Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Методи оптимізації та планування експерименту

Лабораторна робота №4

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

Виконав:
студент групи ІО-93
Варченко Є. В.
Номер у списку групи – 3

Перевірив:
ас. Регіда П. Г.

Київ – 2021

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

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

Завдання

1. Скласти матрицю планування для повного трьохфакторного експерименту.
2. Провести експеримент, повторивши N раз досліди у всіх точках факторного простору і знайти значення відгуку Y. Знайти значення Y шляхом моделювання випадкових чисел у певному діапазоні відповідно варіанту. Варіанти вибираються за номером в списку в журналі викладача.

$$y_{i\max} = 200 + x_{cp\max}$$
$$y_{i\min} = 200 + x_{cp\min}$$

де $x_{cp\max} = \frac{x_{1\max} + x_{2\max} + x_{3\max}}{3}$, $x_{cp\min} = \frac{x_{1\min} + x_{2\min} + x_{3\min}}{3}$

3. Знайти коефіцієнти рівняння регресії і записати його.
4. Провести 3 статистичні перевірки – за критеріями Кохрена, Стьюдента, Фішера.
5. Зробити висновки по адекватності регресії та значимості окремих коефіцієнтів і записати скореговане рівняння регресії.
6. Написати комп'ютерну програму, яка усе це моделює.

