Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №3

з дисципліни МОПЕ

на тему:

# «ПРОВЕДЕННЯ ТРЬОХФАКТОРНОГО ЕКСПЕРИМЕНТУ З ВИКОРИСТАННЯМ ЛІНІЙНОГО РІВНЯННЯ РЕГРЕСІЇ»

Виконав:

студент групи ІВ-83

Денисюк Тарас

Варіант: 309

Перевірив:

Регіда П.Г.

Київ 2020



Варіант:

X1min = -20 X2min = -35 X3min = 10

X1max = 15 X2max = 10 X3max = 10

Код програми

import org.apache.commons.math3.linear.Array2DRowRealMatrix  
import org.apache.commons.math3.linear.LUDecomposition  
import kotlin.math.absoluteValue  
import kotlin.math.pow  
import kotlin.random.Random  
  
fun main() {  
  
 val x1min = -20  
 val x1max = 15  
 val x2min = -35  
 val x2max = 10  
 val x3min = 10  
 val x3max = 20  
 val xAvmax: Double = (x1max + x2max + x3max) / 3.0  
 val xAvmin: Double = (x1min + x2min + x3min) / 3.0  
 val ymax = 200 + xAvmax.toInt()  
 val ymin = 200 + xAvmin.toInt()  
 val random = *Random*(1)  
  
 val x1 = *arrayOf*(x1min, x1min, x1max, x1max)  
 val x2 = *arrayOf*(x2min, x2max, x2min, x2max)  
 val x3 = *arrayOf*(x3min, x3max, x3max, x3min)  
  
  
 val y1 = Array(4) **{** random.nextInt(ymin, ymax) **}** val y2 = Array(4) **{** random.nextInt(ymin, ymax) **}** val y3 = Array(4) **{** random.nextInt(ymin, ymax) **}** *println*("Кодовані значення Х:")  
 *println*(  
 "№ X1 X2 X3\n" +  
 "1 -1 -1 -1\n" +  
 "2 -1 +1 +1\n" +  
 "3 +1 -1 +1\n" +  
 "4 +1 +1 -1\n"  
 )  
 *println*("Матриця для m = 3:")  
 *println*(  
 "№ X1 X2 X3 Y1 Y2 Y3\n" +  
 "1 $x1min $x2min $x3min ${y1[0]} ${y2[0]} ${y3[0]}\n" +  
 "2 $x1min $x2max $x3max ${y1[1]} ${y2[1]} ${y3[1]}\n" +  
 "3 $x1max $x2min $x3max ${y1[2]} ${y2[2]} ${y3[2]}\n" +  
 "4 $x1max $x2max $x3min ${y1[3]} ${y2[3]} ${y3[3]}"  
 )  
  
 ///////  
 //////  
 val y1av1 = (y1[0] + y2[0] + y3[0]) / 3.0  
 val y2av2 = (y1[1] + y2[1] + y3[1]) / 3.0  
 val y3av3 = (y1[2] + y2[2] + y3[2]) / 3.0  
 val y4av4 = (y1[3] + y2[3] + y3[3]) / 3.0  
  
 val mx1 = x1.*sum*() / 4.0  
 val mx2 = x2.*sum*() / 4.0  
 val mx3 = x3.*sum*() / 4.0  
  
 val my = (y1av1 + y2av2 + y3av3 + y4av4) / 4.0  
  
 val a1 = (x1[0] \* y1av1 + x1[1] \* y2av2 + x1[2] \* y3av3 + x1[3] \* y4av4) / 4.0  
 val a2 = (x2[0] \* y1av1 + x2[1] \* y2av2 + x2[2] \* y3av3 + x2[3] \* y4av4) / 4.0  
 val a3 = (x3[0] \* y1av1 + x3[1] \* y2av2 + x3[2] \* y3av3 + x3[3] \* y4av4) / 4.0  
  
 val a11 = (x1[0].*square*() + x1[1].*square*() + x1[2].*square*() + x1[3].*square*()) / 4.0  
 val a22 = (x2[0].*square*() + x2[1].*square*() + x2[2].*square*() + x2[3].*square*()) / 4.0  
 val a33 = (x3[0].*square*() + x3[1].*square*() + x3[2].*square*() + x3[3].*square*()) / 4.0  
 val a12 = (x1[0] \* x2[0] + x1[1] \* x2[1] + x1[2] \* x2[2] + x1[3] \* x2[3]) / 4.0  
 val a21 = (x1[0] \* x2[0] + x1[1] \* x2[1] + x1[2] \* x2[2] + x1[3] \* x2[3]) / 4.0  
 val a13 = (x1[0] \* x3[0] + x1[1] \* x3[1] + x1[2] \* x3[2] + x1[3] \* x3[3]) / 4.0  
 val a31 = (x1[0] \* x3[0] + x1[1] \* x3[1] + x1[2] \* x3[2] + x1[3] \* x3[3]) / 4.0  
 val a23 = (x2[0] \* x3[0] + x2[1] \* x3[1] + x2[2] \* x3[2] + x2[3] \* x3[3]) / 4.0  
 val a32 = (x2[0] \* x3[0] + x2[1] \* x3[1] + x2[2] \* x3[2] + x2[3] \* x3[3]) / 4.0  
  
