
        params = [partResult1, f1, (f2 - 1) * f1]
        fisher = f.isf(*params)
        result = fisher / (fisher + (f2 - 1))
```

```
        return Decimal(result).quantize(Decimal('.0001')).__float__()

    print("\nПеревірка за критерієм Кохрена: m = {}, N = {}".format(m, N))
    y_variations = [numpy.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 => змінюємо значення m")
        return False


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

    print("\nПеревірка за критерієм Стьюдента: m = {}, N = {} ".format(m, N))
    average_variation = numpy.average(list(map(numpy.var, y_table)))
    variation_beta_s = average_variation / N / m
    standard_deviation_beta_s = math.sqrt(variation_beta_s)
    t_i = [abs(beta_coefficients[i]) / standard_deviation_beta_s for i in
range(len(beta_coefficients))]
    f3 = (m - 1) * N
    q = 0.05
    t_our = get_student_value(f3, q)
    importance = [True if el > t_our else False for el in list(t_i)]

    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_our))
    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))
    print(*to_print, sep="; ")
    print_equation(beta_coefficients, importance)
    return importance


def fisher_criteria(m, N, d, x_table, y_table, b_coefficients, importance):
    def get_fisher_value(f3, f4, q):
        return Decimal(abs(f.isf(q, f4,
f3))).quantize(Decimal('.0001')).__float__()

    f3 = (m - 1) * N
    f4 = N - d
    q = 0.05
    theoretical_y = numpy.array([equation_of_regression(row[0], row[1],
row[2], b_coefficients) for row in x_table])
    average_y = numpy.array(list(map(lambda el: numpy.average(el), y_table)))
```

```
    s_ad = m / (N - d) * sum((theoretical_y - average_y) ** 2)
    y_variations = numpy.array(list(map(numpy.var, y_table)))
    s_v = numpy.average(y_variations)
    f_p = float(s_ad / s_v)
    f_t = get_fisher_value(f3, f4, q)
    theoretical_values_to_print = list(
        zip(map(lambda x: "x1 = {0[1]:<10} x2 = {0[2]:<10} x3 =
{0[3]:<10}".format(x), x_table), theoretical_y))
    print("\nПеревірка 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


m = 3
N = 15
natural_plan = generate_factors_table(natur_plan_raw)
y_arr = generate_y(m, natur_plan_raw)
while not cochran_criteria(m, N, y_arr):
    m += 1
    y_arr = generate_y(m, natural_plan)

print_matrix(m, N, natural_plan, y_arr, " для натуралізованих факторів:")
coefficients = find_coefficients(natural_plan, y_arr)
print_equation(coefficients)
importance = student_criteria(m, N, y_arr, coefficients)
d = len(list(filter(None, importance)))
fisher_criteria(m, N, d, natural_plan, y_arr, coefficients, importance)
```

Результати роботи програми:

```
Перевірка за критерієм Кохрена: m = 3, N = 15
Gp = 0.15168539325842695; Gt = 0.3346; f1 = 2; f2 = 15; q = 0.05
Gp < Gt => дисперсії рівномірні => все правильно

