Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №6

з дисципліни «Методи оптимізації та планування експерименту»

# «Проведення трьохфакторного експерименту при використанні рівняння регресії з квадратичними членами»

Виконав:

студент групи ІО-83

Черноштан О.С.

Варіант: 326

Перевірив:

Регіда П.Г.

Київ 2020

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

# Завдання:

1. Ознайомитися з теоретичними відомостями.

2. Вибрати з таблиці варіантів і записати в протокол інтервали значень х1, х2, х3. Обчислити і записати значення, відповідні кодованим значенням

факторів

3. Значення функції відгуку знайти за допомогою підстановки в формулу:

yi = f(х1, х2, х3) + random(10)-5,

де f(х1, х2, х3) вибирається по номеру в списку в журналі викладача.

4. Провести експерименти і аналізуючи значення статистичних перевірок, отримати адекватну модель рівняння регресії. При розрахунках

використовувати натуральні значення факторів.

5. Зробити висновки по виконаній роботі.

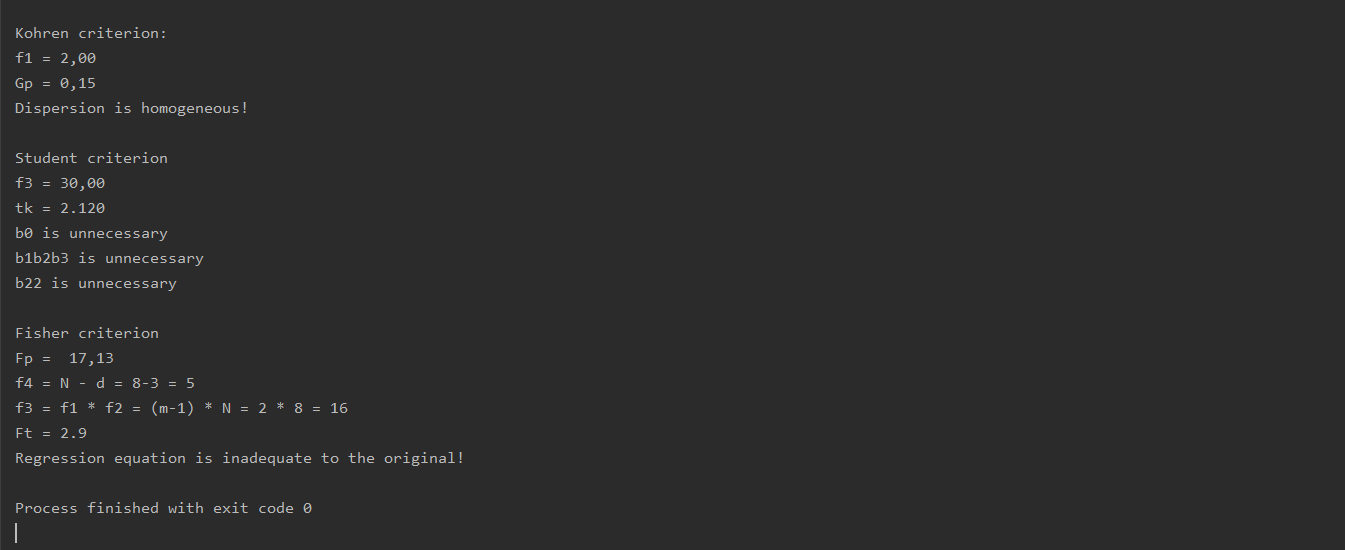
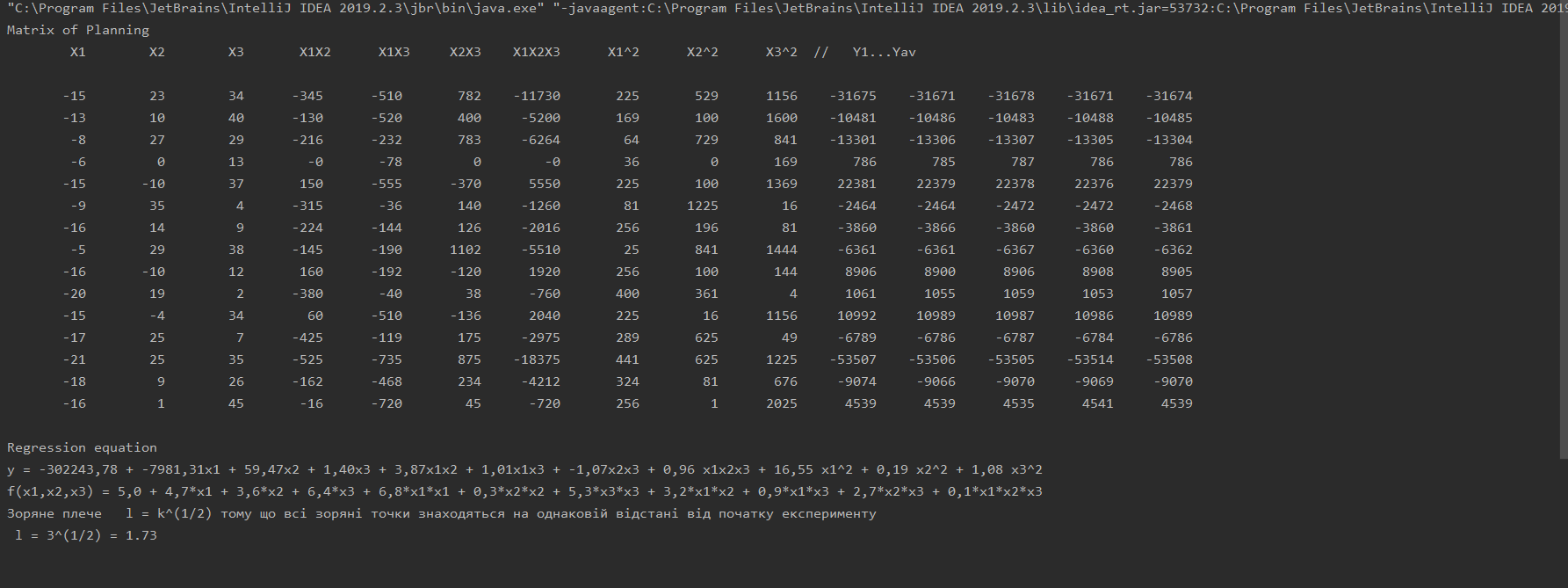
**Варіант №326:**



