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

Методи наукових досліджень

Лабораторна робота №6

«Проведення трьохфакторного експерименту
при використанні рівняння регресії з квадратичними членами
(рототабельний композиційний план)»

Виконав:
студент 2 курсу, групи ІВ-91
Коренюк Андрій Олександрович
Залікова книжка № ІВ-9115
Варіант: 14

Перевірив: ас. Регіда П.Г.

Київ – 2021

Мета: провести трьохфакторний експеримент і отримати адекватну модель – рівняння регресії, використовуючи рототабельний композиційний план.

Завдання

№варіанта	x_1		x_2		x_3	
	min	**max**	**min**	**max**	**min**	**max**
114	-25	75	25	65	25	40
$7.7 + 2.8 \cdot x_1 + 0.5 \cdot x_2 + 2.6 \cdot x_3 + 5.0 \cdot x_1 \cdot x_2 + 0.3 \cdot x_1 \cdot x_3 + 9.3 \cdot x_2 \cdot x_3 + 4.1 \cdot x_1 \cdot x_2 \cdot x_3 + 1.4 \cdot x_1^2 + 0.3 \cdot x_2^2 + 7.1 \cdot x_3^2$						

Лістинг програми

configuration.py

```
"""Довірча ймовірність p = 0.95 (критерій значимості 0.05)"""
variant = {"n": 114, "x1min": -25, "x1max": 75, "x2min": 25, "x2max": 65, "x3min": 25, "x3max": 40}

x1_average = (variant["x1min"] + variant["x1max"]) / 2
x2_average = (variant["x2min"] + variant["x2max"]) / 2
x3_average = (variant["x3min"] + variant["x3max"]) / 2

del_x1 = variant["x1max"] - x1_average
del_x2 = variant["x2max"] - x2_average
del_x3 = variant["x3max"] - x3_average

x0 = [1, 1, 1, 1]
x1 = [-1, -1, 1, 1]
x2 = [-1, 1, -1, 1]
x3 = [1, -1, -1, 1]

nx0 = [1, 1, 1, 1]
nx1 = [variant["x1min"] if x1[i] == -1 else variant["x1max"] for i in range(4)]
nx2 = [variant["x2min"] if x2[i] == -1 else variant["x2max"] for i in range(4)]
nx3 = [variant["x3min"] if x3[i] == -1 else variant["x3max"] for i in range(4)]

sp_x0 = [1, 1, 1, 1]
sp_x1 = [-1, -1, 1, 1]
sp_x2 = [-1, 1, -1, 1]
sp_x3 = [-1, 1, 1, -1]

sp_nx0 = [1, 1, 1, 1]
sp_nx1 = [variant["x1min"] if sp_x1[i] == -1 else variant["x1max"] for i in range(4)]
sp_nx2 = [variant["x2min"] if sp_x2[i] == -1 else variant["x2max"] for i in range(4)]
sp_nx3 = [variant["x3min"] if sp_x3[i] == -1 else variant["x3max"] for i in range(4)]

tp_count = 6
tp_x0 = [1 for _ in range(tp_count)]
tp_x1 = [-1.73, 1.73, 0, 0, 0, 0]
tp_x2 = [0, 0, -1.73, 1.73, 0, 0]
tp_x3 = [0, 0, 0, 0, -1.73, 1.73]

tp_nx0 = [1 for _ in range(tp_count)]
tp_nx1 = [tp_x1[i] * del_x1 + x1_average for i in range(tp_count)]
tp_nx2 = [tp_x2[i] * del_x2 + x2_average for i in range(tp_count)]
tp_nx3 = [tp_x3[i] * del_x3 + x3_average for i in range(tp_count)]
```

lab_6.py

```
from experiment import Experiment
from linear_without_interaction import LinearWithoutInteractionModel
from linear_with_interaction import LinearWithInteractionModel
from square_central_orthogonal import SquareCentralOrthogonalModel
from configuration import *
from copy import deepcopy
import logs

x = list()
nx = list()


def get_factor_lines(x, N):
    lines = list()
    for i in range(N):
        lines.append([x[0][i], x[1][i], x[2][i], x[3][i]])
    return lines


def extend_view(step, view, N):
    if step == 2:
        for i in range(N):
            view[i].append(view[i][1] * view[i][2])
            view[i].append(view[i][1] * view[i][3])
            view[i].append(view[i][2] * view[i][3])
            view[i].append(view[i][1] * view[i][2] * view[i][3])
    elif step == 3:
        for i in range(N):
            view[i].append(view[i][1] * view[i][1])
            view[i].append(view[i][2] * view[i][2])
            view[i].append(view[i][3] * view[i][3])


def linear_model_without_interaction():
    global x
    global nx
    logs.comment(0, [])
    N, K, m = 4, 4, 2
    logs.comment(3, [N, K, m])

    x = [deepcopy(x0), deepcopy(x1), deepcopy(x2), deepcopy(x3)]
    nx = [deepcopy(nx0), deepcopy(nx1), deepcopy(nx2), deepcopy(nx3)]

    logs.comment(4, [])
    # Складання плану експерименту
    x_lines = get_factor_lines(x, N)
    nx_lines = get_factor_lines(nx, N)

    # Виконання експерименту
    experiment = Experiment(nx_lines, m, N)
    lm_without = LinearWithoutInteractionModel(K, N)
    experiment.do()

    # Перевірка критерія Кохрена
    experiment.check_kohren()
    logs.comment(13, [experiment.m, experiment.f1, experiment.f2, experiment.Gp])
    logs.show_plan(0, 0, nx_lines, experiment)
    logs.show_plan(1, 0, x_lines, experiment)

    # Пошук коефіцієнтів
```

```
    logs.comment(5, [])
    lm_without.find_nature_cfs(nx, experiment.y_average)
    logs.comment(7, [round(el, 4) for el in lm_without.A])

    logs.comment(6, [])
    lm_without.find_encoded_cfs(x, experiment.y_average)
    logs.comment(8, [round(el, 4) for el in lm_without.B])

    logs.show_natured_checking_matrix(6, 0, nx_lines, lm_without, experiment)
    logs.show_encoded_checking_matrix(7, 0, x_lines, lm_without, experiment)

    # Перевірка критерія Стьюдента
    experiment.check_student(lm_without.K, lm_without.A, lm_without.B)
    logs.comment(14, [experiment.f3, [round(el, 4) for el in experiment.t]])
    logs.comment(15, [round(el, 4) for el in lm_without.A])
    logs.comment(16, [round(el, 4) for el in lm_without.B])

    # Перевірка критерія Фішера
    is_suitable = experiment.check_fisher(lm_without, nx_lines)
    logs.comment(21, [experiment.f3, experiment.f4, experiment.Fp])

    # Передача даних
    if is_suitable:
        logs.comment(22, [])
        logs.comment(25, [])
        logs.show_natured_checking_matrix(6, 0, nx_lines, lm_without, experiment)
        logs.show_encoded_checking_matrix(7, 0, x_lines, lm_without, experiment)
        del x, nx
    else:
        logs.comment(23, [])
        logs.comment(24, [])

    # Видалення зайвого
    del lm_without, experiment
    return is_suitable


def linear_model_with_interaction():
    global x
    global nx
    logs.comment(1, [])
    N, K, m = 8, 8, 2
    logs.comment(3, [N, K, m])

    x[0].extend(sp_x0)
    x[1].extend(sp_x1)
    x[2].extend(sp_x2)
    x[3].extend(sp_x3)
    nx[0].extend(sp_nx0)
    nx[1].extend(sp_nx1)
    nx[2].extend(sp_nx2)
    nx[3].extend(sp_nx3)

    logs.comment(4, [])
    # Складання плану експерименту
    x_lines = get_factor_lines(x, N)
    nx_lines = get_factor_lines(nx, N)
    x_views = deepcopy(x_lines)
    nx_views = deepcopy(nx_lines)

    extend_view(2, x_views, N)
    extend_view(2, nx_views, N)
```

```
    # Виконання експерименту
    experiment = Experiment(nx_lines, m, N)
    lm_with = LinearWithInteractionModel(K, N)
    experiment.do()

    # Перевірка критерія Кохрена
    experiment.check_kohren()
    logs.comment(13, [experiment.m, experiment.f1, experiment.f2, experiment.Gp])
    logs.show_plan(2, 1, nx_views, experiment)
    logs.show_plan(3, 1, x_views, experiment)

    # Пошук коефіцієнтів
    logs.comment(5, [])
    lm_with.find_nature_cfs(nx, experiment.y_average)
    logs.comment(9, [round(el, 4) for el in lm_with.A])

    logs.comment(6, [])
    lm_with.find_encoded_cfs(x, experiment.y_average)
    logs.comment(10, [round(el, 4) for el in lm_with.B])

    logs.show_natured_checking_matrix(8, 1, nx_views, lm_with, experiment)
    logs.show_encoded_checking_matrix(9, 1, x_views, lm_with, experiment)

    # Перевірка критерія Стьюдента
    experiment.check_student(lm_with.K, lm_with.A, lm_with.B)
    logs.comment(14, [experiment.f3, [round(el, 4) for el in experiment.t]])
    logs.comment(17, [round(el, 4) for el in lm_with.A])
    logs.comment(18, [round(el, 4) for el in lm_with.B])

    # Перевірка критерія Фішера
    is_suitable = experiment.check_fisher(lm_with, nx_lines)
    logs.comment(21, [experiment.f3, experiment.f4, experiment.Fp])

    # Передача даних
    if is_suitable:
        logs.comment(22, [])
        logs.comment(25, [])
        logs.show_natured_checking_matrix(8, 1, nx_views, lm_with, experiment)
        logs.show_encoded_checking_matrix(9, 1, x_views, lm_with, experiment)
        del x, nx
    else:
        logs.comment(23, [])
        logs.comment(24, [])

    # Видалення зайвого
    del x_views, nx_views, lm_with, experiment
    return is_suitable


def square_central_orthogonal_model():
    global x
    global nx
    logs.comment(2, [])
    N, K, m = 14, 11, 2
    logs.comment(3, [N, K, m])

    x[0].extend(tp_x0)
    x[1].extend(tp_x1)
    x[2].extend(tp_x2)
    x[3].extend(tp_x3)
    nx[0].extend(tp_nx0)
```

```
nx[1].extend(tp_nx1)
nx[2].extend(tp_nx2)
nx[3].extend(tp_nx3)

logs.comment(4, [])
# Складання плану експерименту
x_lines = get_factor_lines(x, N)
nx_lines = get_factor_lines(nx, N)
x_views = deepcopy(x_lines)
nx_views = deepcopy(nx_lines)

extend_view(2, x_views, N)
extend_view(2, nx_views, N)
extend_view(3, x_views, N)
extend_view(3, nx_views, N)

# Виконання експерименту
experiment = Experiment(nx_lines, m, N)
sq_co = SquareCentralOrthogonalModel(K, N)
experiment.do()

# Перевірка критерія Кохрена
experiment.check_kohren()
logs.comment(13, [experiment.m, experiment.f1, experiment.f2, experiment.Gp])
logs.show_plan(4, 2, nx_views, experiment)
logs.show_plan(5, 2, x_views, experiment)

# Пошук коефіцієнтів
logs.comment(5, [])
sq_co.find_nature_cfs(experiment.m, nx, experiment.y)
logs.comment(11, [round(el, 4) for el in sq_co.A])

logs.comment(6, [])
sq_co.find_encoded_cfs(experiment.m, x, experiment.y)
logs.comment(12, [round(el, 4) for el in sq_co.B])

logs.show_natured_checking_matrix(10, 2, nx_views, sq_co, experiment)
logs.show_encoded_checking_matrix(11, 2, x_views, sq_co, experiment)
# Перевірка критерія Стьюдента
experiment.check_student(sq_co.K, sq_co.A, sq_co.B)
logs.comment(14, [experiment.f3, [round(el, 4) for el in experiment.t]])
logs.comment(19, [round(el, 4) for el in sq_co.A])
logs.comment(20, [round(el, 4) for el in sq_co.B])

# Перевірка критерія Фішера
is_suitable = experiment.check_fisher(sq_co, nx_lines)
logs.comment(21, [experiment.f3, experiment.f4, experiment.Fp])

# Передача даних
if is_suitable:
    logs.comment(22, [])
    logs.comment(25, [])
    logs.show_natured_checking_matrix(10, 2, nx_views, sq_co, experiment)
    logs.show_encoded_checking_matrix(11, 2, x_views, sq_co, experiment)
    del x, nx
else:
    logs.comment(23, [])
    logs.comment(24, [])

# Видалення зайвого
del x_views, nx_views, sq_co, experiment
return is_suitable
```

```
def main():
    while True:
        if linear_model_without_interaction():
            break
        elif linear_model_with_interaction():
            break
        elif square_central_orthogonal_model():
            break
        logs.comment(26, [])


if __name__ == "__main__":
    main()
```