Варіант

№варианта	X_1		X_2		X_3	
	min	max	min	max	min	max
303	20	70	-15	45	20	35

Код програми

```
import kotlin.math.*
import kotlin.system.exitProcess

fun determinant(array: Array<DoubleArray>): Double {
    var result = 0.0

    if (array.size == 1) {
        result = array[0][0]
        return result
    }

    if (array.size == 2) {
        result = array[0][0] * array[1][1] - array[0][1] * array[1][0]
        return result
    }

    for (i in array[0].indices) {
        val temp = Array(array.size - 1) { DoubleArray(array[0].size - 1) }

        for (j in 1 until array.size) {
            for (k in array[0].indices) {
                if (k < i) {
                    temp[j - 1][k] = array[j][k]
                } else if (k > i) {
                    temp [j - 1][k - 1] = array[j][k]
                }
            }
        }

        result += array[0][i] * (-1.0).pow(i.toDouble()) * determinant(temp)
    }

    return result
}
```

```
fun main() {
    val x1Min = 20
    val x1Max = 70
    val x2Min = -15
    val x2Max = 45
    val x3Min = 20
    val x3Max = 35

    var m = 3

    val xAverageMin = (x1Min + x2Min + x3Min) / 3
    val xAverageMax = (x1Max + x2Max + x3Max) / 3

    val yMin = 200 + xAverageMin
    val yMax = 200 + xAverageMax

    var restartFlag = true

    val x = arrayOf(
        intArrayOf(1, -1, -1, -1),
        intArrayOf(1, -1, 1, 1),
        intArrayOf(1, 1, -1, 1),
        intArrayOf(1, 1, 1, -1)
    )

    val xArray = arrayOf(
        intArrayOf(-20, 30, 30),
        intArrayOf(-20, 80, 45),
        intArrayOf(30, 30, 45),
        intArrayOf(30, 80, 30))

    val aCoef = Array(3) { DoubleArray(3) }

    val mx = DoubleArray(3)
    var sum = 0.0
    var my = 0.0
```

```
    val a = DoubleArray(3)
    val yAverage = DoubleArray(4)
    val bArray = DoubleArray(4)
    val dispersionArray = DoubleArray(4)
    var f1 = 0
    var f2 = 0
    var q = 0.0

    var workFlag = true

    while (restartFlag) {
        while (workFlag) {
            val y: MutableList<DoubleArray> = ArrayList()

            println("Нормована матриця планування експерименту:")
            print("X0\tX1\tX2\tX3\t")

            for (i in 0 until m) {
                print("Y ${i + 1}\t\t\t\t")
            }

            println()

            for (i in 0..3) {
                val yTemp = DoubleArray(m)

                for (j in 0..3) {
                    print(x[i][j].toString() + "\t")
                }
                for (j in 0 until m) {
                    yTemp[j] = Math.random() * (yMax - yMin) + yMin
                    print(yTemp[j].toFloat().toString() + "\t\t")
                }
                println()
                y.add(yTemp)
            }
```

```
println("Матриця планування експерименту:")
print("X1\tX2\tX3\t")

for (i in 0 until m) {
    print("Y ${i + 1}\t\t\t\t")
}

println()

for (i in 0..3) {
    var yTemp = DoubleArray(m)

    for (j in 0..2) {
        print("${xArray[i][j]} \t")
    }

    yTemp = y[i]

    for (j in 0 until m) {
        print("${yTemp[j].toFloat()} \t\t")
    }

    println()
}

for (i in 0..3) {
    sum = 0.0

    var yTemp = DoubleArray(m)
    yTemp = y[i]

    for (j in 0 until m) {
        sum += yTemp[j]
    }

    yAverage[i] = sum / m
```

```
            }

            for (i in 0..2) {
                sum = 0.0

                for (j in 0..3) {
                    sum += xArray[j][i]
                }

                mx[i] = sum / 4
            }
            sum = 0.0
            for (i in 0..3) {
                sum += yAverage[i]
            }
            my = sum / 4
            for (i in 0..2) {
                sum = 0.0
                for (j in 0..3) {
                    sum += xArray[j][i] * yAverage[j]
                }
                a[i] = sum / 4
            }
            for (i in 0..2) {
                sum = 0.0
                for (j in 0..3) {
                    sum += Math.pow(xArray[j][i].toDouble(), 2.0)
                }
                aCoef[i][i] = sum / 4
            }
            aCoef[1][0] =
                (xArray[0][0] * xArray[0][1] + xArray[1][0] * xArray[1][1] + xArray[2][0]
* xArray[2][1] + xArray[3][0] * xArray[3][1]) / 4.0
            aCoef[0][1] = aCoef[1][0]
            aCoef[2][0] =
                (xArray[0][0] * xArray[0][2] + xArray[1][0] * xArray[1][2] + xArray[2][0]
* xArray[2][2] + xArray[3][0] * xArray[3][2]) / 4.0
            aCoef[0][2] = aCoef[2][0]
```

```
            aCoef[2][1] =
                (xArray[0][1] * xArray[0][2] + xArray[1][1] * xArray[1][2] + xArray[2][1]
* xArray[2][2] + xArray[3][1] * xArray[3][2]) / 4.0
            aCoef[1][2] = aCoef[2][1]
            val matrixTemp1 = arrayOf(
                doubleArrayOf(my, mx[0], mx[1], mx[2]),
                doubleArrayOf(a[0], aCoef[0][0], aCoef[0][1], aCoef[0][2]),
                doubleArrayOf(
                    a[1], aCoef[0][1], aCoef[1][1], aCoef[2][1]
                ),
                doubleArrayOf(a[2], aCoef[0][2], aCoef[1][2], aCoef[2][2])
            )
            val matrixTemp2 = arrayOf(
                doubleArrayOf(1.0, mx[0], mx[1], mx[2]),
                doubleArrayOf(mx[0], aCoef[0][0], aCoef[0][1], aCoef[0][2]),
                doubleArrayOf(