 val b01 = *arrayOf*(  
 *doubleArrayOf*(1.0, mx1.toDouble(), mx2.toDouble(), mx3.toDouble()),  
 *doubleArrayOf*(mx1.toDouble(), a11.toDouble(), a12.toDouble(), a13.toDouble()),  
 *doubleArrayOf*(mx2.toDouble(), a12.toDouble(), a22.toDouble(), a23.toDouble()),  
 *doubleArrayOf*(mx3.toDouble(), a13.toDouble(), a23.toDouble(), a33.toDouble())  
 )  
 val b02 = *arrayOf*(  
 *doubleArrayOf*(my.toDouble(), mx1.toDouble(), mx2.toDouble(), mx3.toDouble()),  
 *doubleArrayOf*(a1.toDouble(), a11.toDouble(), a12.toDouble(), a13.toDouble()),  
 *doubleArrayOf*(a2.toDouble(), a12.toDouble(), a22.toDouble(), a23.toDouble()),  
 *doubleArrayOf*(a3.toDouble(), a13.toDouble(), a23.toDouble(), a33.toDouble())  
 )  
  
 val b03 = *arrayOf*(  
 *doubleArrayOf*(1.0, my.toDouble(), mx2.toDouble(), mx3.toDouble()),  
 *doubleArrayOf*(mx1.toDouble(), a1.toDouble(), a12.toDouble(), a13.toDouble()),  
 *doubleArrayOf*(mx2.toDouble(), a2.toDouble(), a22.toDouble(), a23.toDouble()),  
 *doubleArrayOf*(mx3.toDouble(), a3.toDouble(), a23.toDouble(), a33.toDouble())  
 )  
  
 val b04 = *arrayOf*(  
 *doubleArrayOf*(1.0, mx1.toDouble(), my.toDouble(), mx3.toDouble()),  
 *doubleArrayOf*(mx1.toDouble(), a11.toDouble(), a1.toDouble(), a13.toDouble()),  
 *doubleArrayOf*(mx2.toDouble(), a12.toDouble(), a2.toDouble(), a23.toDouble()),  
 *doubleArrayOf*(mx3.toDouble(), a13.toDouble(), a3.toDouble(), a33.toDouble())  
 )  
  
 val b05 = *arrayOf*(  
 *doubleArrayOf*(1.0, mx1.toDouble(), mx2.toDouble(), my.toDouble()),  
 *doubleArrayOf*(mx1.toDouble(), a11.toDouble(), a12.toDouble(), a1.toDouble()),  
 *doubleArrayOf*(mx2.toDouble(), a12.toDouble(), a22.toDouble(), a2.toDouble()),  
 *doubleArrayOf*(mx3.toDouble(), a13.toDouble(), a23.toDouble(), a3.toDouble())  
 )  
  
 var b0 =  
 LUDecomposition(Array2DRowRealMatrix(b02)).*determinant* / LUDecomposition(Array2DRowRealMatrix(b01)).*determinant* var b1 =  
 LUDecomposition(Array2DRowRealMatrix(b03)).*determinant* / LUDecomposition(Array2DRowRealMatrix(b01)).*determinant* var b2 =  
 LUDecomposition(Array2DRowRealMatrix(b04)).*determinant* / LUDecomposition(Array2DRowRealMatrix(b01)).*determinant* var b3 =  
 LUDecomposition(Array2DRowRealMatrix(b05)).*determinant* / LUDecomposition(Array2DRowRealMatrix(b01)).*determinant  
  
 println*("Середнє значення відгуку функції за рядками:")  
 *println*("y1av1 = ${b0 + b1 \* x1[0] + b2 \* x2[0] + b3 \* x3[0]} = $y1av1")  
 *println*("y2av2 = ${b0 + b1 \* x1[1] + b2 \* x2[1] + b3 \* x3[1]} = $y2av2")  
 *println*("y3av3 = ${b0 + b1 \* x1[2] + b2 \* x2[2] + b3 \* x3[2]} = $y3av3")  
 *println*("y4av4 = ${b0 + b1 \* x1[3] + b2 \* x2[3] + b3 \* x3[3]} = $y4av4")  
 *println*("Значення співпадають")  
 ///////  
 //////  
  