experiment.py

```
from copy import deepcopy
from random import randint
from math import sqrt, inf

import criterion_tables as ct


class Experiment:
    m = 0
    N = 0
    x_lines = list()
    y = list()
    y_average = list()
    S2_dis = list()
    f1, f2, f3, f4 = 0, 0, 0, 0
    Gp = 0
    t = list()
    Fp = 0
    d = 0
    s2b = 0

    def __init__(self, x_lines, m, N):
        self.x_lines = deepcopy(x_lines)
        self.m = m
        self.N = N

    def __del__(self):
        del self.m, self.N, self.x_lines
        del self.y, self.y_average, self.S2_dis
        del self.f1, self.f2, self.f3, self.f4
        del self.Gp, self.t, self.Fp, self.d, self.s2b

    def f(self, xl):
        return 7.7 * xl[0] + 2.8 * xl[1] + 0.5 * xl[2] + 2.6 * xl[3] + 5.0 * xl[1] * xl[2] + 0.3 * xl[1] * xl[3] + \
            9.3 * xl[2] * xl[3] + 4.1*xl[1]*xl[2]*xl[3] + 1.4*xl[1]*xl[1] + 0.3*xl[2]*xl[2] + 7.1*xl[3]*xl[3]

    def do(self):
        self.y = list()
        for i in range(self.N):
            self.y.append([self.f(self.x_lines[i]) + randint(0, 10) - 5 for _ in range(self.m)])

    def do_more(self):
        for i in range(self.N):
            self.y[i].append(self.f(self.x_lines[i]) + randint(0, 10) - 5)
```

```
    def check_kohren(self):
        self.y_average = [0 for _ in range(self.N)]
        for i in range(self.N):
            self.y_average[i] = sum(self.y[i]) / self.m

        self.S2_dis = [0 for _ in range(self.N)]
        for i in range(self.N):
            for j in range(self.m):
                self.S2_dis[i] += (self.y[i][j] - self.y_average[i]) ** 2
            self.S2_dis[i] /= self.m

        self.Gp = max(self.S2_dis) / sum(self.S2_dis)
        self.f1 = self.m - 1
        self.f2 = self.N
        if not(ct.compare_kohren_with_table_value(self.f1, self.f2, self.Gp)):
            self.m += 1
            self.do_more()
            return self.check_kohren()

    def check_student(self, K, A, B):
        self.s2b = sum(self.S2_dis) / self.N
        s2_b = self.s2b / (self.N * self.m)
        s_b = sqrt(s2_b)
        self.t = [0 for _ in range(K)]
        for i in range(K):
            self.t[i] = abs(B[i]) / s_b
        self.f3 = self.f1 * self.f2

        self.d = K
        for i in range(K):
            if not ct.compare_student_with_table_value(self.f3, self.t[i]):
                A[i] = 0
                B[i] = 0
                self.d -= 1

    def check_fisher(self, model, nx_lines):
        y_for_fisher = [0 for _ in range(self.N)]
        for i in range(self.N):
            y_for_fisher[i] = model.calculate_with_nature_cfs(nx_lines[i])
        s2ad = 0
        for i in range(self.N):
            s2ad += (y_for_fisher[i] - self.y_average[i]) ** 2

        if self.N - self.d > 0:
            s2ad = self.m * s2ad / (self.N - self.d)
        else:
            s2ad = inf

        self.f4 = self.N - self.d
        self.Fp = s2ad / self.s2b
        return ct.compare_phisher_with_table_value(self.f3, self.f4, self.Fp)
```

linear_without_interaction.py

```
from numpy.linalg import det


class LinearWithoutInteractionModel:
    K = 0
    N = 0
    A = list()
    B = list()
```

```
    def __init__(self, K, N):
        self.K = K
        self.N = N

    def __del__(self):
        del self.K, self.N, self.A, self.B

    def find_nature_cfs(self, nx, y_average):
        """nx - матриця натуральних значень x"""
        mx1, mx2, mx3, my = sum(nx[1]) / self.N, sum(nx[2]) / self.N, sum(nx[3]) / self.N, sum(y_average) / self.N
        a11, a22, a33 = 0, 0, 0
        a12, a13, a23 = 0, 0, 0
        a1, a2, a3 = 0, 0, 0
        for i in range(self.N):
            a11 += nx[1][i] ** 2
            a22 += nx[2][i] ** 2
            a33 += nx[3][i] ** 2
            a12 += nx[1][i] * nx[2][i]
            a13 += nx[1][i] * nx[3][i]
            a23 += nx[2][i] * nx[3][i]
            a1 += y_average[i] * nx[1][i]
            a2 += y_average[i] * nx[2][i]
            a3 += y_average[i] * nx[3][i]
        a11, a22, a33 = a11 / self.N, a22 / self.N, a33 / self.N
        a12, a13, a23 = a12 / self.N, a13 / self.N, a23 / self.N
        a1, a2, a3 = a1 / self.N, a2 / self.N, a3 / self.N
        a21 = a12
        a31 = a13
        a32 = a23

        main_det = det([[1, mx1, mx2, mx3], [mx1, a11, a12, a13], [mx2, a21, a22, a23], [mx3, a31, a32, a33]])
        A0 = det([[my, mx1, mx2, mx3], [a1, a11, a12, a13], [a2, a21, a22, a23], [a3, a31, a32, a33]]) / main_det
        A1 = det([[1, my, mx2, mx3], [mx1, a1, a12, a13], [mx2, a2, a22, a23], [mx3, a3, a32, a33]]) / main_det
        A2 = det([[1, mx1, my, mx3], [mx1, a11, a1, a13], [mx2, a21, a2, a23], [mx3, a31, a3, a33]]) / main_det
        A3 = det([[1, mx1, mx2, my], [mx1, a11, a12, a1], [mx2, a21, a22, a2], [mx3, a31, a32, a3]]) / main_det
        self.A = [A0, A1, A2, A3]

    def find_encoded_cfs(self, x, y_average):
        """x - матриця натуральных значень факторів"""
        self.B = [0 for _ in range(self.K)]
        for i in range(self.N):
            self.B[0] += y_average[i] * x[0][i]
            self.B[1] += y_average[i] * x[1][i]
            self.B[2] += y_average[i] * x[2][i]
            self.B[3] += y_average[i] * x[3][i]

        for i in range(self.K):
            self.B[i] /= self.N

    def calculate_with_nature_cfs(self, nxl):
        """nxl - nature x line"""
        return self.A[0] * nxl[0] + self.A[1] * nxl[1] + self.A[2] * nxl[2] + self.A[3] * nxl[3]

    def calculate_with_encoded_cfs(self, xl):
        """xl - encoded x line"""
        return self.B[0] * xl[0] + self.B[1] * xl[1] + self.B[2] * xl[2] + self.B[3] * xl[3]
```