                    mx[1], aCoef[0][1], aCoef[1][1], aCoef[2][1]
                ),
                doubleArrayOf(mx[2], aCoef[0][2], aCoef[1][2], aCoef[2][2])
            )
            bArray[0] = determinant(matrixTemp1) / determinant(matrixTemp2)
            val matrixTemp3 = arrayOf(
                doubleArrayOf(1.0, my, mx[1], mx[2]),
                doubleArrayOf(mx[0], a[0], aCoef[0][1], aCoef[0][2]),
                doubleArrayOf(
                    mx[1], a[1], aCoef[1][1], aCoef[2][1]
                ),
                doubleArrayOf(mx[2], a[2], aCoef[1][2], aCoef[2][2])
            )
            bArray[1] = determinant(matrixTemp3) / determinant(matrixTemp2)
            val matrixTemp4 = arrayOf(
                doubleArrayOf(1.0, mx[0], my, mx[2]),
                doubleArrayOf(mx[0], aCoef[0][0], a[0], aCoef[0][2]),
                doubleArrayOf(
                    mx[1], aCoef[0][1], a[1], aCoef[2][1]
                ),
                doubleArrayOf(mx[2], aCoef[0][2], a[2], aCoef[2][2])
            )
            bArray[2] = determinant(matrixTemp4) / determinant(matrixTemp2)
```

```
            val matrixTemp5 = arrayOf(
                doubleArrayOf(1.0, mx[0], mx[1], my),
                doubleArrayOf(mx[0], aCoef[0][0], aCoef[0][1], a[0]),
                doubleArrayOf(
                    mx[1], aCoef[0][1], aCoef[1][1], a[1]
                ),
                doubleArrayOf(mx[2], aCoef[0][2], aCoef[1][2], a[2])
            )
            bArray[3] = determinant(matrixTemp5) / determinant(matrixTemp2)
            println("\nНатуралізоване рівняння регресії: ")
            System.out.printf("y = %.2f", bArray[0])
            if (bArray[1] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1", abs(bArray[1]))
            if (bArray[2] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x2", abs(bArray[2]))
            if (bArray[3] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x3\n", abs(bArray[3]))
            println("\nПеревірка: ")
            var ok = false
            for (i in 0..3) {
                ok =
                    (bArray[0] + bArray[1] * xArray[i][0] + bArray[2] * xArray[i][1] +
bArray[3] * xArray[i][2]).toFloat() == yAverage[i].toFloat()
                System.out.printf(
                    "%.2f = %.2f\n",
                    bArray[0] + bArray[1] * xArray[i][0] + bArray[2] * xArray[i][1] +
bArray[3] * xArray[i][2],
                    yAverage[i]
                )
            }
            if (ok) println("\nНатуралізовані коефіцієнти рівняння регресії b0,b1,b2,b3
визначено правильно") else println(
                "\nНатуралізовані коефіцієнти рівняння регресії b0,b1,b2,b3 визначено
неправильно"
            )
            val aNorm = DoubleArray(4)
            sum = 0.0
            for (i in 0..3) {
                sum += yAverage[i]
```

```
            }
            aNorm[0] = sum / 4.0
            aNorm[1] = bArray[1] * (x1Max - x1Min) / 2.0
            aNorm[2] = bArray[2] * (x2Max - x2Min) / 2.0
            aNorm[3] = bArray[3] * (x3Max - x3Min) / 2.0
            println("\nНормоване рівняння регресії: ")
            System.out.printf("y = %.2f", aNorm[0])
            if (aNorm[1] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1", abs(aNorm[1]))
            if (aNorm[2] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x2", abs(aNorm[2]))
            if (aNorm[3] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x3\n", abs(aNorm[3]))
            println("\nПеревірка: ")
            for (i in 0..3) {
                ok =
                    if ((aNorm[0] + aNorm[1] * x[i][1] + aNorm[2] * x[i][2] + aNorm[3] *
x[i][3]).toFloat() == yAverage[i].toFloat()
                    ) true else false
                System.out.printf(
                    "%.2f = %.2f\n",
                    aNorm[0] + aNorm[1] * x[i][1] + aNorm[2] * x[i][2] + aNorm[3] * x[i]
[3],
                    yAverage[i]
                )
            }
            if (ok) println("\nНормовані коефіцієнти рівняння регресії a0,a1,a2,a3
визначено правильно") else println(
                "\nНормовані коефіцієнти рівняння регресії a0,a1,a2,a3 визначено
неправильно"
            )



            //критерій Кохрена
            for (i in 0..2) {
                sum = 0.0
                val yTemp = y[i]
                for (j in 0 until m) {
```

```
                    sum += Math.pow(yTemp[j] - yAverage[i], 2.0)
                }
                dispersionArray[i] = sum / m
            }
            var maxDispersion = dispersionArray[0]
            for (i in 0..3) {
                if (maxDispersion < dispersionArray[i]) maxDispersion = dispersionArray[i]
            }
            var Gp = 0.0
            sum = 0.0
            for (i in 0..3) {
                sum += dispersionArray[i]
            }
            Gp = maxDispersion / sum
            f1 = m - 1
            f2 = 4
            q = 0.05
            val KohrenTable = doubleArrayOf(
                0.9065,
                0.7679,
                0.6841,
                0.6287,
                0.5892,
                0.5598,
                0.5365,
                0.5175,
                0.5017,
                0.4884,
                0.4366,
                0.372,
                0.3093,