 val d1 = *getDispersion*(y1, y2, y3, y1av1, y2av2, y3av3, 0)  
 val d2 = *getDispersion*(y1, y2, y3, y1av1, y2av2, y3av3, 1)  
 val d3 = *getDispersion*(y1, y2, y3, y1av1, y2av2, y3av3, 2)  
 val d4 = *getDispersion*(y1, y2, y3, y1av1, y2av2, y3av3, 3)  
 *println*("Дисперсія по рядкам")  
 *println*("d1= $d1, d2 = $d2, d3 = $d3, d4 = $d4")  
  
 val dispertion = *arrayOf*(d1, d2, d3, d4)  
  
 val m = 3  
 val gp = dispertion.*max*()!! / dispertion.*sum*()  
 val f1 = m - 1  
 val f2 = 4  
 val n = 4  
 val gt = 0.7679  
 if (*isHomogeus*(gt, gp))  
 *print*("Дисперсія однорідна")  
 else  
 *print*("Дисперсія неоднорідна")  
 ///////  
 //////  
  
 val sigmaB = dispertion.*sum*() / n  
 val sigmaSbs = sigmaB / n \* m  
 val sigmaBs = sigmaSbs.*pow*(0.5)  
  
 val beta0 = (y1av1 \* 1 + y2av2 \* 1 + y3av3 \* 1 + y4av4 \* 1) / 4  
 val beta1 = (y1av1 \* (-1) + y2av2 \* (-1) + y3av3 \* 1 + y4av4 \* 1) / 4  
 val beta2 = (y1av1 \* (-1) + y2av2 \* 1 + y3av3 \* (-1) + y4av4 \* 1) / 4  
 val beta3 = (y1av1 \* (-1) + y2av2 \* 1 + y3av3 \* 1 + y4av4 \* (-1)) / 4  
  
 val t0 = beta0.*absoluteValue* / sigmaBs  
 val t1 = beta1.*absoluteValue* / sigmaBs  
 val t2 = beta2.*absoluteValue* / sigmaBs  
 val t3 = beta3.*absoluteValue* / sigmaBs  
  
 val f3 = f1 \* f2  
 val ttabl = 2.306  
 *println*("\nКритерій Стьюдента")  
 *println*("f3 = f1 \* f2, з таблиці tтабл = 2.306")  
 if (t0 < ttabl) {  
 *println*("t0 < ttabl, b0 не значимий")  
 b0 = 0.0  
 }  
 if (t1 < ttabl) {  
 *println*("t1 < ttabl, b1 не значимий")  
 b1 = 0.0  
 }  
 if (t2 < ttabl) {  
 *println*("t2 < ttabl, b2 не значимий")  
 b2 = 0.0  
 }  
 if (t3 < ttabl) {  
 *println*("t3 < ttabl, b3 не значимий")  
 b3 = 0.0  
 }  
  
 val yy1 = b0 + b1 \* x1min + b2 \* x2min + b3 \* x3min  
 val yy2 = b0 + b1 \* x1min + b2 \* x2max + b3 \* x3max  
 val yy3 = b0 + b1 \* x1max + b2 \* x2min + b3 \* x3max  
 val yy4 = b0 + b1 \* x1max + b2 \* x2max + b3 \* x3min  
 ///////  
 //////  
  
 *println*("Критерій Фішера")  
 val d = 1  
 *println*("$d значимих коефіцієнтів")  
 val f4 = n - d  
  
 val sad =  
 ((yy1 - y1av1).*pow*(2) + (yy2 - y2av2).*pow*(2) + (yy3 - y3av3).*pow*(2) + (yy4 - y4av4).*pow*(2)) \* (m / (n - d))  
 val fp = sad / sigmaB  
 *println*("d1 = $d1, d2 = $d2, d3 = $d3, d4 = $d4, d5 = $sigmaB")  
 *println*("Fp = $fp")  
 *println*("Ft берем із таблиці, Ft = 4.1")  
 val ft = 4.1  
 if (*isAdequate*(fp, ft))  
 *print*("Fp = $fp < Ft, Рівняння адекватно оригіналу")  
 else  
 *print*("Fp = $fp > Ft, Рівняння неадекватно оригіналу")  
}  
  
private fun Int.square(): Int = this \* this  
  
private fun getDispersion(y1: Array<Int>, y2: Array<Int>, y3: Array<Int>, y1av1: Double, y2av2: Double, y3av3: Double, num: Int) =  
 ((y1[num] - y1av1).*pow*(2) + (y2[num] - y2av2).*pow*(2) + (y3[num] - y3av3).*pow*(2)) / 3  
  
private fun isHomogeus(gt: Double, gp: Double) = gp < gt  
  
private fun isAdequate(fp: Double, ft: Double) = fp < ft

Результати роботи