linear_with_interaction.py

```
from numpy.linalg import det
```

```
class LinearWithInteractionModel:
    K = 0
    N = 0
    A = list()
    B = list()

    def __init__(self, K, N):
        self.K = K
        self.N = N

    def __del__(self):
        del self.K, self.N, self.A, self.B

    def find_nature_cfs(self, nx, y_average):
        """nx - матриця натуральних значень x"""
        m00, m10, m20, m30, m40, m50, m60, m70, k0 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m01, m11, m21, m31, m41, m51, m61, m71, k1 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m02, m12, m22, m32, m42, m52, m62, m72, k2 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m03, m13, m23, m33, m43, m53, m63, m73, k3 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m04, m14, m24, m34, m44, m54, m64, m74, k4 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m05, m15, m25, m35, m45, m55, m65, m75, k5 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m06, m16, m26, m36, m46, m56, m66, m76, k6 = 0, 0, 0, 0, 0, 0, 0, 0, 0
        m07, m17, m27, m37, m47, m57, m67, m77, k7 = 0, 0, 0, 0, 0, 0, 0, 0, 0

        for i in range(self.N):
            m00 = self.N
            m10 += nx[1][i]
            m20 += nx[2][i]
            m30 += nx[3][i]
            m40 += nx[1][i] * nx[2][i]
            m50 += nx[1][i] * nx[3][i]
            m60 += nx[2][i] * nx[3][i]
            m70 += nx[1][i] * nx[2][i] * nx[3][i]
            k0 += y_average[i]
            m01 += nx[1][i]
            m11 += nx[1][i] ** 2
            m21 += nx[1][i] * nx[2][i]
            m31 += nx[1][i] * nx[3][i]
            m41 += (nx[1][i] ** 2) * nx[2][i]
            m51 += (nx[1][i] ** 2) * nx[3][i]
            m61 += nx[1][i] * nx[2][i] * nx[3][i]
            m71 += (nx[1][i] ** 2) * nx[2][i] * nx[3][i]
            k1 += y_average[i] * nx[1][i]
            m02 += nx[2][i]
            m12 += nx[1][i] * nx[2][i]
            m22 += nx[2][i] ** 2
            m32 += nx[2][i] * nx[3][i]
            m42 += nx[1][i] * (nx[2][i] ** 2)
            m52 += nx[1][i] * nx[2][i] * nx[3][i]
            m62 += (nx[2][i] ** 2) * nx[3][i]
            m72 += nx[1][i] * (nx[2][i] ** 2) * nx[3][i]
            k2 += y_average[i] * nx[2][i]
            m03 += nx[3][i]
            m13 += nx[1][i] * nx[3][i]
            m23 += nx[2][i] * nx[3][i]
            m33 += nx[3][i] ** 2
            m43 += nx[1][i] * nx[2][i] * nx[3][i]
            m53 += nx[1][i] * (nx[3][i] ** 2)
            m63 += nx[2][i] * (nx[3][i] ** 2)
            m73 += nx[1][i] * nx[2][i] * (nx[3][i] ** 2)
            k3 += y_average[i] * nx[3][i]
```

```
    m04 += nx[1][i] * nx[2][i]
    m14 += (nx[1][i] ** 2) * nx[2][i]
    m24 += nx[1][i] * (nx[2][i] ** 2)
    m34 += nx[1][i] * nx[2][i] * nx[3][i]
    m44 += (nx[1][i] ** 2) * (nx[2][i] ** 2)
    m54 += (nx[1][i] ** 2) * nx[2][i] * nx[3][i]
    m64 += nx[1][i] * (nx[2][i] ** 2) * nx[3][i]
    m74 += (nx[1][i] ** 2) * (nx[2][i] ** 2) * nx[3][i]
    k4 += y_average[i] * nx[1][i] * nx[2][i]
    m05 += nx[1][i] * nx[3][i]
    m15 += (nx[1][i] ** 2) * nx[3][i]
    m25 += nx[1][i] * nx[2][i] * nx[3][i]
    m35 += nx[1][i] * (nx[3][i] ** 2)
    m45 += (nx[1][i] ** 2) * nx[2][i] * nx[3][i]
    m55 += (nx[1][i] ** 2) * (nx[3][i] ** 2)
    m65 += nx[1][i] * nx[2][i] * (nx[3][i] ** 2)
    m75 += (nx[1][i] ** 2) * nx[2][i] * (nx[3][i] ** 2)
    k5 += y_average[i] * nx[1][i] * nx[3][i]
    m06 += nx[2][i] * nx[3][i]
    m16 += nx[1][i] * nx[2][i] * nx[3][i]
    m26 += (nx[2][i] ** 2) * nx[3][i]
    m36 += nx[2][i] * (nx[3][i] ** 2)
    m46 += nx[1][i] * (nx[2][i] ** 2) * nx[3][i]
    m56 += nx[1][i] * nx[2][i] * (nx[3][i] ** 2)
    m66 += (nx[2][i] ** 2) * (nx[3][i] ** 2)
    m76 += nx[1][i] * (nx[2][i] ** 2) * (nx[3][i] ** 2)
    k6 += y_average[i] * nx[2][i] * nx[3][i]
    m07 += nx[1][i] * nx[2][i] * nx[3][i]
    m17 += (nx[1][i] ** 2) * nx[2][i] * nx[3][i]
    m27 += nx[1][i] * (nx[2][i] ** 2) * nx[3][i]
    m37 += nx[1][i] * nx[2][i] * (nx[3][i] ** 2)
    m47 += (nx[1][i] ** 2) * (nx[2][i] ** 2) * nx[3][i]
    m57 += (nx[1][i] ** 2) * nx[2][i] * (nx[3][i] ** 2)
    m67 += nx[1][i] * (nx[2][i] ** 2) * (nx[3][i] ** 2)
    m77 += (nx[1][i] ** 2) * (nx[2][i] ** 2) * (nx[3][i] ** 2)
    k7 += y_average[i] * nx[1][i] * nx[2][i] * nx[3][i]

main_det = det([
    [m00, m10, m20, m30, m40, m50, m60, m70],
    [m01, m11, m21, m31, m41, m51, m61, m71],
    [m02, m12, m22, m32, m42, m52, m62, m72],
    [m03, m13, m23, m33, m43, m53, m63, m73],
    [m04, m14, m24, m34, m44, m54, m64, m74],
    [m05, m15, m25, m35, m45, m55, m65, m75],
    [m06, m16, m26, m36, m46, m56, m66, m76],
    [m07, m17, m27, m37, m47, m57, m67, m77]])

det0 = det([
    [k0, m10, m20, m30, m40, m50, m60, m70],
    [k1, m11, m21, m31, m41, m51, m61, m71],
    [k2, m12, m22, m32, m42, m52, m62, m72],
    [k3, m13, m23, m33, m43, m53, m63, m73],
    [k4, m14, m24, m34, m44, m54, m64, m74],
    [k5, m15, m25, m35, m45, m55, m65, m75],
    [k6, m16, m26, m36, m46, m56, m66, m76],
    [k7, m17, m27, m37, m47, m57, m67, m77]])

det1 = det([
    [m00, k0, m20, m30, m40, m50, m60, m70],
    [m01, k1, m21, m31, m41, m51, m61, m71],
    [m02, k2, m22, m32, m42, m52, m62, m72],
    [m03, k3, m23, m33, m43, m53, m63, m73],
```

```
    [m04, k4, m24, m34, m44, m54, m64, m74],
    [m05, k5, m25, m35, m45, m55, m65, m75],
    [m06, k6, m26, m36, m46, m56, m66, m76],
    [m07, k7, m27, m37, m47, m57, m67, m77]])

det2 = det([
    [m00, m10, k0, m30, m40, m50, m60, m70],
    [m01, m11, k1, m31, m41, m51, m61, m71],
    [m02, m12, k2, m32, m42, m52, m62, m72],
    [m03, m13, k3, m33, m43, m53, m63, m73],
    [m04, m14, k4, m34, m44, m54, m64, m74],
    [m05, m15, k5, m35, m45, m55, m65, m75],
    [m06, m16, k6, m36, m46, m56, m66, m76],
    [m07, m17, k7, m37, m47, m57, m67, m77]])

det3 = det([
    [m00, m10, m20, k0, m40, m50, m60, m70],
    [m01, m11, m21, k1, m41, m51, m61, m71],
    [m02, m12, m22, k2, m42, m52, m62, m72],
    [m03, m13, m23, k3, m43, m53, m63, m73],
    [m04, m14, m24, k4, m44, m54, m64, m74],
    [m05, m15, m25, k5, m45, m55, m65, m75],
    [m06, m16, m26, k6, m46, m56, m66, m76],
    [m07, m17, m27, k7, m47, m57, m67, m77]])

det4 = det([
    [m00, m10, m20, m30, k0, m50, m60, m70],
    [m01, m11, m21, m31, k1, m51, m61, m71],
    [m02, m12, m22, m32, k2, m52, m62, m72],
    [m03, m13, m23, m33, k3, m53, m63, m73],
    [m04, m14, m24, m34, k4, m54, m64, m74],
    [m05, m15, m25, m35, k5, m55, m65, m75],
    [m06, m16, m26, m36, k6, m56, m66, m76],
    [m07, m17, m27, m37, k7, m57, m67, m77]])

det5 = det([
    [m00, m10, m20, m30, m40, k0, m60, m70],
    [m01, m11, m21, m31, m41, k1, m61, m71],
    [m02, m12, m22, m32, m42, k2, m62, m72],
    [m03, m13, m23, m33, m43, k3, m63, m73],
    [m04, m14, m24, m34, m44, k4, m64, m74],
    [m05, m15, m25, m35, m45, k5, m65, m75],
    [m06, m16, m26, m36, m46, k6, m66, m76],
    [m07, m17, m27, m37, m47, k7, m67, m77]])

det6 = det([
    [m00, m10, m20, m30, m40, m50, k0, m70],
    [m01, m11, m21, m31, m41, m51, k1, m71],
    [m02, m12, m22, m32, m42, m52, k2, m72],
    [m03, m13, m23, m33, m43, m53, k3, m73],
    [m04, m14, m24, m34, m44, m54, k4, m74],
    [m05, m15, m25, m35, m45, m55, k5, m75],
    [m06, m16, m26, m36, m46, m56, k6, m76],
    [m07, m17, m27, m37, m47, m57, k7, m77]])

det7 = det([
    [m00, m10, m20, m30, m40, m50, m60, k0],
    [m01, m11, m21, m31, m41, m51, m61, k1],
    [m02, m12, m22, m32, m42, m52, m62, k2],
    [m03, m13, m23, m33, m43, m53, m63, k3],
    [m04, m14, m24, m34, m44, m54, m64, k4],
    [m05, m15, m25, m35, m45, m55, m65, k5],
```

```
            [m06, m16, m26, m36, m46, m56, m66, k6],
            [m07, m17, m27, m37, m47, m57, m67, k7]])

        A0 = det0 / main_det
        A1 = det1 / main_det
        A2 = det2 / main_det
        A3 = det3 / main_det
        A4 = det4 / main_det
        A5 = det5 / main_det
        A6 = det6 / main_det
        A7 = det7 / main_det

        self.A = [A0, A1, A2, A3, A4, A5, A6, A7]

    def find_encoded_cfs(self, x, y_average):
        """x - матриця натуральных значень факторів"""
        self.B = [0 for _ in range(self.K)]
        for i in range(self.N):
            self.B[0] += y_average[i] * x[0][i]
            self.B[1] += y_average[i] * x[1][i]
            self.B[2] += y_average[i] * x[2][i]
            self.B[3] += y_average[i] * x[3][i]
            self.B[4] += y_average[i] * x[1][i] * x[2][i]
            self.B[5] += y_average[i] * x[1][i] * x[3][i]
            self.B[6] += y_average[i] * x[2][i] * x[3][i]
            self.B[7] += y_average[i] * x[1][i] * x[2][i] * x[3][i]
        for i in range(self.K):
            self.B[i] /= self.N

    def calculate_with_nature_cfs(self, nxl):
        """nxl - nature x line"""
        return self.A[0]*nxl[0] + self.A[1]*nxl[1] + self.A[2]*nxl[2] + self.A[3]*nxl[3] + \
               self.A[4]*nxl[1]*nxl[2] + self.A[5]*nxl[1]*nxl[3] + self.A[6]*nxl[2]*nxl[3] + \
               self.A[7]*nxl[1]*nxl[2]*nxl[3]

    def calculate_with_encoded_cfs(self, xl):
        """xl - encoded x line"""
        return self.B[0]*xl[0] + self.B[1]*xl[1] + self.B[2]*xl[2] + self.B[3]*xl[3] + \
               self.B[4]*xl[1]*xl[2] + self.B[5]*xl[1]*xl[3] + self.B[6]*xl[2]*xl[3] + \
               self.B[7]*xl[1]*xl[2]*xl[3]
```

