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

Методи наукових досліджень
Лабораторна робота №4

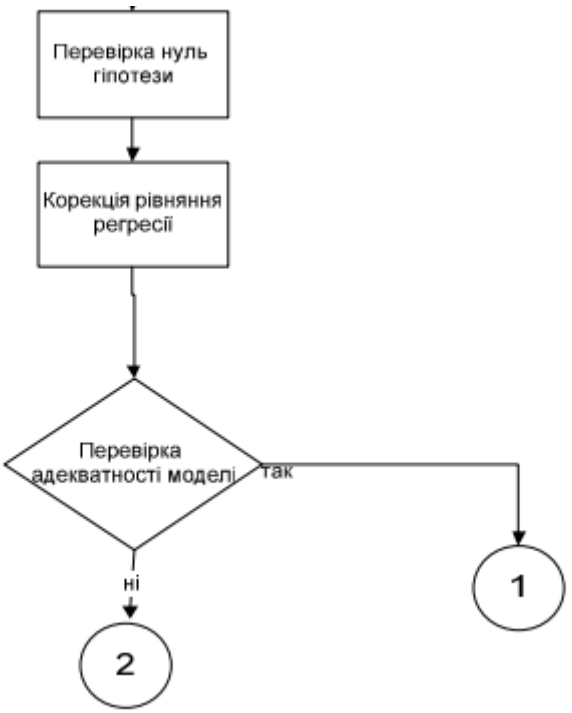
«Проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням ефекту взаємодії»

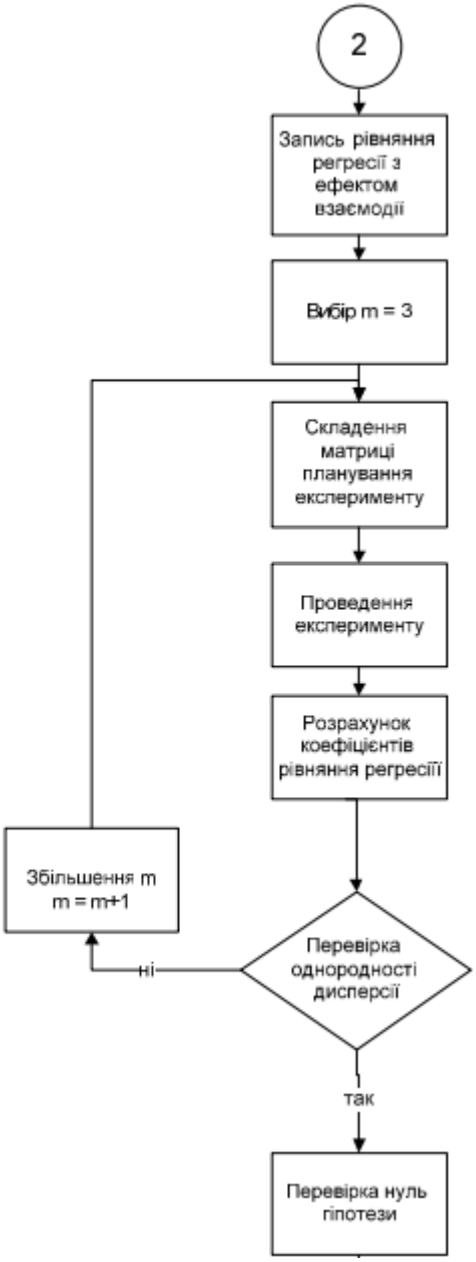
Виконала:
студентка групи ІВ-93
Стефанович І. В.
Варіант: 23
Перевірив:
Регіда П.Г