                0.25
            )
            var Gt = 0.0
            if (f1 <= 1) Gt = KohrenTable[0] else if (f1 <= 2) Gt = KohrenTable[1] else if
(f1 <= 3) Gt =
                KohrenTable[2] else if (f1 <= 4) Gt = KohrenTable[3] else if (f1 <= 5) Gt
=
                KohrenTable[4] else if (f1 <= 6) Gt = KohrenTable[5] else if (f1 <= 7) Gt
```

```
=
                KohrenTable[6] else if (f1 <= 8) Gt = KohrenTable[7] else if (f1 <= 9) Gt
=
                KohrenTable[8] else if (f1 <= 10) Gt = KohrenTable[9] else if (f1 <= 16)
Gt =
                KohrenTable[10] else if (f1 <= 36) Gt = KohrenTable[11] else if (f1 <=
144) Gt =
                KohrenTable[12] else if (f1 > 144) Gt = KohrenTable[13]
            if (Gp < Gt) {
                System.out.printf("Gp = %.2f < Gt = %.2f\n", Gp, Gt)
                println("Дисперсії однорідні\n")
                workFlag = false
            } else {
                workFlag = true
                System.out.printf("Gp = %.2f > Gt = %.2f\n", Gp, Gt)
            }
            m++
            if (workFlag) println("ДИСПЕРСІЇ НЕОДНОРІДНІ\nПОМИЛКА : Gp > Gt \nЗБІЛЬШУЄМО
КІЛЬКІСТЬ ДОСЛІДІВ : m+1\n")
        }
        //критерій Стьюдента
        var sBetaSquareAverage = 0.0
        var sBetaS = 0.0
        var sSquareBetaS = 0.0
        sum = 0.0
        for (i in 0..3) {
            sum += dispersionArray[i]
        }
        sBetaSquareAverage = sum / 4
        sSquareBetaS = sBetaSquareAverage / (4.0 * m)
        sBetaS = Math.sqrt(sSquareBetaS)
        val beta = DoubleArray(4)
        for (i in 0..3) {
            sum = 0.0
            for (j in 0..3) {
                sum += yAverage[j] * x[j][i]
            }
            beta[i] = sum / 4
        }
```

```
        val t = DoubleArray(4)
        for (i in 0..3) {
            t[i] = abs(beta[i]) / sBetaS
        }
        var f3 = f1 * f2
        val studentTable =
            doubleArrayOf(2.306, 2.262, 2.228, 2.201, 2.179, 2.16, 2.145, 2.131, 2.12,
2.11, 2.101, 2.093, 2.086)
        if (f3 > 16) {
            println("Відсутнє значення для такого f3")
            System.exit(1)
        }
        val stNow = studentTable[f3 - 8]
        var d = 4
        if (t[0] < stNow) {
            bArray[0] = 0.0
            d--
        }
        if (t[1] < stNow) {
            bArray[1] = 0.0
            d--
        }
        if (t[2] < stNow) {
            bArray[2] = 0.0
            d--
        }
        if (t[3] < stNow) {
            bArray[3] = 0.0
            d--
        }
        println("Рівняння регресії після критерію Стьюдента: ")
        System.out.printf("y = %.2f", bArray[0])
        if (bArray[1] < 0) print(" - ") else print(" + ")
        System.out.printf("%.2f * x1", abs(bArray[1]))
        if (bArray[2] < 0) print(" - ") else print(" + ")
        System.out.printf("%.2f * x2", abs(bArray[2]))
        if (bArray[3] < 0) print(" - ") else print(" + ")
        System.out.printf("%.2f * x3\n", abs(bArray[3]))
        val yAverageAfterStudent = DoubleArray(4)
```

```
        println("\nПеревірка: ")
        for (i in 0..3) {
            System.out.printf(
                "%.2f != %.2f\n",
                (bArray[0] + bArray[1] * xArray[i][0] + bArray[2] * xArray[i][1] +
bArray[3] * xArray[i][2]).also {
                    yAverageAfterStudent[i] = it
                },
                yAverage[i]
            )
        }

        var f4 = 4 - d
        var sSquareAdequate = 0.0
        sum = 0.0
        for (i in 0..3) {
            sum += (yAverageAfterStudent[i] - yAverage[i]).pow(2.0)
        }
        sSquareAdequate = sum * (m / (4 - d))
        val Fp = sSquareAdequate / sBetaSquareAverage
        val fisherTable = arrayOf(
            doubleArrayOf(5.3, 4.5, 4.1, 3.8, 3.7, 3.6, 3.3, 3.1, 2.9),
            doubleArrayOf(4.8, 3.9, 3.5, 3.3, 3.1, 3.0, 2.7, 2.5, 2.3),
            doubleArrayOf(4.5, 3.6, 3.2, 3.0, 2.9, 2.7, 2.4, 2.2, 2.0),
            doubleArrayOf(4.4, 3.5, 3.1, 2.9, 2.7, 2.6, 2.3, 2.1, 1.9)
        )

        var fisherNow = 0.0

        if (f4 <= 1) fisherNow = fisherTable[m - 3][0] else if (f4 <= 2) fisherNow =
            fisherTable[m - 3][1] else if (f4 <= 3) fisherNow = fisherTable[m - 3][2] else
if (f4 <= 4) fisherNow =
            fisherTable[m - 3][3]

        if (Fp < fisherNow) {
            System.out.printf("\nFp = %.2f < Ft = %.2f\n", Fp, fisherNow)
        } else if (Fp > fisherNow) {
            System.out.printf("\nFp = %.2f > Ft = %.2f\n", Fp, fisherNow)
```

```
        }

        if (Fp > fisherNow) {
            println("\nРівняння регресії неадекватно оригіналу при q = 0.05")
            println(
                "Рівняння регресії з ефектом взаємодії має вигляд : y = b0 + b1*x1 + b2*x2
+ b3*x3 + " +
                        "b12*x1*x2 + b13*x1*x3 + b23*x2*x3 + b123*x1*x2*x3"
            )

            val xInteraction = arrayOf(
                doubleArrayOf(1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0, -1.0),
                doubleArrayOf(1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0),
                doubleArrayOf(1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0),
                doubleArrayOf(1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, -1.0),