**Роздруківка коду програми:**

import Jama.Matrix;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Iterator;  
  
public class S6Lab {  
  
 private static final int *x1min* = -25;  
 private static final int *x1max* = -5;  
 private static final int *x2min* = -15;  
 private static final int *x2max* = 35;  
 private static final int *x3min* = -5;  
 private static final int *x3max* = 60;  
  
 private static int *m* = 3;  
 private static double[][] *mainMatrix* = new double[15][12+*m*];  
  
 private static void generateMatrix() {  
 for (int i = 0; i < *mainMatrix*.length; i++) {  
 *mainMatrix*[i][0] = (int) (Math.*random*() \* (*x1max* - *x1min*)) + *x1min* + 1;  
 *mainMatrix*[i][1] = (int) (Math.*random*() \* (*x2max* - *x2min*)) + *x2min* + 1;  
 *mainMatrix*[i][2] = (int) (Math.*random*() \* (*x3max* - *x3min*)) + *x3min* + 1;  
 *mainMatrix*[i][3] = *mainMatrix*[i][0] \* *mainMatrix*[i][1];  
 *mainMatrix*[i][4] = *mainMatrix*[i][0] \* *mainMatrix*[i][2]; //X's  
 *mainMatrix*[i][5] = *mainMatrix*[i][1] \* *mainMatrix*[i][2];  
 *mainMatrix*[i][6] = *mainMatrix*[i][0] \* *mainMatrix*[i][1] \* *mainMatrix*[i][2];  
 *mainMatrix*[i][7] = *mainMatrix*[i][0] \* *mainMatrix*[i][0];  
 *mainMatrix*[i][8] = *mainMatrix*[i][1] \* *mainMatrix*[i][1];  
 *mainMatrix*[i][9] = *mainMatrix*[i][2] \* *mainMatrix*[i][2];  
  