square_central_orthogonal.py

```
from numpy.linalg import det


class SquareCentralOrthogonalModel:
    K = 0
    N = 0
    A = list()
    B = list()

    def __init__(self, K, N):
        self.K = K
        self.N = N

    def __del__(self):
        del self.K, self.N, self.A, self.B

    def find_nature_cfs(self, m, nx, y):
        """nx - матриця натуральних значень x"""
        nx.append([nx[1][i] * nx[2][i] for i in range(self.N)])
```

```
        nx.append([nx[1][i] * nx[3][i] for i in range(self.N)])
        nx.append([nx[2][i] * nx[3][i] for i in range(self.N)])
        nx.append([nx[1][i] * nx[2][i] * nx[3][i] for i in range(self.N)])
        nx.append([nx[1][i] ** 2 for i in range(self.N)])
        nx.append([nx[2][i] ** 2 for i in range(self.N)])
        nx.append([nx[3][i] ** 2 for i in range(self.N)])
        self.A = self.find_cfs_core(m, nx, y)


    def find_encoded_cfs(self, m, x, y):
        """x - матриця натуральных значень факторів"""
        x.append([x[1][i] * x[2][i] for i in range(self.N)])
        x.append([x[1][i] * x[3][i] for i in range(self.N)])
        x.append([x[2][i] * x[3][i] for i in range(self.N)])
        x.append([x[1][i] * x[2][i] * x[3][i] for i in range(self.N)])
        x.append([x[1][i] ** 2 for i in range(self.N)])
        x.append([x[2][i] ** 2 for i in range(self.N)])
        x.append([x[3][i] ** 2 for i in range(self.N)])
        self.B = self.find_cfs_core(m, x, y)

    def find_cfs_core(self, m, x, y):
        m00, m10, m20, m30, m40, m50, m60, m70, m80, m90, m100, k0 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m01, m11, m21, m31, m41, m51, m61, m71, m81, m91, m101, k1 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m02, m12, m22, m32, m42, m52, m62, m72, m82, m92, m102, k2 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m03, m13, m23, m33, m43, m53, m63, m73, m83, m93, m103, k3 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m04, m14, m24, m34, m44, m54, m64, m74, m84, m94, m104, k4 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m05, m15, m25, m35, m45, m55, m65, m75, m85, m95, m105, k5 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m06, m16, m26, m36, m46, m56, m66, m76, m86, m96, m106, k6 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m07, m17, m27, m37, m47, m57, m67, m77, m87, m97, m107, k7 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m08, m18, m28, m38, m48, m58, m68, m78, m88, m98, m108, k8 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m09, m19, m29, m39, m49, m59, m69, m79, m89, m99, m109, k9 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        m010, m110, m210, m310, m410, m510, m610, m710, m810, m910, m1010, k10 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0

        for i in range(self.N):
            for j in range(m):
                m00 += x[0][i]
                m10 += x[1][i]
                m20 += x[2][i]
                m30 += x[3][i]
                m40 += x[4][i]
                m50 += x[5][i]
                m60 += x[6][i]
                m70 += x[7][i]
                m80 += x[8][i]
                m90 += x[9][i]
                m100 += x[10][i]
                k0 += y[i][j]
                m01 += x[0][i] * x[1][i]
                m11 += x[1][i] * x[1][i]
                m21 += x[2][i] * x[1][i]
                m31 += x[3][i] * x[1][i]
                m41 += x[4][i] * x[1][i]
                m51 += x[5][i] * x[1][i]
                m61 += x[6][i] * x[1][i]
                m71 += x[7][i] * x[1][i]
                m81 += x[8][i] * x[1][i]
                m91 += x[9][i] * x[1][i]
                m101 += x[10][i] * x[1][i]
                k1 += y[i][j] * x[1][i]
                m02 += x[0][i] * x[2][i]
                m12 += x[1][i] * x[2][i]
```

```
m22 += x[2][i] * x[2][i]
m32 += x[3][i] * x[2][i]
m42 += x[4][i] * x[2][i]
m52 += x[5][i] * x[2][i]
m62 += x[6][i] * x[2][i]
m72 += x[7][i] * x[2][i]
m82 += x[8][i] * x[2][i]
m92 += x[9][i] * x[2][i]
m102 += x[10][i] * x[2][i]
k2 += y[i][j] * x[2][i]
m03 += x[0][i] * x[3][i]
m13 += x[1][i] * x[3][i]
m23 += x[2][i] * x[3][i]
m33 += x[3][i] * x[3][i]
m43 += x[4][i] * x[3][i]
m53 += x[5][i] * x[3][i]
m63 += x[6][i] * x[3][i]
m73 += x[7][i] * x[3][i]
m83 += x[8][i] * x[3][i]
m93 += x[9][i] * x[3][i]
m103 += x[10][i] * x[3][i]
k3 += y[i][j] * x[3][i]
m04 += x[0][i] * x[4][i]
m14 += x[1][i] * x[4][i]
m24 += x[2][i] * x[4][i]
m34 += x[3][i] * x[4][i]
m44 += x[4][i] * x[4][i]
m54 += x[5][i] * x[4][i]
m64 += x[6][i] * x[4][i]
m74 += x[7][i] * x[4][i]
m84 += x[8][i] * x[4][i]
m94 += x[9][i] * x[4][i]
m104 += x[10][i] * x[4][i]
k4 += y[i][j] * x[4][i]
m05 += x[0][i] * x[5][i]
m15 += x[1][i] * x[5][i]
m25 += x[2][i] * x[5][i]
m35 += x[3][i] * x[5][i]
m45 += x[4][i] * x[5][i]
m55 += x[5][i] * x[5][i]
m65 += x[6][i] * x[5][i]
m75 += x[7][i] * x[5][i]
m85 += x[8][i] * x[5][i]
m95 += x[9][i] * x[5][i]
m105 += x[10][i] * x[5][i]
k5 += y[i][j] * x[5][i]
m06 += x[0][i] * x[6][i]
m16 += x[1][i] * x[6][i]
m26 += x[2][i] * x[6][i]
m36 += x[3][i] * x[6][i]
m46 += x[4][i] * x[6][i]
m56 += x[5][i] * x[6][i]
m66 += x[6][i] * x[6][i]
m76 += x[7][i] * x[6][i]
m86 += x[8][i] * x[6][i]
m96 += x[9][i] * x[6][i]
m106 += x[10][i] * x[6][i]
k6 += y[i][j] * x[6][i]
m07 += x[0][i] * x[7][i]
m17 += x[1][i] * x[7][i]
m27 += x[2][i] * x[7][i]
m37 += x[3][i] * x[7][i]
```

```
        m47 += x[4][i] * x[7][i]
        m57 += x[5][i] * x[7][i]
        m67 += x[6][i] * x[7][i]
        m77 += x[7][i] * x[7][i]
        m87 += x[8][i] * x[7][i]
        m97 += x[9][i] * x[7][i]
        m107 += x[10][i] * x[7][i]
        k7 += y[i][j] * x[7][i]
        m08 += x[0][i] * x[8][i]
        m18 += x[1][i] * x[8][i]
        m28 += x[2][i] * x[8][i]
        m38 += x[3][i] * x[8][i]
        m48 += x[4][i] * x[8][i]
        m58 += x[5][i] * x[8][i]
        m68 += x[6][i] * x[8][i]
        m78 += x[7][i] * x[8][i]
        m88 += x[8][i] * x[8][i]
        m98 += x[9][i] * x[8][i]
        m108 += x[10][i] * x[8][i]
        k8 += y[i][j] * x[8][i]
        m09 += x[0][i] * x[9][i]
        m19 += x[1][i] * x[9][i]
        m29 += x[2][i] * x[9][i]
        m39 += x[3][i] * x[9][i]
        m49 += x[4][i] * x[9][i]
        m59 += x[5][i] * x[9][i]
        m69 += x[6][i] * x[9][i]
        m79 += x[7][i] * x[9][i]
        m89 += x[8][i] * x[9][i]
        m99 += x[9][i] * x[9][i]
        m109 += x[10][i] * x[9][i]
        k9 += y[i][j] * x[9][i]
        m010 += x[0][i] * x[10][i]
        m110 += x[1][i] * x[10][i]
        m210 += x[2][i] * x[10][i]
        m310 += x[3][i] * x[10][i]
        m410 += x[4][i] * x[10][i]
        m510 += x[5][i] * x[10][i]
        m610 += x[6][i] * x[10][i]
        m710 += x[7][i] * x[10][i]
        m810 += x[8][i] * x[10][i]
        m910 += x[9][i] * x[10][i]
        m1010 += x[10][i] * x[10][i]
        k10 += y[i][j] * x[10][i]

    main_det = det([
      [m00, m10, m20, m30, m40, m50, m60, m70, m80, m90, m100],
      [m01, m11, m21, m31, m41, m51, m61, m71, m81, m91, m101],
      [m02, m12, m22, m32, m42, m52, m62, m72, m82, m92, m102],
      [m03, m13, m23, m33, m43, m53, m63, m73, m83, m93, m103],
      [m04, m14, m24, m34, m44, m54, m64, m74, m84, m94, m104],
      [m05, m15, m25, m35, m45, m55, m65, m75, m85, m95, m105],
      [m06, m16, m26, m36, m46, m56, m66, m76, m86, m96, m106],
      [m07, m17, m27, m37, m47, m57, m67, m77, m87, m97, m107],
      [m08, m18, m28, m38, m48, m58, m68, m78, m88, m98, m108],
      [m09, m19, m29, m39, m49, m59, m69, m79, m89, m99, m109],
      [m010, m110, m210, m310, m410, m510, m610, m710, m810, m910, m1010]
    ])

    det0 = det([
      [k0, m10, m20, m30, m40, m50, m60, m70, m80, m90, m100],
      [k1, m11, m21, m31, m41, m51, m61, m71, m81, m91, m101],
```

```
    [k2, m12, m22, m32, m42, m52, m62, m72, m82, m92, m102],
    [k3, m13, m23, m33, m43, m53, m63, m73, m83, m93, m103],
    [k4, m14, m24, m34, m44, m54, m64, m74, m84, m94, m104],
    [k5, m15, m25, m35, m45, m55, m65, m75, m85, m95, m105],
    [k6, m16, m26, m36, m46, m56, m66, m76, m86, m96, m106],
    [k7, m17, m27, m37, m47, m57, m67, m77, m87, m97, m107],
    [k8, m18, m28, m38, m48, m58, m68, m78, m88, m98, m108],
    [k9, m19, m29, m39, m49, m59, m69, m79, m89, m99, m109],
    [k10, m110, m210, m310, m410, m510, m610, m710, m810, m910, m1010]
])

det1 = det([
    [m00, k0, m20, m30, m40, m50, m60, m70, m80, m90, m100],
    [m01, k1, m21, m31, m41, m51, m61, m71, m81, m91, m101],
    [m02, k2, m22, m32, m42, m52, m62, m72, m82, m92, m102],
    [m03, k3, m23, m33, m43, m53, m63, m73, m83, m93, m103],
    [m04, k4, m24, m34, m44, m54, m64, m74, m84, m94, m104],
    [m05, k5, m25, m35, m45, m55, m65, m75, m85, m95, m105],
    [m06, k6, m26, m36, m46, m56, m66, m76, m86, m96, m106],
    [m07, k7, m27, m37, m47, m57, m67, m77, m87, m97, m107],
    [m08, k8, m28, m38, m48, m58, m68, m78, m88, m98, m108],
    [m09, k9, m29, m39, m49, m59, m69, m79, m89, m99, m109],
    [m010, k10, m210, m310, m410, m510, m610, m710, m810, m910, m1010]
])

det2 = det([
    [m00, m10, k0, m30, m40, m50, m60, m70, m80, m90, m100],
    [m01, m11, k1, m31, m41, m51, m61, m71, m81, m91, m101],
    [m02, m12, k2, m32, m42, m52, m62, m72, m82, m92, m102],
    [m03, m13, k3, m33, m43, m53, m63, m73, m83, m93, m103],
    [m04, m14, k4, m34, m44, m54, m64, m74, m84, m94, m104],
    [m05, m15, k5, m35, m45, m55, m65, m75, m85, m95, m105],
    [m06, m16, k6, m36, m46, m56, m66, m76, m86, m96, m106],
    [m07, m17, k7, m37, m47, m57, m67, m77, m87, m97, m107],
    [m08, m18, k8, m38, m48, m58, m68, m78, m88, m98, m108],
    [m09, m19, k9, m39, m49, m59, m69, m79, m89, m99, m109],
    [m010, m110, k10, m310, m410, m510, m610, m710, m810, m910, m1010]
])

det3 = det([
    [m00, m10, m20, k0, m40, m50, m60, m70, m80, m90, m100],
    [m01, m11, m21, k1, m41, m51, m61, m71, m81, m91, m101],
    [m02, m12, m22, k2, m42, m52, m62, m72, m82, m92, m102],
    [m03, m13, m23, k3, m43, m53, m63, m73, m83, m93, m103],
    [m04, m14, m24, k4, m44, m54, m64, m74, m84, m94, m104],
    [m05, m15, m25, k5, m45, m55, m65, m75, m85, m95, m105],
    [m06, m16, m26, k6, m46, m56, m66, m76, m86, m96, m106],
    [m07, m17, m27, k7, m47, m57, m67, m77, m87, m97, m107],
    [m08, m18, m28, k8, m48, m58, m68, m78, m88, m98, m108],
    [m09, m19, m29, k9, m49, m59, m69, m79, m89, m99, m109],
    [m010, m110, m210, k10, m410, m510, m610, m710, m810, m910, m1010]
])

det4 = det([
    [m00, m10, m20, m30, k0, m50, m60, m70, m80, m90, m100],
    [m01, m11, m21, m31, k1, m51, m61, m71, m81, m91, m101],
    [m02, m12, m22, m32, k2, m52, m62, m72, m82, m92, m102],
    [m03, m13, m23, m33, k3, m53, m63, m73, m83, m93, m103],
    [m04, m14, m24, m34, k4, m54, m64, m74, m84, m94, m104],
    [m05, m15, m25, m35, k5, m55, m65, m75, m85, m95, m105],
    [m06, m16, m26, m36, k6, m56, m66, m76, m86, m96, m106],
    [m07, m17, m27, m37, k7, m57, m67, m77, m87, m97, m107],
```

```
    [m08, m18, m28, m38, k8, m58, m68, m78, m88, m98, m108],
    [m09, m19, m29, m39, k9, m59, m69, m79, m89, m99, m109],
    [m010, m110, m210, m310, k10, m510, m610, m710, m810, m910, m1010]
])

det5 = det([
    [m00, m10, m20, m30, m40, k0, m60, m70, m80, m90, m100],
    [m01, m11, m21, m31, m41, k1, m61, m71, m81, m91, m101],
    [m02, m12, m22, m32, m42, k2, m62, m72, m82, m92, m102],
    [m03, m13, m23, m33, m43, k3, m63, m73, m83, m93, m103],
    [m04, m14, m24, m34, m44, k4, m64, m74, m84, m94, m104],
    [m05, m15, m25, m35, m45, k5, m65, m75, m85, m95, m105],
    [m06, m16, m26, m36, m46, k6, m66, m76, m86, m96, m106],
    [m07, m17, m27, m37, m47, k7, m67, m77, m87, m97, m107],
    [m08, m18, m28, m38, m48, k8, m68, m78, m88, m98, m108],
    [m09, m19, m29, m39, m49, k9, m69, m79, m89, m99, m109],
    [m010, m110, m210, m310, m410, k10, m610, m710, m810, m910, m1010]
])

det6 = det([
    [m00, m10, m20, m30, m40, m50, k0, m70, m80, m90, m100],
    [m01, m11, m21, m31, m41, m51, k1, m71, m81, m91, m101],
    [m02, m12, m22, m32, m42, m52, k2, m72, m82, m92, m102],
    [m03, m13, m23, m33, m43, m53, k3, m73, m83, m93, m103],
    [m04, m14, m24, m34, m44, m54, k4, m74, m84, m94, m104],
    [m05, m15, m25, m35, m45, m55, k5, m75, m85, m95, m105],
    [m06, m16, m26, m36, m46, m56, k6, m76, m86, m96, m106],
    [m07, m17, m27, m37, m47, m57, k7, m77, m87, m97, m107],
    [m08, m18, m28, m38, m48, m58, k8, m78, m88, m98, m108],
    [m09, m19, m29, m39, m49, m59, k9, m79, m89, m99, m109],
    [m010, m110, m210, m310, m410, m510, k10, m710, m810, m910, m1010]
])

det7 = det([
    [m00, m10, m20, m30, m40, m50, m60, k0, m80, m90, m100],
    [m01, m11, m21, m31, m41, m51, m61, k1, m81, m91, m101],
    [m02, m12, m22, m32, m42, m52, m62, k2, m82, m92, m102],
    [m03, m13, m23, m33, m43, m53, m63, k3, m83, m93, m103],
    [m04, m14, m24, m34, m44, m54, m64, k4, m84, m94, m104],
    [m05, m15, m25, m35, m45, m55, m65, k5, m85, m95, m105],
    [m06, m16, m26, m36, m46, m56, m66, k6, m86, m96, m106],
    [m07, m17, m27, m37, m47, m57, m67, k7, m87, m97, m107],
    [m08, m18, m28, m38, m48, m58, m68, k8, m88, m98, m108],
    [m09, m19, m29, m39, m49, m59, m69, k9, m89, m99, m109],
    [m010, m110, m210, m310, m410, m510, m610, k10, m810, m910, m1010]
])

det8 = det([
    [m00, m10, m20, m30, m40, m50, m60, m70, k0, m90, m100],
    [m01, m11, m21, m31, m41, m51, m61, m71, k1, m91, m101],
    [m02, m12, m22, m32, m42, m52, m62, m72, k2, m92, m102],
    [m03, m13, m23, m33, m43, m53, m63, m73, k3, m93, m103],
    [m04, m14, m24, m34, m44, m54, m64, m74, k4, m94, m104],
    [m05, m15, m25, m35, m45, m55, m65, m75, k5, m95, m105],
    [m06, m16, m26, m36, m46, m56, m66, m76, k6, m96, m106],
    [m07, m17, m27, m37, m47, m57, m67, m77, k7, m97, m107],
    [m08, m18, m28, m38, m48, m58, m68, m78, k8, m98, m108],
    [m09, m19, m29, m39, m49, m59, m69, m79, k9, m99, m109],
    [m010, m110, m210, m310, m410, m510, m610, m710, k10, m910, m1010]
])

det9 = det([
```

```
        [m00, m10, m20, m30, m40, m50, m60, m70, m80, k0, m100],
        [m01, m11, m21, m31, m41, m51, m61, m71, m81, k1, m101],
        [m02, m12, m22, m32, m42, m52, m62, m72, m82, k2, m102],
        [m03, m13, m23, m33, m43, m53, m63, m73, m83, k3, m103],
        [m04, m14, m24, m34, m44, m54, m64, m74, m84, k4, m104],
        [m05, m15, m25, m35, m45, m55, m65, m75, m85, k5, m105],
        [m06, m16, m26, m36, m46, m56, m66, m76, m86, k6, m106],
        [m07, m17, m27, m37, m47, m57, m67, m77, m87, k7, m107],
        [m08, m18, m28, m38, m48, m58, m68, m78, m88, k8, m108],
        [m09, m19, m29, m39, m49, m59, m69, m79, m89, k9, m109],
        [m010, m110, m210, m310, m410, m510, m610, m710, m810, k10, m1010]
    ])

    det10 = det([
        [m00, m10, m20, m30, m40, m50, m60, m70, m80, m90, k0],
        [m01, m11, m21, m31, m41, m51, m61, m71, m81, m91, k1],
        [m02, m12, m22, m32, m42, m52, m62, m72, m82, m92, k2],
        [m03, m13, m23, m33, m43, m53, m63, m73, m83, m93, k3],
        [m04, m14, m24, m34, m44, m54, m64, m74, m84, m94, k4],
        [m05, m15, m25, m35, m45, m55, m65, m75, m85, m95, k5],
        [m06, m16, m26, m36, m46, m56, m66, m76, m86, m96, k6],
        [m07, m17, m27, m37, m47, m57, m67, m77, m87, m97, k7],
        [m08, m18, m28, m38, m48, m58, m68, m78, m88, m98, k8],
        [m09, m19, m29, m39, m49, m59, m69, m79, m89, m99, k9],
        [m010, m110, m210, m310, m410, m510, m610, m710, m810, m910, k10]
    ])

    cfs0 = det0 / main_det
    cfs1 = det1 / main_det
    cfs2 = det2 / main_det
    cfs3 = det3 / main_det
    cfs4 = det4 / main_det
    cfs5 = det5 / main_det
    cfs6 = det6 / main_det
    cfs7 = det7 / main_det
    cfs8 = det8 / main_det
    cfs9 = det9 / main_det
    cfs10 = det10 / main_det

    return [cfs0, cfs1, cfs2, cfs3, cfs4, cfs5, cfs6, cfs7, cfs8, cfs9, cfs10]


  def calculate_with_nature_cfs(self, nxl):
    """nxl - nature x line"""
    return self.A[0]*nxl[0] + self.A[1]*nxl[1] + self.A[2]*nxl[2] + self.A[3]*nxl[3] + \
        self.A[4]*nxl[1]*nxl[2] + self.A[5]*nxl[1]*nxl[3] + self.A[6]*nxl[2]*nxl[3] + \
        self.A[7]*nxl[1]*nxl[2]*nxl[3] + self.A[8]*(nxl[1] ** 2) + self.A[9]*(nxl[2] ** 2) +\
        self.A[10]*(nxl[3] ** 2)

  def calculate_with_encoded_cfs(self, xl):
    """xl - encoded x line"""
    return self.B[0]*xl[0] + self.B[1]*xl[1] + self.B[2]*xl[2] + self.B[3]*xl[3] + \
        self.B[4]*xl[1]*xl[2] + self.B[5]*xl[1]*xl[3] + self.B[6]*xl[2]*xl[3] + \
        self.B[7]*xl[1]*xl[2]*xl[3] + self.B[8]*(xl[1] ** 2) + self.B[9]*(xl[2] ** 2) +\
        self.B[10]*(xl[3] ** 2)
```