                doubleArrayOf(1.0, 1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0),
                doubleArrayOf(1.0, 1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0),
                doubleArrayOf(1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0),
                doubleArrayOf(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)
            )

            val xNaturInteraction = arrayOf(
                doubleArrayOf(1.0, -20.0, 30.0, 30.0, -600.0, -600.0, 900.0, -18000.0),
                doubleArrayOf(1.0, -20.0, 30.0, 45.0, -600.0, -900.0, 1350.0, -27000.0),
                doubleArrayOf(1.0, -20.0, 80.0, 30.0, -1600.0, -600.0, 2400.0, -48000.0),
                doubleArrayOf(1.0, -20.0, 80.0, 45.0, -1600.0, -900.0, 3600.0, -72000.0),
                doubleArrayOf(1.0, 30.0, 30.0, 30.0, 900.0, 900.0, 900.0, 27000.0),
                doubleArrayOf(1.0, 30.0, 30.0, 45.0, 900.0, 1350.0, 1350.0, 40500.0),
                doubleArrayOf(1.0, 30.0, 80.0, 30.0, 2400.0, 900.0, 2400.0, 72000.0),
                doubleArrayOf(1.0, 30.0, 80.0, 45.0, 2400.0, 1350.0, 3600.0, 108000.0)
            )

            val matrixTemp = Array(8) { DoubleArray(8) }
            val kArray = DoubleArray(8)
            val yInteraction: MutableList<DoubleArray> = ArrayList()
            val yInteractionAverage = DoubleArray(8)
            val dispersionInteractionArray = DoubleArray(8)
            val mCoefMatrixInteraction = Array(8) { DoubleArray(8) }
```

```
        val bNatur = DoubleArray(8)
        val bNorm = DoubleArray(8)
        var workInteraction = true

        m = 3

        println("Нормована матриця планування експерименту з ефектом взаємодії: ")
        print("X0\tX1\tX2\tX3\tX1X2\tX1X3\tX2X3\tX1X2X3\t")
        for (i in 0 until m) {
            print("Y" + (i + 1) + "\t\t\t\t")
        }

        print("YAvr\t\t\tDisp")
        println()

        for (i in 0..7) {
            val yTemp = DoubleArray(m)

            for (j in 0..7) {
                print(xInteraction[i][j].toInt())
                if (j < 4) print("\t") else print("\t\t")
            }

            for (j in 0 until m) {
                yTemp[j] = Math.random() * (yMax - yMin) + yMin
                print("${yTemp[j].toFloat()} \t\t")
            }

            yInteraction.add(yTemp)
            sum = 0.0

            for (j in 0 until m) {
                sum += yTemp[j]
            }

            yInteractionAverage[i] = sum / m
            print("${yInteractionAverage[i].toFloat().toString()} \t\t")
```

```
            sum = 0.0

            for (k in 0 until m) {
                sum += (yTemp[k] - yInteractionAverage[i]).pow(2.0)
            }

            dispersionInteractionArray[i] = sum / m
            println(dispersionInteractionArray[i].toFloat())
        }

        for (i in 0..7) {
            for (j in 0..7) {
                sum = 0.0

                for (k in 0..7) {
                    sum += xNaturInteraction[k][i] * xNaturInteraction[k][j]
                }

                mCoefMatrixInteraction[i][j] = sum
            }
        }

        for (i in 0..7) {
            sum = 0.0
            for (j in 0..7) {
                sum += yInteractionAverage[j] * xNaturInteraction[j][i]
            }
            kArray[i] = sum
        }
        val det = determinant(mCoefMatrixInteraction)
        for (i in 0..7) {
            for (j in 0..7) {
                for (k in 0..7) {
                    matrixTemp[j][k] = mCoefMatrixInteraction[j][k]
                }
            }
            for (j in 0..7) {
```

```
                    matrixTemp[j][i] = kArray[j]
                }
                bNatur[i] = determinant(matrixTemp) / det
            }
            println("\nНатуралізоване рівняння регресії з ефектом взаємодії: ")
            System.out.printf("y = %.2f", bNatur[0])
            if (bNatur[1] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1", abs(bNatur[1]))
            if (bNatur[2] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x2", abs(bNatur[2]))
            if (bNatur[3] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x3", abs(bNatur[3]))
            if (bNatur[4] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1*x2", abs(bNatur[4]))
            if (bNatur[5] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1*x3", abs(bNatur[5]))
            if (bNatur[6] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x2*x3", abs(bNatur[6]))
            if (bNatur[7] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1*x2*x3\n", abs(bNatur[7]))
            println("\nПеревірка: ")

            var ok = false
            for (i in 0..7) {
                ok =
                    (bNatur[0] + bNatur[1] * xNaturInteraction[i][1] + bNatur[2] *
xNaturInteraction[i][2] + bNatur[3] * xNaturInteraction[i][3] + bNatur[4] *
xNaturInteraction[i][4] + bNatur[5] * xNaturInteraction[i][5] + bNatur[6] *
xNaturInteraction[i][6] + bNatur[7] * xNaturInteraction[i][7]).toFloat() ==
yInteractionAverage[i].toFloat()
                System.out.printf(
                    "%.2f = %.2f\n",
                    bNatur[0] + bNatur[1] * xNaturInteraction[i][1] + bNatur[2] *
xNaturInteraction[i][2] + bNatur[3] * xNaturInteraction[i][3] + bNatur[4] *