Матриця планування для натуралізованих факторів:
x1         x2         x3         x12        x13        x23        x123       x1^2       x2^2       x3^2       y1         y2         y3
+20        +5         +20        +100       +400       +100       +2000      +400       +25        +400       -4         -2         -4
+20        +5         +45        +100       +900       +225       +4500      +400       +25        +2025      +4         +2         -4
+20        +40        +20        +800       +400       +800       +16000     +400       +1600      +400       -5         -3         -5
+20        +40        +45        +800       +900       +1800      +36000     +400       +1600      +2025      +2         +1         -1
+70        +5         +20        +350       +1400      +100       +7000      +4900      +25        +400       -4         +1         +1
+70        +5         +45        +350       +3150      +225       +15750     +4900      +25        +2025      -3         +5         -3
+70        +40        +20        +2800      +1400      +800       +56000     +4900      +1600      +400       -2         +5         +0
+70        +40        +45        +2800      +3150      +1800      +126000    +4900      +1600      +2025      +2         -2         -3
+1.75      +22.5      +32.5      +39.375    +56.875    +731.25    +1279.688  +3.062     +506.25    +1056.25   +5         +3         -1
+88.25     +22.5      +32.5      +1985.625  +2868.125  +731.25    +64532.812 +7788.062  +506.25    +1056.25   -2         -2         +2
+45.0      -7.775     +32.5      -349.875   +1462.5    -252.687   -11370.937 +2025.0    +60.451    +1056.25   -4         -1         -1
+45.0      +52.775    +32.5      +2374.875  +1462.5    +1715.188  +77183.438 +2025.0    +2785.201  +1056.25   +3         -4         +4
+45.0      +22.5      +10.875    +1012.5    +489.375   +244.688   +11010.938 +2025.0    +506.25    +118.266   +5         +5         -4
+45.0      +22.5      +54.125    +1012.5    +2435.625  +1217.812  +54801.562 +2025.0    +506.25    +2929.516  -1         +5         -4
+45.0      +22.5      +32.5      +1012.5    +1462.5    +731.25    +32906.25  +2025.0    +506.25    +1056.25   -5         -5         +3

Рівняння регресії: y = +1.64 -0.04x1 -0.14x2 -0.20x3 +0.00x12 -0.00x13 +0.00x23 -0.00x123 +0.00x1^2 +0.00x2^2 +0.01x3^2

Перевірка за критерієм Стьюдента: m = 3, N = 15
Оцінки коефіцієнтів βs: 1.635, -0.044, -0.142, -0.199, 0.003, -0.003, 0.003, -0.0, 0.001, 0.001, 0.005
Коефіцієнти ts: 3.90, 0.10, 0.34, 0.47, 0.01, 0.01, 0.01, 0.00, 0.00, 0.00, 0.01
f3 = 30; q = 0.05; tтабл = 2.0423
β0 важливий; β1 неважливий; β2 неважливий; β3 неважливий; β12 неважливий; β13 неважливий; β23 неважливий; β123 неважливий; β11 неважливий; β22 неважливий; β33 неважливий
Рівняння регресії: y = +1.64
```

```
Перевірка за критерієм Фішера: m = 3, N = 15 для таблиці y_table
Теоретичні значення Y для різних комбінацій факторів:
x1 = 5          x2 = 20         x3 = 100        : y = 0
x1 = 5          x2 = 45         x3 = 100        : y = 0
x1 = 40         x2 = 20         x3 = 800        : y = 0
x1 = 40         x2 = 45         x3 = 800        : y = 0
x1 = 5          x2 = 20         x3 = 350        : y = 0
x1 = 5          x2 = 45         x3 = 350        : y = 0
x1 = 40         x2 = 20         x3 = 2800       : y = 0
x1 = 40         x2 = 45         x3 = 2800       : y = 0
x1 = 22.5       x2 = 32.5       x3 = 39.375     : y = 0
x1 = 22.5       x2 = 32.5       x3 = 1985.625   : y = 0
x1 = -7.775     x2 = 32.5       x3 = -349.875   : y = 0
x1 = 52.775     x2 = 32.5       x3 = 2374.875   : y = 0
x1 = 22.5       x2 = 10.875     x3 = 1012.5     : y = 0
x1 = 22.5       x2 = 54.125     x3 = 1012.5     : y = 0
x1 = 22.5       x2 = 32.5       x3 = 1012.5     : y = 0
Fp = 1.4536516853932577, Ft = 2.0374
Fp < Ft => модель адекватна

Process finished with exit code 0
```

Висновок:

Успішно проведено трьохфакторний експеримент і отримано адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план. Написана відповідна програма. Результати виконання програми надані у звіті.