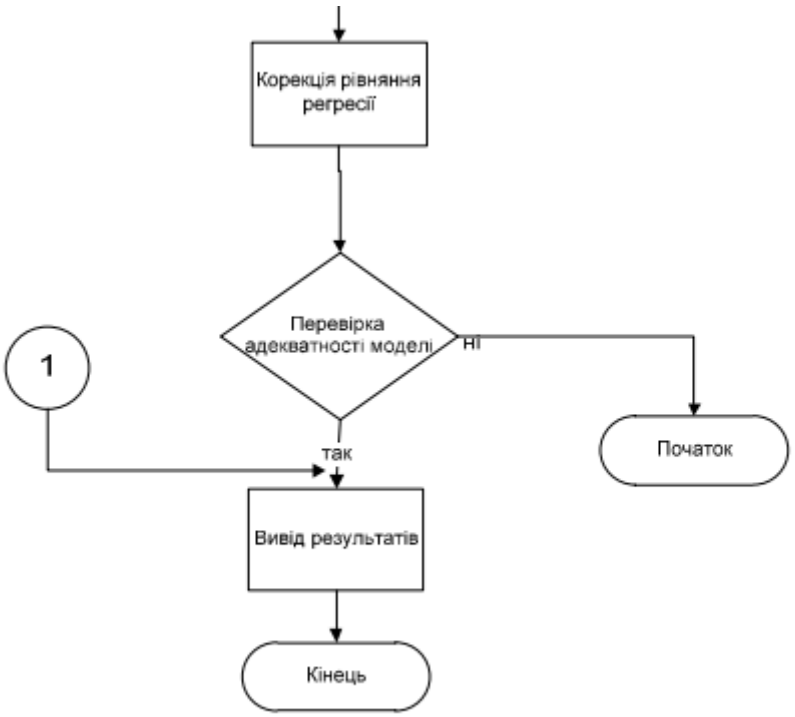
Київ - 2021 р.

Мета: провести повний трьохфакторний експеримент. Знайти рівняння регресії адекватне об'єкту.

Короткі теоретичні відомості:


Перевірка нуль гіпотези
Корекція рівняння регресії
Перевірка адекватності моделі
так
ні
1
2


2
Запись рівняння регресії з ефектом взаємодії
Вибір m = 3
Складення матриці планування експерименту
Проведення експерименту
Розрахунок коефіцієнтів рівняння регресіїї
Збільшення m
m = m+1
Перевірка однородності дисперсії
ні
так
Перевірка нуль гіпотези



Індивідуальне завдання:

323	10	40	30	80	10	20

Лістинг коду програми:

```
from prettytable import PrettyTable
from itertools import compress
from scipy.stats import f, t
from functools import reduce
from random import randint
import numpy as np
import math


class Lab4:
    def __init__(self, n, m):
        self.x_min = [10, 30, 10]
        self.x_max = [40, 80, 20]
```

```
        self.n = n
        self.m = m

        self.f1 = self.m - 1
        self.f2 = self.n
        self.f3 = self.f1 * self.f2

        self.p = 0.95
        self.q = 1 - self.p

        self.N = [i + 1 for i in range(self.n + 1)]

        self.average_x_min = round(np.average(self.x_min))
        self.average_x_max = round(np.average(self.x_max))

        self.y_min = 200 + self.average_x_min
        self.y_max = 200 + self.average_x_max

        self.important_coef = str()

        self.norm_x = [[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]]

        self.x12 = [self.norm_x[1][i] * self.norm_x[2][i] for i in range(self.n)]
        self.x13 = [self.norm_x[1][i] * self.norm_x[3][i] for i in range(self.n)]
        self.x23 = [self.norm_x[2][i] * self.norm_x[3][i] for i in range(self.n)]
            self.x123 = [self.norm_x[1][i] * self.norm_x[2][i] * self.norm_x[3][i] for i in
range(self.n)]

        self.norm_x += [self.x12, self.x13, self.x23, self.x123]
        self.norm_xt = np.array(self.norm_x)
        self.norm_xt = self.norm_xt.transpose()

        self.factors_x = [[-20, -20, 30, 30, -20, -20, 30, 30],
                          [30, 80, 30, 80, 30, 80, 30, 80],
```

```
                    [30, 45, 45, 30, 45, 30, 30, 45]]

        self.xf12 = [self.factors_x[0][i] * self.factors_x[1][i] for i in range(self.n)]
        self.xf13 = [self.factors_x[0][i] * self.factors_x[2][i] for i in range(self.n)]
        self.xf23 = [self.factors_x[1][i] * self.factors_x[2][i] for i in range(self.n)]
            self.xf123 = [self.factors_x[0][i] * self.factors_x[1][i] * self.factors_x[2][i] for i in
range(self.n)]

        self.factors_x += [self.xf12, self.xf13, self.xf23, self.xf123]
        self.factors_xt = np.array(self.factors_x)
        self.factors_xt = self.factors_xt.transpose()

          self.y_t = np.array([[randint((self.y_min), (self.y_max)) for i in range(self.n)] for j in
range(self.m)])
        self.y = self.y_t.transpose()

        self.av_y = [round(sum(i) / len(i), 2) for i in self.y]

        self.S2 = [round(np.var(i), 2) for i in self.y]

        self.x1 = np.array(list(zip(*self.factors_xt))[0])
        self.x2 = np.array(list(zip(*self.factors_xt))[1])
        self.x3 = np.array(list(zip(*self.factors_xt))[2])

        self.natural_bi = Lab4.naturalized_b(self.n, self.x1, self.x2, self.x3, self.av_y)

           print("ŷ = b0 + b1*x1 + b2*x2 + b3*x3 + b12*x1*x2 + b13*x1*x3 + b23*x2*x3 +
b123*x1*x2*x3")

        th = ["N", "x0", "x1", "x2", "x3", "x1*x2", "x1*x3", "x2*x3", "x1*x2*x3"]
        th += ["y" + str(i + 1) for i in range(self.m)]
        th.append("<y>")
        th.append("S^2")
        columns = len(th)
        print('\n')
        table = PrettyTable(th)
        table.title = "Нормована матриця планування експерименту"
```

```
        for i in range(self.n):
                        td = [self.N[i], self.norm_x[0][i], self.norm_x[1][i], self.norm_x[2][i],