xNaturInteraction[i][4] + bNatur[5] * xNaturInteraction[i][5] + bNatur[6] *
xNaturInteraction[i][6] + bNatur[7] * xNaturInteraction[i][7],
                    yInteractionAverage[i]
                )
```

```
            }
            if (ok) println("\nНатуралізовані коефіцієнти рівняння регресії визначено
правильно") else println(
                "\nНатуралізовані коефіцієнти рівняння регресії визначено неправильно"
            )

            for (i in 0..7) {
                sum = 0.0
                for (j in 0..7) {
                    sum += yInteractionAverage[j] * xInteraction[j][i]
                }
                kArray[i] = sum
            }

            for (i in 0..7) {
                bNorm[i] = kArray[i] / 8.0
            }

            println("\nНормоване рівняння регресії з ефектом взаємодії: ")

            System.out.printf("y = ${bNorm[0]}")
            if (bNorm[1] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[1])} * x1")
            if (bNorm[2] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[2])} * x2")
            if (bNorm[3] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[3])} * x3")
            if (bNorm[4] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[4])} * x1*x2")
            if (bNorm[5] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[5])} * x1*x3")
            if (bNorm[6] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[6])} * x2*x3")
            if (bNorm[7] < 0) print(" - ") else print(" + ")
            System.out.printf("${abs(bNorm[7])} * x1*x2*x3\n")

            println("\nПеревірка: ")
            ok = false
```

```
            for (i in 0..7) {
                ok = (bNorm[0] + bNorm[1] * xInteraction[i][1] + bNorm[2] *
xInteraction[i][2] + bNorm[3] * xInteraction[i][3] + bNorm[4] * xInteraction[i][4] +
bNorm[5] * xInteraction[i][5] + bNorm[6] * xInteraction[i][6] + bNorm[7] * xInteraction[i]
[7]).toFloat() == yInteractionAverage[i].toFloat()
                System.out.printf(
                    "%.2f = %.2f\n",
                    bNorm[0] + bNorm[1] * xInteraction[i][1] + bNorm[2] * xInteraction[i]
[2] + bNorm[3] * xInteraction[i][3] + bNorm[4] * xInteraction[i][4] + bNorm[5] *
xInteraction[i][5] + bNorm[6] * xInteraction[i][6] + bNorm[7] * xInteraction[i][7],
                    yInteractionAverage[i]
                )
            }

            if (ok) {
                println("\nНормовані коефіцієнти рівняння регресії b0, b1, b2, b3, b12,
b13, b23, b123 визначено правильно")
            } else {
                println("\nНормовані коефіцієнти рівняння регресії b0, b1, b2, b3, b12,
b13, b23, b123 визначено неправильно")
            }

            var maxDispersionInteraction = dispersionInteractionArray[0]
            for (i in 0..3) {
                if (maxDispersionInteraction < dispersionInteractionArray[i])
maxDispersionInteraction =
                    dispersionInteractionArray[i]
            }

            var Gp = 0.0
            sum = 0.0

            for (i in 0..3) {
                sum += dispersionInteractionArray[i]
            }
```

```
            Gp = maxDispersionInteraction / sum
            f1 = m - 1
            f2 = 8
            q = 0.05

            val KohrenTableInteraction = doubleArrayOf(0.6798, 0.5157, 0.4377, 0.391,
0.3595, 0.3362, 0.3185, 0.3043, 0.2926, 0.2829, 0.2462, 0.2022, 0.1616, 0.125)

            var Gt = 0.0

            if (f1 <= 1) Gt = KohrenTableInteraction[0] else if (f1 <= 2) Gt =
                KohrenTableInteraction[1] else if (f1 <= 3) Gt = KohrenTableInteraction[2]
else if (f1 <= 4) Gt =
                KohrenTableInteraction[3] else if (f1 <= 5) Gt = KohrenTableInteraction[4]
else if (f1 <= 6) Gt =
                KohrenTableInteraction[5] else if (f1 <= 7) Gt = KohrenTableInteraction[6]
else if (f1 <= 8) Gt =
                KohrenTableInteraction[7] else if (f1 <= 9) Gt = KohrenTableInteraction[8]
else if (f1 <= 10) Gt =
                KohrenTableInteraction[9] else if (f1 <= 16) Gt =
KohrenTableInteraction[10] else if (f1 <= 36) Gt =
                KohrenTableInteraction[11] else if (f1 <= 144) Gt =
                KohrenTableInteraction[12] else if (f1 > 144) Gt =
KohrenTableInteraction[13]
            if (Gp < Gt) {
                System.out.printf("Gp = %.2f < Gt = %.2f\n", Gp, Gt)
                println("Дисперсії однорідні\n")
                workInteraction = false
            } else {
                workInteraction = true
                System.out.printf("Gp = %.2f > Gt = %.2f\n", Gp, Gt)
            }

            m++

            if (workInteraction) {
                println("Неоднорідні дисперсії\nПомилка: Gp > Gt\nЗбільшуємо кількість
дослідів: m + 1\n")
```

```
            }

            var sBetaSquareAverageInteraction = 0.0
            var sBetaSInteraction = 0.0
            var sSquareBetaSInteraction = 0.0

            sum = 0.0

            for (i in 0..7) {
                sum += dispersionInteractionArray[i]
            }

            sBetaSquareAverageInteraction = sum / 8
            sSquareBetaSInteraction = sBetaSquareAverageInteraction / (8.0 * m)
            sBetaSInteraction = sqrt(sSquareBetaSInteraction)

            val betaInteraction = DoubleArray(8)
            for (i in 0..7) {
                sum = 0.0