criterion_tables.py

```
"""Таблиця для критерія Кохрена"""
base_kohren = [
  [9985, 9750, 9392, 9057, 8772, 8534, 8332, 8159, 8010, 7880, 7341, 6602, 5813, 5000],
  [9669, 8709, 7977, 7457, 7071, 6771, 6530, 6333, 6167, 6025, 5466, 4748, 4031, 3333],
```

```
    [9065, 7679, 6841, 6287, 5892, 5598, 5365, 5175, 5017, 4884, 4366, 3720, 3093, 2500],
    [8412, 6838, 5981, 5440, 5063, 4783, 4564, 4387, 4241, 4118, 3645, 3066, 2513, 2000],
    [7808, 6161, 5321, 4803, 4447, 4184, 3980, 3817, 3682, 3568, 3135, 2612, 2119, 1667],
    [7271, 5612, 4800, 4307, 3974, 3726, 3535, 3384, 3259, 3154, 2756, 2278, 1833, 1429],
    [6798, 5157, 4377, 3910, 3595, 3362, 3185, 3043, 2926, 2829, 2462, 2022, 1616, 1250],
    [6385, 4775, 4027, 3584, 3286, 3067, 2901, 2768, 2659, 2568, 2226, 1820, 1446, 1111],
    [6020, 4450, 3733, 3311, 3029, 2823, 2666, 2541, 2439, 2353, 2032, 1655, 1308, 1000],
    [5410, 3924, 3264, 2880, 2624, 2439, 2299, 2187, 2098, 2020, 1737, 1403, 1000, 833],
    [4709, 3346, 2758, 2419, 2159, 2034, 1911, 1815, 1736, 1671, 1429, 1144, 889,  667],
    [3894, 2705, 2205, 1921, 1735, 1602, 1501, 1422, 1357, 1303, 1108, 879,  675,  500],
    [3434, 2354, 1907, 1656, 1493, 1374, 1286, 1216, 1160, 1113, 942,  743,  567,  417],
    [2929, 1980, 1593, 1377, 1237, 1137, 1061, 1002, 958,  921,  771,  604,  457,  333],
    [2370, 1576, 1259, 1082, 968,  887,  827,  780,  745,  713,  595,  462,  347,  250],
    [1737, 1131, 895,  766,  682,  623,  583,  552,  520,  497,  411,  316,  234,  167],
    [998,  632,  495,  419,  371,  337,  312,  292,  279,  266,  218,  165,  120,  83],
]
column_kohren_f1 = {(1,): 0, (2,): 1, (3,): 2, (4,): 3, (5,): 4, (6,): 5, (7,): 6, (8,): 7, (9,): 8,
                    (range(10, 14)): 9, (range(14, 26)): 10, (range(26, 91)): 11, (range(91, 145)): 12}
COLUMN_KOHREN_F1_ELSE = 13

row_kohren_f2 = {(2,): 0, (3,): 1, (4,): 2, (5,): 3, (6,): 4, (7,): 5, (8,): 6, (9,): 7,
                 (range(10, 12)): 8, (range(12, 14)): 9, (range(14, 18)): 10, (range(18, 23)): 11,
                 (range(23, 28)): 12, (range(28, 36)): 13, (range(36, 51)): 14, (range(51, 81)): 15,
                 (range(81, 121)): 16}

ROW_KOHREN_F2_ELSE = 16

"""Таблиця для t-критерія Стьюдента"""
base_student_f3 = {(1,): 12.706, (2,): 4.303, (3,): 3.182, (4,): 2.776, (5,): 2.571, (6,): 2.447, (7,): 2.365,
                   (8,): 2.306, (9,): 2.262, (10,): 2.228, (11,): 2.201, (12,): 2.179, (13,): 2.160, (14,): 2.145,
                   (15,): 2.131, (16,): 2.120, (17,): 2.110, (18,): 2.101, (19,): 2.093, (20,): 2.086,
                   (range(21, 25)): 2.069, (range(25, 30)): 2.060, (range(30, 40)): 2.042, (range(40, 60)): 2.021,
                   (range(60, 81)): 2.000, (range(81, 111)): 1.980, (range(111, 121)): 1.960}
T_STUDENT_ELSE = 1.960

"""Таблиця для F-критерія Фішера"""
base_phisher = [
    [164.4, 199.5, 215.7, 224.6, 230.2, 234.0, 244.9, 249.0, 254.3],
    [18.5,  19.2,  19.2,  19.3,  19.3,  19.3,  19.4,  19.4,  19.5],
    [10.1,  9.6,   9.3,   9.1,   9.0,   8.9,   8.7,   8.6,   8.5],
    [7.7,   6.9,   6.6,   6.4,   6.3,   6.2,   5.9,   5.8,   5.6],
    [6.6,   5.8,   5.4,   5.2,   5.1,   5.0,   4.7,   4.5,   4.4],
    [6.0,   5.1,   4.8,   4.5,   4.4,   4.3,   4.0,   3.8,   3.7],
    [5.5,   4.7,   4.4,   4.1,   4.0,   3.9,   3.6,   3.4,   3.2],
    [5.3,   4.5,   4.1,   3.8,   3.7,   3.6,   3.3,   3.1,   2.9],
    [5.1,   4.3,   3.9,   3.6,   3.5,   3.4,   3.1,   2.9,   2.7],
    [5.0,   4.1,   3.7,   3.5,   3.3,   3.2,   2.9,   2.7,   2.5],
    [4.8,   4.0,   3.6,   3.4,   3.2,   3.1,   2.8,   2.6,   2.4],
    [4.8,   3.9,   3.5,   3.3,   3.1,   3.0,   2.7,   2.5,   2.3],
    [4.7,   3.8,   3.4,   3.2,   3.0,   2.9,   2.6,   2.4,   2.2],
    [4.6,   3.7,   3.3,   3.1,   3.0,   2.9,   2.5,   2.3,   2.1],
    [4.5,   3.7,   3.3,   3.1,   2.9,   2.8,   2.5,   2.3,   2.1],
    [4.5,   3.6,   3.2,   3.0,   2.9,   2.7,   2.4,   2.2,   2.0],
    [4.5,   3.6,   3.2,   3.0,   2.8,   2.7,   2.4,   2.2,   2.0],
    [4.4,   3.6,   3.2,   2.9,   2.8,   2.7,   2.3,   2.1,   1.9],
    [4.4,   3.5,   3.1,   2.9,   2.7,   2.6,   2.3,   2.1,   1.9],
    [4.4,   3.5,   3.1,   2.9,   2.7,   2.6,   2.3,   2.1,   1.9],
    [4.3,   3.4,   3.1,   2.8,   2.7,   2.6,   2.2,   2.0,   1.8],
    [4.3,   3.4,   3.0,   2.8,   2.6,   2.5,   2.2,   2.0,   1.7],
    [4.2,   3.4,   3.0,   2.7,   2.6,   2.5,   2.2,   2.0,   1.7],
    [4.2,   3.3,   3.0,   2.7,   2.6,   2.4,   2.1,   1.9,   1.7],
    [4.2,   3.3,   2.9,   2.7,   2.5,   2.4,   2.1,   1.9,   1.6],
```

```
    [4.1,  3.2,  2.9,  2.6,  2.5,  2.3,  2.0,  1.8,  1.5],
    [4.0,  3.2,  2.8,  2.5,  2.4,  2.3,  1.9,  1.7,  1.4],
    [3.9,  3.1,  2.7,  2.5,  2.3,  2.2,  1.8,  1.6,  1.3],
    [3.8,  3.0,  2.6,  2.4,  2.2,  2.1,  1.8,  1.5,  1.0]
]

row_phisher_f3 = {(1,): 0, (2,): 1, (3,): 2, (4,): 3, (5,): 4, (6,): 5, (7,): 6, (8,): 7, (9,): 8, (10,): 9,
                  (11,): 10, (12,): 11, (13,): 12, (14,): 13, (15,): 14, (16,): 15, (17,): 16, (18,): 17, (19,): 18,
                  (20, 21): 19, (22, 23): 20, (24, 25): 21, (26, 27): 22, (28, 29): 23, (range(30, 36)): 24,
                  (range(36, 51)): 25, (range(51, 91)): 26, (range(91, 121)): 27}
ROW_PHISHER_F3_ELSE = 28

column_phisher_f4 = {(1,): 0, (2,): 1, (3,): 2, (4,): 3, (5,): 4, (range(6, 10)): 5, (range(10, 19)): 6,
                     (range(19, 25)): 7}
COLUMN_PHISHER_F4_ELSE = 8


def compare_kohren_with_table_value(f1, f2, Gp):
    """True, якщо дисперсія однорідна"""
    row = -1
    column = -1
    for key in row_kohren_f2.keys():
        if f2 in key:
            row = row_kohren_f2[key]
            break
    if row == -1:
        row = ROW_KOHREN_F2_ELSE

    for key in column_kohren_f1.keys():
        if f1 in key:
            column = column_kohren_f1[key]
            break
    if column == -1:
        column = COLUMN_KOHREN_F1_ELSE
    return Gp < (base_kohren[row][column]/1000)


def compare_student_with_table_value(f3, t_exp):
    """True, якщо коефіцієнт Bs є значущим."""
    t_teo = -1
    for key in base_student_f3.keys():
        if f3 in key:
            t_teo = base_student_f3[key]
            break

    if t_teo == -1:
        t_teo = T_STUDENT_ELSE

    return t_exp > t_teo


def compare_phisher_with_table_value(f3, f4, Fp):
    """True, якщо отримана математична модель адекватна експериментальним даним."""
    row = -1
    column = -1
    for key in row_phisher_f3.keys():
        if f3 in key:
            row = row_phisher_f3[key]
            break
    if row == -1:
        row = ROW_PHISHER_F3_ELSE
```

```
    for key in column_phisher_f4.keys():
        if f4 in key:
            column = column_phisher_f4[key]
            break
    if column == -1:
        column = COLUMN_PHISHER_F4_ELSE
    return Fp <= base_phisher[row][column]
```