self.norm_x[3][i], \
                self.x12[i], self.x13[i], self.x23[i], self.x123[i]]
            td += [self.y_t[j][i] for j in range(self.m)]
            td.append(self.av_y[i])
            td.append(self.S2[i])
            td_data = td[:]
            while td_data:
                table.add_row(td_data[:columns])
                td_data = td_data[columns:]
        print(table)

        th = ["N", "x1", "x2", "x3", "x1*x2", "x1*x3", "x2*x3", "x1*x2*x3"]
        th += ["y" + str(i + 1) for i in range(self.m)]
        th.append("<y>")
        th.append("S^2")
        columns = len(th)
        print('\n')
        table = PrettyTable(th)
        table.title = "Матриця планування експерименту"
        for i in range(self.n):

td = [self.N[i], self.factors_x[0][i], self.factors_x[1][i], self.factors_x[2][i], \
                self.xf12[i], self.xf13[i], self.xf23[i], self.xf123[i]]
            td += [self.y_t[j][i] for j in range(self.m)]
            td.append(self.av_y[i])
            td.append(self.S2[i])
            td_data = td[:]
            while td_data:
                table.add_row(td_data[:columns])
                td_data = td_data[columns:]
        print(table)

        self.__kohren_criteria(self.m, self.n, self.y, self.p, self.q, self.f1, self.f2)

        self.__student_criteria(self.m, self.n, self.y, self.av_y, self.norm_xt, self.f3, self.q)
```

```
            self.__fisher_criteria(self.m, self.n, 1, self.f3, self.q, self.factors_xt, self.y, self.natural_bi, self.y_x)

    @staticmethod
    def naturalized_b(n, x1, x2, x3, av_y):
        def m_ij(*arrays):
            return np.average(reduce(lambda accum, el: accum * el, arrays))

        coefficient = [[n, m_ij(x1), m_ij(x2), m_ij(x3), m_ij(x1 * x2), m_ij(x1 * x3), m_ij(x2 * x3), m_ij(x1 * x2 * x3)],
                       [m_ij(x1), m_ij(x1 ** 2), m_ij(x1 * x2), m_ij(x1 * x3), m_ij(x1 ** 2 * x2), m_ij(x1 ** 2 * x3),
                        m_ij(x1 * x2 * x3), m_ij(x1 ** 2 * x2 * x3)],
                       [m_ij(x2), m_ij(x1 * x2), m_ij(x2 ** 2), m_ij(x2 * x3), m_ij(x1 * x2 ** 2), m_ij(x1 * x2 * x3),
                        m_ij(x2 ** 2 * x3), m_ij(x1 * x2 ** 2 * x3)],
                       [m_ij(x3), m_ij(x1 * x3), m_ij(x2 * x3), m_ij(x3 ** 2), m_ij(x1 * x2 * x3), m_ij(x1 * x3 ** 2),
                        m_ij(x2 * x3 ** 2), m_ij(x1 * x2 * x3 ** 2)],
                       [m_ij(x1 * x2), m_ij(x1 ** 2 * x2), m_ij(x1 * x2 ** 2), m_ij(x1 * x2 * x3), m_ij(x1 ** 2 * x2 ** 2),
                        m_ij(x1 ** 2 * x2 * x3), m_ij(x1 * x2 ** 2 * x3), m_ij(x1 ** 2 * x2 ** 2 * x3)],
                       [m_ij(x1 * x3), m_ij(x1 ** 2 * x3), m_ij(x1 * x2 * x3), m_ij(x1 * x3 ** 2), m_ij(x1 ** 2 * x2 * x3),
                        m_ij(x1 ** 2 * x3 ** 2), m_ij(x1 * x2 * x3 ** 2), m_ij(x1 ** 2 * x2 * x3 ** 2)],
                       [m_ij(x2 * x3), m_ij(x1 * x2 * x3), m_ij(x2 ** 2 * x3), m_ij(x2 * x3 ** 2), m_ij(x1 * x2 ** 2 * x3),
                        m_ij(x1 * x2 * x3 ** 2), m_ij(x2 ** 2 * x3 ** 2), m_ij(x1 * x2 ** 2 * x3 ** 2)],
                       [m_ij(x1 * x2 * x3), m_ij(x1 ** 2 * x2 * x3), m_ij(x1 * x2 ** 2 * x3), m_ij(x1 * x2 * x3 ** 2),
                        m_ij(x1 ** 2 * x2 ** 2 * x3), m_ij(x1 ** 2 * x2 * x3 ** 2), m_ij(x1 * x2 ** 2 * x3 ** 2),