                for (j in 0..7) {
                    sum += yInteractionAverage[j] * xInteraction[j][i]
                }
                betaInteraction[i] = sum / 8
            }

            val tInteraction = DoubleArray(8)

            for (i in 0..7) {
                tInteraction[i] = abs(betaInteraction[i]) / sBetaSInteraction
            }

            f3 = f1 * f2

            val studentTableInteraction = doubleArrayOf(2.12, 2.11, 2.101, 2.093, 2.086,
2.08, 2.074, 2.069, 2.064, 2.06, 2.056)

            if (f3 > 24) {
```

```
        println("Відсутнє значення для такого f3")
        exitProcess(1)
    }

    val stInteractionNow = studentTableInteraction[f3 - 16]
    d = 8

    if (tInteraction[0] < stInteractionNow) {
        bNatur[0] = 0.0
        d--
    }
    if (tInteraction[1] < stInteractionNow) {
        bNatur[1] = 0.0
        d--
    }
    if (tInteraction[2] < stInteractionNow) {
        bNatur[2] = 0.0
        d--
    }
    if (tInteraction[3] < stInteractionNow) {
        bNatur[3] = 0.0
        d--
    }
    if (tInteraction[4] < stInteractionNow) {
        bNatur[4] = 0.0
        d--
    }
    if (tInteraction[5] < stInteractionNow) {
        bNatur[5] = 0.0
        d--
    }
    if (tInteraction[6] < stInteractionNow) {
        bNatur[6] = 0.0
        d--
    }
    if (tInteraction[7] < stInteractionNow) {
        bNatur[7] = 0.0
        d--
```

```
            }

            println("Рівняння регресії після критерію Стьюдента з ефектом взаємодії: ")
            System.out.printf("y = ${bNatur[0]}")

            if (bNatur[1] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1", abs(bNatur[1]))
            if (bNatur[2] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x2", abs(bNatur[2]))
            if (bNatur[3] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x3", abs(bNatur[3]))
            if (bNatur[4] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1*x2", abs(bNatur[4]))
            if (bNatur[5] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1*x3", abs(bNatur[5]))
            if (bNatur[6] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x2*x3", abs(bNatur[6]))
            if (bNatur[7] < 0) print(" - ") else print(" + ")
            System.out.printf("%.2f * x1*x2*x3\n", abs(bNatur[7]))
            val yAverageAfterStudentInteraction = DoubleArray(8)
            println("\nПеревірка: ")

            for (i in 0..7) {
                System.out.printf(
                    "%.2f != %.2f\n",
                    (bNatur[0] + bNatur[1] * xNaturInteraction[i][1] + bNatur[2] *
xNaturInteraction[i][2] + bNatur[3] * xNaturInteraction[i][3] + bNatur[4] *
xNaturInteraction[i][4] + bNatur[5] * xNaturInteraction[i][5] + bNatur[6] *
xNaturInteraction[i][6] + bNatur[7] * xNaturInteraction[i][7]).also {
                        yAverageAfterStudentInteraction[i] = it
                    },
                    yInteractionAverage[i]
                )
            }

            f4 = 8 - d

            var sSquareAdequateInteraction = 0.0
```

```
            sum = 0.0

            for (i in 0..7) {
                sum += (yAverageAfterStudentInteraction[i] -
yInteractionAverage[i]).pow(2.0)
            }

            sSquareAdequateInteraction = sum * (m / (8 - d).toDouble())

            val FpInteraction = sSquareAdequateInteraction / sBetaSquareAverageInteraction

            val fisherTableInteraction = arrayOf(
                doubleArrayOf(4.5, 3.6, 3.2, 3.0, 2.9, 2.7, 2.4, 2.2, 2.0),
                doubleArrayOf(4.3, 3.4, 3.0, 2.8, 2.6, 2.5, 2.2, 2.0, 1.7),
                doubleArrayOf(4.1, 3.2, 2.9, 2.6, 2.5, 2.3, 2.0, 1.8, 1.5)
            )

            var fisherIntercationNow = 0.0

            if (f4 <= 1) fisherIntercationNow =
                fisherTableInteraction[m - 3][0] else if (f4 <= 2) fisherIntercationNow =
                fisherTableInteraction[m - 3][1] else if (f4 <= 3) fisherIntercationNow =
                fisherTableInteraction[m - 3][2] else if (f4 <= 4) fisherIntercationNow =
                fisherTableInteraction[m - 3][3] else if (f4 <= 5) fisherIntercationNow =
                fisherTableInteraction[m - 3][4] else if (f4 <= 6) fisherIntercationNow =
                fisherTableInteraction[m - 3][5] else if (f4 <= 12) fisherIntercationNow =
                fisherTableInteraction[m - 3][6]

            if (FpInteraction < fisherIntercationNow) {
                System.out.printf("\nFp = %.2f < Ft = %.2f\n", FpInteraction,
fisherIntercationNow)
            } else if (FpInteraction > fisherIntercationNow) {
                System.out.printf("\nFp = %.2f > Ft = %.2f\n", FpInteraction,
fisherIntercationNow)
            }

            if (FpInteraction > fisherIntercationNow) {
```

```
                println("\nРівняння регресії з ефектом взаємодії неадекватно оригіналу при
q = 0.05")
                m = 3
                workFlag = true
            } else if (FpInteraction < fisherIntercationNow) {
                println("\nРівняння регресії з ефектом взаємодії адекватно оригіналу при q
= 0.05")
                restartFlag = false
            }
        } else {
            println("\nРівняння регресії адекватно оригіналу при q = 0.05")
            restartFlag = false
        }
    }
}
```