logs.py

```
from beautifultable import BeautifulTable

SYSTEM_MARK_L = "Action: "
SYSTEM_MARK_V = "View: "

text = {
    0: "Розглянемо лінійне рівняння регресії без взаємодії факторів:\ny = b0 + b1·x1 + b2·x2 + b3·x3",
    1: "Розглянемо лінійне рівняння регресії із врахуванням взаємодії факторів:\n"
       "y = b0 + b1·x1 + b2·x2 + b3·x3 + b12·x1·x2 + b13·x1·x3 + b23·x2·x3 + b123·x1·x2·x3",
    2: "Розглянемо рівняння регресії із врахуванням квадратичних членів:\n"
       "y = b0 + b1·x1 + b2·x2 + b3·x3 + b12·x1·x2 + b13·x1·x3 + b23·x2·x3 + b123·x1·x2·x3 + " +
       "b11·x1·x1 + b22·x2·x2 + b33·x3·x3",
    3: "Маємо, N = {0}, K = {1}, m = {2}.",
    4: "Складаємо матрицю планування і проведимо експерименти",
    5: "Розраховуємо натуральні значення коефіцієнтів.",
    6: "Розраховуємо кодовані значення коефіцієнтів.",
    7: "Рівняння регресії має вигляд (нат. знач. коеф.):\n" +
       "y = {0} + {1}·x1 + {2}·x2 + {3}·x3",
    8: "Рівняння регресії має вигляд (код. знач. коеф.):\n" +
       "y = {0} + {1}·x1 + {2}·x2 + {3}·x3",
    9: "Рівняння регресії має вигляд (нат. знач. коеф.):\n" +
       "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3",
    10: "Рівняння регресії має вигляд (код. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3",
    11: "Рівняння регресії має вигляд (нат. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3 + " +
        "{8}·x1·x1 + {9}·x2·x2 + {10}·x3·x3",
    12: "Рівняння регресії має вигляд (код. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3 + " +
        "{8}·x1·x1 + {9}·x2·x2 + {10}·x3·x3",
    13: "Дисперсія однорідна за критерієм Кохрена при m = {0}:\n" +
        "f1 = {1}, f2 = {2}, Gp = {3}",
    14: "Перевіряємо нуль гіпотезу та корегуємо рівняння регресії:\n" +
        "f3 = {0}, t = {1}.",
    15: "Нове рівняння регресії має вигляд (нат. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3",
    16: "Нове рівняння регресії має вигляд (код. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3",
    17: "Нове рівняння регресії має вигляд (нат. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3",
    18: "Нове рівняння регресії має вигляд (код. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3",
    19: "Нове рівняння регресії має вигляд (нат. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3 " +
        "+ {8}·x1·x1 + {9}·x2·x2 + {10}·x3·x3",
    20: "Нове рівняння регресії має вигляд (код. знач. коеф.):\n" +
        "y = {0} + {1}·x1 + {2}·x2 + {3}·x3 + {4}·x1·x2 + {5}·x1·x3 + {6}·x2·x3 + {7}·x1·x2·x3 " +
        "+ {8}·x1·x1 + {9}·x2·x2 + {10}·x3·x3",
    21: "Перевіряємо адекватність моделі.\n" +
        "f3 = {0}, f4 = {1}, Fp = {2}",
    22: "Модель адекватна оригіналу.",
    23: "Модель не адекватна оригіналу.",
    24: "Змінюємо рівняння регресії.\n",
```

```
    25: "Виводимо результати.",
    26: "Оскільки всі моделі не адекватні, то почнемо експерименти з початку.",
    27: "Перевірка критерія Кохрена займає: {0}",
    28: "Перевірка критерія Стьюдента займає: {0}",
    29: "Перевірка критерія Фішера займає: {0}"}


def comment(key, par):
    return print(SYSTEM_MARK_L + text[key].format(*par))


titles = {0: "Матриця планування експерименту (нат. знач. коеф., без взаємодії)",
          1: "Матриця планування експерименту (код. знач. коеф., без взаємодії)",
          2: "Матриця планування експерименту (нат. знач. коеф., із взаємодією)",
          3: "Матриця планування експерименту (код. знач. коеф., із взаємодією)",
          4: "Рототабельний композиційний план (нат. знач. коеф.)",
          5: "Рототабельний композиційний план (код. знач. коеф.)",
          6: "Перевірка знайдених коефіцієнтів (нат. знач. коеф., без взаємодії)",
          7: "Перевірка знайдених коефіцієнтів (код. знач. коеф., без взаємодії)",
          8: "Перевірка знайдених коефіцієнтів (нат. знач. коеф., із взаємодією)",
          9: "Перевірка знайдених коефіцієнтів (код. знач. коеф., із взаємодією)",
          10: "Перевірка знайдених коефіцієнтів (нат. знач. коеф., із квад. членами)",
          11: "Перевірка знайдених коефіцієнтів (код. знач. коеф., із квад. членами)"}

x_headers = {
    0: ["№", "X0", "X1", "X2", "X3"],
    1: ["№", "X0", "X1", "X2", "X3", "X1·X2", "X1·X3", "X2·X3", "X1·X2·X3"],
    2: ["№", "X0", "X1", "X2", "X3", "X1·X2", "X1·X3", "X2·X3", "X1·X2·X3", "X1·X1", "X2·X2", "X3·X3"]
}


def show_plan(title_index, x_header_index, x_lines, experiment):
    print(SYSTEM_MARK_V + titles[title_index])
    plan = BeautifulTable()
    plan.max_table_width = 1000
    y_headers = [f"Y{i + 1}" for i in range(experiment.m)]
    plan.column_headers = [*x_headers[x_header_index], *y_headers]
    for i in range(experiment.N):
        plan.append_row([i + 1, *x_lines[i], *experiment.y[i]])
    print(plan, "\n")


def show_natured_checking_matrix(title_index, x_header_index, x_lines, model, experiment):
    print(SYSTEM_MARK_V + titles[title_index])
    natured_checking_matrix = BeautifulTable()
    natured_checking_matrix.max_table_width = 1000
    natured_checking_matrix.column_headers = [*x_headers[x_header_index], "Average Y[j]", "Exp-tal Y[j]"]
    y_average = experiment.y_average
    for i in range(experiment.N):
        y_exp = model.calculate_with_nature_cfs(x_lines[i])
        natured_checking_matrix.append_row([i+1, *x_lines[i], y_average[i], y_exp])
    print(natured_checking_matrix, "\n")


def show_encoded_checking_matrix(title_index, x_header_index, x_lines, model, experiment):
    print(SYSTEM_MARK_V + titles[title_index])
    encoded_checking_matrix = BeautifulTable()
    encoded_checking_matrix.max_table_width = 1000
    encoded_checking_matrix.column_headers = [*x_headers[x_header_index], "Average Y[j]", "Exp-tal Y[j]"]
    y_average = experiment.y_average
    for i in range(experiment.N):
        y_exp = model.calculate_with_encoded_cfs(x_lines[i])
```

```
    encoded_checking_matrix.append_row([i+1, *x_lines[i], y_average[i], y_exp])
print(encoded_checking_matrix, "\n")
```