                        m_ij(x1 ** 2 * x2 ** 2 * x3 ** 2)]]

        free_vector = [m_ij(av_y), m_ij(av_y * x1), m_ij(av_y * x2), m_ij(av_y * x3), m_ij(av_y * x1 * x2),
```

```
                m_ij(av_y * x1 * x3), m_ij(av_y * x2 * x3), m_ij(av_y * x1 * x2 * x3)]
    natural_bi = np.linalg.solve(coefficient, free_vector)
    return natural_bi

def __kohren_criteria(self, m, n, y, p, q, f1, f2):
    S = [np.var(i) for i in y]
    Gp = max(S) / sum(S)
    q_ = q / f2
    khr = f.ppf(q=1 - q_, dfn=f1, dfd=(f2 - 1) * f1)
    Gt = khr / (khr + f2 - 1)
    print("\nКритерій Кохрена: Gr = " + str(round(Gp, 3)))
    if Gp < Gt:
        print("Дисперсії однорідні з вірогідністю 95%.")
        pass
    else:
        print("\nДисперсії не однорідні.\nПроводимо експеремент для m+=1\n")
        Lab4(self.n, self.m + 1)

def __student_criteria(self, m, n, y, av_y, norm_xt, f3, q):
    av_S = np.average(list(map(np.var, y)))
    s2_beta = av_S / n / m
    s_beta = math.sqrt(s2_beta)
    xi = np.array([[el[i] for el in norm_xt] for i in range(len(norm_xt))])
    k_beta = np.array([round(np.average(av_y * xi[i]), 3) for i in range(len(xi))])

    T = np.array([abs(k_beta[i]) / s_beta for i in range(len(k_beta))])

    T_tabl = t.ppf(q=1 - q, df=f3)

    print("\nКритерій Стьюдента:")
    T_ = list(map(lambda i: "{:.2f}".format(i), T))
    for i in T_: print(str(i))
    imp = [i if i > T_tabl else 0 for i in T]
    index_list = []
    b = ["b0", "b1", "b2", "b3", "b4", "b12", "b13", "b23", "b123"]
    index_list = [i f
```

```
or i, x in enumerate(imp) if x == 0]
        index_list = [b[i] for i in index_list]
        deleted_coef = ', '.join(
                index_list) + " - коефіцієнти рівняння регресії приймаємо незначними, виключаємо їх з рівняння.\n"
        print(deleted_coef)

        imp2 = [i if i < T_tabl else 0 for i in T]
        index_list2 = [i for i, x in enumerate(imp2) if x == 0]
        index_list2 = [b[i] for i in index_list2]
            self.important_coef = ', '.join(index_list2) + " - значемі коефіцієнти рівняння регресії."

        self.y_x = [True if i > T_tabl else False for i in T]
            x_i = list(compress(["", "*x1", "*x2", "*x3", "*x12", "*x13", "*x23", "*x123"], self.y_x))
        p = list(compress(k_beta, self.y_x))
        y = " ".join(["".join(i) for i in zip(list(map(lambda x: "{:+.2f}".format(x), p)), x_i)])
        print("Рівняння регресії: y = " + y)

    def __fisher_criteria(self, m, n, d, f3, q, natural_x, y_t, b_k, imp):
        f4 = n - d
        b_k = list(compress(b_k, imp))
        y_val = np.array([sum(map(lambda x, b: x * b, row, b_k)) for row in natural_x])
        y_averages = np.array(list(map(np.average, y_t)))
        s_ad = m / (n - d) * (sum((y_val - y_averages) ** 2)) * 0.001
        y_variations = np.array(list(map(np.var, y_t)))
        s_v = np.average(y_variations)

        fp = s_ad / s_v
        print("\nКритерій Фішера: Fp = " + str(round(fp, 4)))

        ft = f.isf(q, f4, f3)
        print("Табличне значення критерія Фішера: Ft = " + str(round(ft, 4)))

        if fp < ft:
            print("\nРівняння регресії адекватно оригіналу.")
```

```
            pass
        else:
            print("\nРівняння регресії НЕ адекватно оригіналу. >>> m+=1\n")
            Lab4(self.n, self.m + 1)


Lab4(8, 3)
```