Результати роботи програми

```
Нормована матриця планування експерименту:
X0  X1  X2  X3  Y 1             Y 2             Y 3
1   -1  -1  -1  240.79579       226.23586       246.30264
1   -1  1   1   246.58563       221.11037       208.69092
1   1   -1  1   238.28564       221.12718       237.72282
1   1   1   -1  233.42284       217.37105       209.959
Матриця планування експерименту:
X1  X2  X3  Y 1             Y 2             Y 3
-20     30  30  240.79579       226.23586       246.30264
-20     80  45  246.58563       221.11037       208.69092
30  30  45  238.28564       221.12718       237.72282
30  80  30  233.42284       217.37105       209.959

Натуралізоване рівняння регресії:
y = 243.18 - 0.11 * x1 - 0.24 * x2 - 0.01 * x3

Перевірка:
237.78 = 237.78
225.46 = 225.46
232.38 = 232.38
220.25 = 220.25

Натуралізовані коефіцієнти рівняння регресії b0,b1,b2,b3 визначено правильно

Нормоване рівняння регресії:
y = 228.97 - 2.65 * x1 - 7.33 * x2 - 0.05 * x3

Перевірка:
239.00 = 237.78
224.24 = 225.46
233.60 = 232.38
219.03 = 220.25

Нормовані коефіцієнти рівняння регресії a0,a1,a2,a3 визначено неправильно
Gp = 0.65 < Gt = 0.77
Дисперсії однорідні
```

```
Рівняння регресії після критерію Стьюдента:
y = 243.18 + 0.00 * x1 - 0.24 * x2 + 0.00 * x3

Перевірка:
235.84 != 237.78
223.62 != 225.46
235.84 != 232.38
223.62 != 220.25

Fp = 0.64 < Ft = 3.90

Рівняння регресії адекватно оригіналу при q = 0.05

Process finished with exit code 0
```

Висновки

- Ознайомилися з темою роботи.
- Були здобуті необхідні навички для виконання завдань.
- Розроблено програму, яка виконує поставлену задачу.
- Вище приведені результати свідчать про успішне виконання умов завдань.
- Основну мету лабораторної роботи було досягнуто.