Результат виконання роботи

```
Action: Розглянемо лінійне рівняння регресії без взаємодії факторів:
y = b0 + b1·x1 + b2·x2 + b3·x3
Action: Маємо, N = 4, K = 4, m = 2.
Action: Складаємо матрицю планування і проведимо експерименти
Action: Дисперсія однорідна за критерієм Кохрена при m = 2:
f1 = 1, f2 = 4, Gp = 0.72
View: Матриця планування експерименту (нат. знач. коеф., без взаємодії)
```

№	X0	X1	X2	X3	Y1	Y2
1	1	-25	25	40	-84148.3	-84151.3
2	1	-25	65	25	-153146.3	-153148.3
3	1	75	25	25	220727.7	220733.7
4	1	75	65	40	869815.7	869816.7

```
View: Матриця планування експерименту (код. знач. коеф., без взаємодії)
```

№	X0	X1	X2	X3	Y1	Y2
1	1	-1	-1	1	-84148.3	-84151.3
2	1	-1	1	-1	-153146.3	-153148.3
3	1	1	-1	-1	220727.7	220733.7
4	1	1	1	1	869815.7	869816.7

```
Action: Розраховуємо натуральні значення коефіцієнтів.
Action: Рівняння регресії має вигляд (нат. знач. коеф.):
y = -1056890.8 + 6639.22·x1 + 7251.1·x2 + 23936.1·x3
Action: Розраховуємо кодовані значення коефіцієнтів.
Action: Рівняння регресії має вигляд (код. знач. коеф.):
y = 213312.45 + 331961.0·x1 + 145022.0·x2 + 179520.75·x3
View: Перевірка знайдених коефіцієнтів (нат. знач. коеф., без взаємодії)
```

№	X0	X1	X2	X3	Average Y[j]	Exp-tal Y[j]
1	1	-25	25	40	-84149.8	-84149.8
2	1	-25	65	25	-153147.3	-153147.3
3	1	75	25	25	220730.7	220730.7
4	1	75	65	40	869816.2	869816.2

View: Перевірка знайдених коефіцієнтів (код. знач. коеф., без взаємодії)

№	X0	X1	X2	X3	Average Y[j]	Exp-tal Y[j]
1	1	-1	-1	1	-84149.8	-84149.8
2	1	-1	1	-1	-153147.3	-153147.3
3	1	1	-1	-1	220730.7	220730.7
4	1	1	1	1	869816.2	869816.2

Action: Перевіряємо нуль гіпотезу та корегуємо рівняння регресії:
f3 = 4, t = [341299.92, 531137.6, 232035.2, 287233.2].
Action: Нове рівняння регресії має вигляд (нат. знач. коеф.):
y = -1056890.8 + 6639.22·x1 + 7251.1·x2 + 23936.1·x3
Action: Нове рівняння регресії має вигляд (код. знач. коеф.):
y = 213312.45 + 331961.0·x1 + 145022.0·x2 + 179520.75·x3
Action: Перевіряємо адекватність моделі.
f3 = 4, f4 = 0, Fp = inf
Action: Модель не адекватна оригіналу.
Action: Змінюємо рівняння регресії.

Action: Розглянемо лінійне рівняння регресії із врахуванням взаємодії факторів:
y = b0 + b1·x1 + b2·x2 + b3·x3 + b12·x1·x2 + b13·x1·x3 + b23·x2·x3 + b123·x1·x2·x3
Action: Маємо, N = 8, K = 8, m = 2.
Action: Складаємо матрицю планування і проведимо експерименти
Action: Дисперсія однорідна за критерієм Кохрена при m = 2:
f1 = 1, f2 = 8, Gp = 0.2932551319648094
View: Матриця планування експерименту (нат. знач. коеф., із взаємодією)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	Y1	Y2
1	1	-25	25	40	-625	-1000	1000	-25000	-84153.3	-84144.3
2	1	-25	65	25	-1625	-625	1625	-40625	-153142.3	-153150.3
3	1	75	25	25	1875	1875	625	46875	220734.7	220728.7
4	1	75	65	40	4875	3000	2600	195000	869812.7	869815.7
5	1	-25	25	25	-625	-625	625	-15625	-56045.3	-56044.3
6	1	-25	65	40	-1625	-1000	2600	-65000	-237168.3	-237163.3
7	1	75	25	40	1875	3000	1000	75000	346832.7	346827.7
8	1	75	65	25	4875	1875	1625	121875	553627.7	553637.7

View: Матриця планування експерименту (код. знач. коеф., із взаємодією)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	Y1	Y2
1	1	-1	-1	1	1	-1	-1	1	-84153.3	-84144.3
2	1	-1	1	-1	-1	1	-1	1	-153142.3	-153150.3
3	1	1	-1	-1	-1	-1	1	1	220734.7	220728.7
4	1	1	1	1	1	1	1	1	869812.7	869815.7
5	1	-1	-1	-1	1	1	1	-1	-56045.3	-56044.3
6	1	-1	1	1	-1	-1	1	-1	-237168.3	-237163.3
7	1	1	-1	1	-1	1	-1	-1	346832.7	346827.7
8	1	1	1	-1	1	-1	-1	-1	553627.7	553637.7

Action: Розраховуємо натуральні значення коефіцієнтів.
Action: Рівняння регресії має вигляд (нат. знач. коеф.):
y = -4944.3729 + 72.6921·x1 + 27.3063·x2 + 463.7698·x3 + 5.0012·x1·x2 + 0.3023·x1·x3 + 9.3069·x2·x3 + 4.1·x1·x2·x3
Action: Розраховуємо кодовані значення коефіцієнтів.
Action: Рівняння регресії має вигляд (код. знач. коеф.):
y = 182562.8875 + 315189.3125·x1 + 75720.8125·x2 + 41269.5625·x3 + 138250.4375·x1·x2 + 69300.4375·x1·x3 + 16770.9375·x2·x3 + 30749.8125·x1·x2·x3
View: Перевірка знайдених коефіцієнтів (нат. знач. коеф., із взаємодією)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	Average Y[j]	Exp-tal Y[j]
1	1	-25	25	40	-625	-1000	1000	-25000	-84148.8	-84148.8
2	1	-25	65	25	-1625	-625	1625	-40625	-153146.3	-153146.3
3	1	75	25	25	1875	1875	625	46875	220731.7	220731.7
4	1	75	65	40	4875	3000	2600	195000	869814.2	869814.2
5	1	-25	25	25	-625	-625	625	-15625	-56044.8	-56044.8
6	1	-25	65	40	-1625	-1000	2600	-65000	-237165.8	-237165.8
7	1	75	25	40	1875	3000	1000	75000	346830.2	346830.2
8	1	75	65	25	4875	1875	1625	121875	553632.7	553632.7

View: Перевірка знайдених коефіцієнтів (код. знач. коеф., із взаємодією)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	Average Y[j]	Exp-tal Y[j]
1	1	-1	-1	1	1	-1	-1	1	-84148.8	-84148.8
2	1	-1	1	-1	-1	1	-1	1	-153146.3	-153146.3
3	1	1	-1	-1	-1	-1	1	1	220731.7	220731.7
4	1	1	1	1	1	1	1	1	869814.2	869814.2
5	1	-1	-1	-1	1	1	1	-1	-56044.8	-56044.8
6	1	-1	1	1	-1	-1	1	-1	-237165.8	-237165.8
7	1	1	-1	1	-1	1	-1	-1	346830.2	346830.2
8	1	1	1	-1	1	-1	-1	-1	553632.7	553632.7

Action: Перевіряємо нуль гіпотезу та корегуємо рівняння регресії:
f3 = 8, t = [223702.2165, 386215.1217, 92783.9925, 50569.3831, 169404.2514, 84916.8288, 20550.1564, 37679.0776].
Action: Нове рівняння регресії має вигляд (нат. знач. коеф.):
y = -4944.3729 + 72.6921·x1 + 27.3063·x2 + 463.7698·x3 + 5.0012·x1·x2 + 0.3023·x1·x3 + 9.3069·x2·x3 + 4.1·x1·x2·x3
Action: Нове рівняння регресії має вигляд (код. знач. коеф.):
y = 182562.8875 + 315189.3125·x1 + 75720.8125·x2 + 41269.5625·x3 + 138250.4375·x1·x2 + 69300.4375·x1·x3 + 16770.9375·x2·x3 + 30749.8125·x1·x2·x3

Action: Перевіряємо адекватність моделі.
f3 = 8, f4 = 0, Fp = inf
Action: Модель не адекватна оригіналу.
Action: Змінюємо рівняння регресії.

Action: Розглянемо рівняння регресії із врахуванням квадратичних членів:
y = b0 + b1·x1 + b2·x2 + b3·x3 + b12·x1·x2 + b13·x1·x3 + b23·x2·x3 + b123·x1·x2·x3 + b11·x1·x1 + b22·x2·x2 + b33·x3·x3
Action: Маємо, N = 14, K = 11, m = 2.
Action: Складаємо матрицю планування і проведимо експерименти
Action: Дисперсія однорідна за критерієм Кохрена при m = 2:
f1 = 1, f2 = 14, Gp = 0.24806201550387597
View: Рототабельний композиційний план (нат. знач. коеф.)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	X1·X1	X2·X2	X3·X3	Y1	Y2
1	1	-25	25	40	-625	-1000	1000	-25000	625	625	1600	-84147.3	-84143.3
2	1	-25	65	25	-1625	-625	1625	-40625	625	4225	625	-153144.3	-153151.3
3	1	75	25	25	1875	1875	625	46875	5625	625	625	220736.7	220728.7
4	1	75	65	40	4875	3000	2600	195000	5625	4225	1600	869815.7	869812.7
5	1	-25	25	25	-625	-625	625	-15625	625	625	625	-56047.3	-56049.3
6	1	-25	65	40	-1625	-1000	2600	-65000	625	4225	1600	-237168.3	-237163.3
7	1	75	25	40	1875	3000	1000	75000	5625	625	1600	346831.7	346832.7
8	1	75	65	25	4875	1875	1625	121875	5625	4225	625	553627.7	553630.7
9	1	-61.5	45.0	32.5	-2767.5	-1998.75	1462.5	-89943.75	3782.25	2025.0	1056.25	-356259.725	-356256.725
10	1	111.5	45.0	32.5	5017.5	3623.75	1462.5	163068.75	12432.25	2025.0	1056.25	734301.675	734298.675
11	1	25.0	10.4	32.5	260.0	812.5	338.0	8450.0	625.0	108.16	1056.25	47902.373	47909.373
12	1	25.0	79.6	32.5	1990.0	812.5	2587.0	64675.0	625.0	6336.16	1056.25	309898.573	309895.573
13	1	25.0	45.0	19.525	1125.0	488.125	878.625	21965.625	625.0	2025.0	381.226	108338.879	108337.879
14	1	25.0	45.0	45.475	1125.0	1136.875	2046.375	51159.375	625.0	2025.0	2067.976	251130.349	251132.349