Результати роботи програми:

```
ŷ = b0 + b1*x1 + b2*x2 + b3*x3 + b12*x1*x2 + b13*x1*x3 + b23*x2*x3 + b123*x1*x2*x3


+-----------------------------------------------------------------------------------------------+
|                          Нормована матриця планування експерименту                            |
+---+----+----+----+----+-------+-------+-------+----------+-----+-----+-----+--------+--------+
| N | x0 | x1 | x2 | x3 | x1*x2 | x1*x3 | x2*x3 | x1*x2*x3 |  y1 |  y2 |  y3 |  <y>   |  S^2   |
+---+----+----+----+----+-------+-------+-------+----------+-----+-----+-----+--------+--------+
| 1 | 1  | -1 | -1 | -1 |   1   |   1   |   1   |    -1    | 241 | 235 | 234 | 236.67 |  9.56  |
| 2 | 1  | -1 | 1  | 1  |   -1  |   -1  |   1   |    -1    | 232 | 246 | 247 | 241.67 | 46.89  |
| 3 | 1  | 1  | -1 | 1  |   -1  |   1   |   -1  |    -1    | 229 | 217 | 227 | 224.33 | 27.56  |
| 4 | 1  | 1  | 1  | -1 |   1   |   -1  |   -1  |    -1    | 243 | 217 | 234 | 231.33 | 116.22 |
| 5 | 1  | -1 | -1 | 1  |   1   |   -1  |   -1  |    1     | 228 | 245 | 234 | 235.67 | 49.56  |
| 6 | 1  | -1 | 1  | -1 |   -1  |   1   |   -1  |    1     | 227 | 232 | 231 | 230.0  |  4.67  |
| 7 | 1  | 1  | -1 | -1 |   -1  |   -1  |   1   |    1     | 226 | 235 | 237 | 232.67 | 22.89  |
| 8 | 1  | 1  | 1  | 1  |   1   |   1   |   1   |    1     | 232 | 236 | 236 | 234.67 |  3.56  |
+---+----+----+----+----+-------+-------+-------+----------+-----+-----+-----+--------+--------+


+-----------------------------------------------------------------------------------------+
|                          Матриця планування експерименту                                |
+---+-----+----+----+-------+-------+-------+----------+-----+-----+-----+--------+--------+
| N |  x1 | x2 | x3 | x1*x2 | x1*x3 | x2*x3 | x1*x2*x3 |  y1 |  y2 |  y3 |  <y>   |  S^2   |
+---+-----+----+----+-------+-------+-------+----------+-----+-----+-----+--------+--------+
| 1 | -20 | 30 | 30 |  -600 |  -600 |  900  |  -18000  | 241 | 235 | 234 | 236.67 |  9.56  |
| 2 | -20 | 80 | 45 | -1600 |  -900 |  3600 |  -72000  | 232 | 246 | 247 | 241.67 | 46.89  |
| 3 |  30 | 30 | 45 |  900  |  1350 |  1350 |  40500   | 229 | 217 | 227 | 224.33 | 27.56  |
| 4 |  30 | 80 | 30 |  2400 |  900  |  2400 |  72000   | 243 | 217 | 234 | 231.33 | 116.22 |
| 5 | -20 | 30 | 45 |  -600 |  -900 |  1350 |  -27000  | 228 | 245 | 234 | 235.67 | 49.56  |
| 6 | -20 | 80 | 30 | -1600 |  -600 |  2400 |  -48000  | 227 | 232 | 231 | 230.0  |  4.67  |
| 7 |  30 | 30 | 30 |  900  |  900  |  900  |  27000   | 226 | 235 | 237 | 232.67 | 22.89  |
| 8 |  30 | 80 | 45 |  2400 |  1350 |  3600 |  108000  | 232 | 236 | 236 | 234.67 |  3.56  |
+---+-----+----+----+-------+-------+-------+----------+-----+-----+-----+--------+--------+

Критерій Кохрена: Gr = 0.414
Дисперсії однорідні з вірогідністю 95%.

Критерій Стьюдента:
192.95
2.17
0.86
```

```
0.86
0.59
1.00
1.62
2.52
0.10
b2, b3, b4, b12, b23 – коефіцієнти рівняння регресії приймаємо незначними, виключаємо їх з рівняння.

Рівняння регресії: y = +233.38 -2.63*x1 +3.04*x23

Критерій Фішера: Fp = 1.6422
Табличне значення критерія Фішера: Ft = 2.6572

Рівняння регресії адекватно оригіналу.
```

Висновки: під час виконання лабораторної роботи було змодельовано трьохфакторний експеримент з використанням лінійного рівняння регресії, сформовано матрицю планування експерименту, визначено коефіцієнти рівняння регресії, натуралізовані та нормовані, перевірено правильність розрахунку коефіцієнтів рівняння регресії. При виявленні неадекватності лінійного рівняння регресії оригіналу було застосовано ефект взаємодії факторів.