View: Рототабельний композиційний план (код. знач. коеф.)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	X1·X1	X2·X2	X3·X3	Y1	Y2
1	1	-1	-1	1	1	-1	-1	1	1	1	1	-84147.3	-84143.3
2	1	-1	1	-1	-1	1	-1	1	1	1	1	-153144.3	-153151.3
3	1	1	-1	-1	-1	-1	1	1	1	1	1	220736.7	220728.7
4	1	1	1	1	1	1	1	1	1	1	1	869815.7	869812.7
5	1	-1	-1	-1	1	1	1	-1	1	1	1	-56047.3	-56049.3
6	1	-1	1	1	-1	-1	1	-1	1	1	1	-237168.3	-237163.3
7	1	1	-1	1	-1	1	-1	-1	1	1	1	346831.7	346832.7
8	1	1	1	-1	1	-1	-1	-1	1	1	1	553627.7	553630.7
9	1	-1.73	0	0	-0.0	-0.0	0	-0.0	2.993	0	0	-356259.725	-356256.725
10	1	1.73	0	0	0.0	0.0	0	0.0	2.993	0	0	734301.675	734298.675
11	1	0	-1.73	0	-0.0	0	-0.0	-0.0	0	2.993	0	47902.373	47909.373
12	1	0	1.73	0	0.0	0	0.0	0.0	0	2.993	0	309898.573	309895.573
13	1	0	0	-1.73	0	-0.0	-0.0	-0.0	0	0	2.993	108338.879	108337.879
14	1	0	0	1.73	0	0.0	0.0	0.0	0	0	2.993	251130.349	251132.349

Action: Розраховуємо натуральні значення коефіцієнтів.
Action: Рівняння регресії має вигляд (нат. знач. коеф.):
y = 2028.42 + 1.01·x1 + -21.3027·x2 + -108.0412·x3 + 4.9963·x1·x2 + 0.295·x1·x3 + 9.301·x2·x3 + 4.1001·x1·x2·x3 + 1.4392·x1·x1 + 0.5419·x2·x2 + 8.8036·x3·x3
Action: Розраховуємо кодовані значення коефіцієнтів.
Action: Рівняння регресії має вигляд (код. знач. коеф.):
y = 178252.7695 + 315189.8019·x1 + 75719.8927·x2 + 41270.5271·x3 + 138249.8125·x1·x2 + 69299.9375·x1·x3 + 16770.5625·x2·x3 + 30750.8125·x1·x2·x3 + 3597.9169·x1·x1 + 216.7475·x2·x
View: Перевірка знайдених коефіцієнтів (нат. знач. коеф., із квад. членами)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	X1·X1	X2·X2	X3·X3	Average Y[j]	Exp-tal Y[j]
1	1	-25	25	40	-625	-1000	1000	-25000	625	625	1600	-84145.3	-84146.413
2	1	-25	65	25	-1625	-625	1625	-40625	625	4225	625	-153147.8	-153147.429
3	1	75	25	25	1875	1875	625	46875	5625	625	625	220732.7	220733.64
4	1	75	65	40	4875	3000	2600	195000	5625	4225	1600	869814.2	869813.985
5	1	-25	25	25	-625	-625	625	-15625	625	625	625	-56048.3	-56048.09
6	1	-25	65	40	-1625	-1000	2600	-65000	625	4225	1600	-237165.8	-237166.753
7	1	75	25	40	1875	3000	1000	75000	5625	625	1600	346832.2	346831.818
8	1	75	65	25	4875	1875	1625	121875	5625	4225	625	553629.2	553630.305
9	1	-61.5	45.0	32.5	-2767.5	-1998.75	1462.5	-89943.75	3782.25	2025.0	1056.25	-356258.225	-356257.391
10	1	111.5	45.0	32.5	5017.5	3623.75	1462.5	163068.75	12432.25	2025.0	1056.25	734300.175	734299.332
11	1	25.0	10.4	32.5	260.0	812.5	338.0	8450.0	625.0	108.16	1056.25	47905.873	47906.052
12	1	25.0	79.6	32.5	1990.0	812.5	2587.0	64675.0	625.0	6336.16	1056.25	309897.073	309896.885
13	1	25.0	45.0	19.525	1125.0	488.125	878.625	21965.625	625.0	2025.0	381.226	108338.379	108336.849
14	1	25.0	45.0	45.475	1125.0	1136.875	2046.375	51159.375	625.0	2025.0	2067.976	251131.349	251132.871

View: Перевірка знайдених коефіцієнтів (код. знач. коеф., із квад. членами)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	X1·X1	X2·X2	X3·X3	Average Y[j]	Exp-tal Y[j]
1	1	-1	-1	1	1	-1	-1	1	1	1	1	-84145.3	-84146.405
2	1	-1	1	-1	-1	1	-1	1	1	1	1	-153147.8	-153147.424
3	1	1	-1	-1	-1	-1	1	1	1	1	1	220732.7	220733.645
4	1	1	1	1	1	1	1	1	1	1	1	869814.2	869813.984
5	1	-1	-1	-1	1	1	1	-1	1	1	1	-56048.3	-56048.084
6	1	-1	1	1	-1	-1	1	-1	1	1	1	-237165.8	-237166.745
7	1	1	-1	1	-1	1	-1	-1	1	1	1	346832.2	346831.824
8	1	1	1	-1	1	-1	-1	-1	1	1	1	553629.2	553630.305
9	1	-1.73	0	0	-0.0	-0.0	0	-0.0	2.993	0	0	-356258.225	-356257.382
10	1	1.73	0	0	0.0	0.0	0	0.0	2.993	0	0	734300.175	734299.332
11	1	0	-1.73	0	-0.0	0	-0.0	-0.0	0	2.993	0	47905.873	47906.059
12	1	0	1.73	0	0.0	0	0.0	0.0	0	2.993	0	309897.073	309896.887
13	1	0	0	-1.73	0	-0.0	-0.0	-0.0	0	0	2.993	108338.379	108336.853
14	1	0	0	1.73	0	0.0	0.0	0.0	0	0	2.993	251131.349	251132.876

Action: Перевіряємо нуль гіпотезу та корегуємо рівняння регресії:
f3 = 14, t = [439439.8748, 777025.611, 186669.415, 101742.6844, 340822.0836, 170842.5398, 41343.8395, 75808.8261, 8869.8098, 534.3394, 1220.807].
Action: Нове рівняння регресії має вигляд (нат. знач. коеф.):
y = 2028.42 + 1.01·x1 + -21.3027·x2 + -108.0412·x3 + 4.9963·x1·x2 + 0.295·x1·x3 + 9.301·x2·x3 + 4.1001·x1·x2·x3 + 1.4392·x1·x1 + 0.5419·x2·x2 + 8.8036·x3·x3
Action: Нове рівняння регресії має вигляд (код. знач. коеф.):
y = 178252.7695 + 315189.8019·x1 + 75719.8927·x2 + 41270.5271·x3 + 138249.8125·x1·x2 + 69299.9375·x1·x3 + 16770.5625·x2·x3 + 30750.8125·x1·x2·x3 + 3597.9169·x1·x1 + 216.7475·x2·
Action: Перевіряємо адекватність моделі.
f3 = 14, f4 = 3, Fp = 1.5563065890499048
Action: Модель адекватна оригіналу.
Action: Виводимо результати.
View: Перевірка знайдених коефіцієнтів (нат. знач. коеф., із квад. членами)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	X1·X1	X2·X2	X3·X3	Average Y[j]	Exp-tal Y[j]
1	1	-25	25	40	-625	-1000	1000	-25000	625	625	1600	-84145.3	-84146.413
2	1	-25	65	25	-1625	-625	1625	-40625	625	4225	625	-153147.8	-153147.429
3	1	75	25	25	1875	1875	625	46875	5625	625	625	220732.7	220733.64
4	1	75	65	40	4875	3000	2600	195000	5625	4225	1600	869814.2	869813.985
5	1	-25	25	25	-625	-625	625	-15625	625	625	625	-56048.3	-56048.09
6	1	-25	65	40	-1625	-1000	2600	-65000	625	4225	1600	-237165.8	-237166.753
7	1	75	25	40	1875	3000	1000	75000	5625	625	1600	346832.2	346831.818
8	1	75	65	25	4875	1875	1625	121875	5625	4225	625	553629.2	553630.305
9	1	-61.5	45.0	32.5	-2767.5	-1998.75	1462.5	-89943.75	3782.25	2025.0	1056.25	-356258.225	-356257.391
10	1	111.5	45.0	32.5	5017.5	3623.75	1462.5	163068.75	12432.25	2025.0	1056.25	734300.175	734299.332
11	1	25.0	10.4	32.5	260.0	812.5	338.0	8450.0	625.0	108.16	1056.25	47905.873	47906.052
12	1	25.0	79.6	32.5	1990.0	812.5	2587.0	64675.0	625.0	6336.16	1056.25	309897.073	309896.885
13	1	25.0	45.0	19.525	1125.0	488.125	878.625	21965.625	625.0	2025.0	381.226	108338.379	108336.849
14	1	25.0	45.0	45.475	1125.0	1136.875	2046.375	51159.375	625.0	2025.0	2067.976	251131.349	251132.871

View: Перевірка знайдених коефіцієнтів (код. знач. коеф., із квад. членами)

№	X0	X1	X2	X3	X1·X2	X1·X3	X2·X3	X1·X2·X3	X1·X1	X2·X2	X3·X3	Average Y[j]	Exp-tal Y[j]
1	1	-1	-1	1	1	-1	-1	1	1	1	1	-84145.3	-84146.405
2	1	-1	1	-1	-1	1	-1	1	1	1	1	-153147.8	-153147.424
3	1	1	-1	-1	-1	-1	1	1	1	1	1	220732.7	220733.645
4	1	1	1	1	1	1	1	1	1	1	1	869814.2	869813.984
5	1	-1	-1	-1	1	1	1	-1	1	1	1	-56048.3	-56048.084
6	1	-1	1	1	-1	-1	1	-1	1	1	1	-237165.8	-237166.745
7	1	1	-1	1	-1	1	-1	-1	1	1	1	346832.2	346831.824
8	1	1	1	-1	1	-1	-1	-1	1	1	1	553629.2	553630.305
9	1	-1.73	0	0	-0.0	-0.0	0	-0.0	2.993	0	0	-356258.225	-356257.382
10	1	1.73	0	0	0.0	0.0	0	0.0	2.993	0	0	734300.175	734299.332
11	1	0	-1.73	0	-0.0	0	-0.0	-0.0	0	2.993	0	47905.873	47906.059
12	1	0	1.73	0	0.0	0	0.0	0.0	0	2.993	0	309897.073	309896.887
13	1	0	0	-1.73	0	-0.0	-0.0	-0.0	0	0	2.993	108338.379	108336.853
14	1	0	0	1.73	0	0.0	0.0	0.0	0	0	2.993	251131.349	251132.876

Висновки

В ході виконання лабораторної роботи було розглянуто проведення трьохфакторного експерименту при використанні рівняння регресії з урахуванням квадратичних членів (рототабельний композиційний план).

Вказане вище рівняння регресії застосовується в тому випадку, коли рівняння регресії без урахування взаємодії факторів та рівняння регресії з урахуванням взаємодії факторів є неадекватними за критерієм Фішера.

Якщо порівнювати рототабельний композиційний план та центральний ортогональний композиційний план, то вони відрізняються наявність центральних точок та значенням зоряних точок l.