 }  
 for (int i = 0; i < *mainMatrix*.length; i++) {  
 for (int j = 10; j < *mainMatrix*[i].length-1; j++) {  
 *mainMatrix*[i][j] = (int)(Math.*random*()\*10) - 5 + 1 + 5.4 + 0.4\**mainMatrix*[i][0] + 1.5\**mainMatrix*[i][1] + 9.3\**mainMatrix*[i][2] + 9.5\**mainMatrix*[i][7] + 0.3\**mainMatrix*[i][8] + 2.2\**mainMatrix*[i][9] + 1.1\**mainMatrix*[i][3] + 0.7\**mainMatrix*[i][4] + 9.3\**mainMatrix*[i][5] + 3.7\**mainMatrix*[i][6];  
 }  
 }  
  
 for(int i = 0; i < *mainMatrix*.length; i++){  
 for(int j = 10; j < *mainMatrix*[i].length -1 ; j++){  
 *mainMatrix*[i][*mainMatrix*[i].length -1] += *mainMatrix*[i][j]/(*m*+1);  
 }  
 }  
 }  
  
 private static void getMatrix() {  
 System.*out*.println("Matrix of Planning");  
 System.*out*.print(" X1 X2 X3 X1X2 X1X3 X2X3 X1X2X3 X1^2 X2^2 X3^2 // Y1...Yav\n");  
 new Matrix(*mainMatrix*).print(8, 0);  
 }  
  
  
  
  
 private static Matrix generateStandartMatrix(){  
 double m00 = 0,m10 = 0,m20 = 0,m30 = 0 ,m40 = 0 ,m50 = 0,m60 = 0 ,m70 = 0 ,m80 = 0,m90 = 0,m100=0,m01 = 0,m11 = 0,m21 = 0,m31 = 0  
 ,m41 = 0,m51 = 0,m61 = 0,m71 = 0,m81 = 0,m91= 0,m101 = 0,m02 = 0,m12 = 0,m22 = 0,m32 = 0,  
 m42= 0 ,m52 = 0,m62 = 0,m72 = 0,m82 = 0,m92 = 0, m102 = 0,m03 = 0,m13 = 0,m23 = 0,m33 = 0,m43 = 0,m53 = 0,m63 = 0 ,m73 = 0,  
 m83 = 0, m93 =0, m103 = 0, m04 = 0,m14 = 0,m24 = 0  
 ,m34 = 0,m44 = 0,m54 = 0,m64 = 0,m74 =0 ,m84 = 0, m94 = 0, m104 = 0,m05 = 0 ,m15 = 0,m25 = 0,m35 =0,m45 =0,m55 =0,m65 =0,m75 =0,  
 m85 = 0, m95 = 0, m105 = 0, m06 = 0,m16 = 0,m26 = 0,m36 = 0,m46 = 0,m56 = 0,m66 = 0,m76 = 0,m86=0,m96=0,m106=0,m07 = 0,m17 = 0,m27 = 0,m37 = 0,m47 = 0,m57 = 0,m67 = 0,m77 = 0,  
 m87=0,m97=0,m107=0,m08=0,m18=0,m28=0,m38=0,m48=0,m58=0,m68=0,m78=0,m88=0,m98=0,m108=0,m09=0,m19=0,m29=0,m39=0,m49=0,m59=0,m69=0,m79=0  
 ,m89=0,m99=0,m109=0,m010=0,m110=0,m210=0,m310=0,m410=0,m510=0,m610=0,m710=0,m810=0,m910=0,m1010=0;  
  
 for(int i = 0; i < *mainMatrix*.length; i++){  
 for(int j = 0; j < *mainMatrix*[i].length; j++){  
 m10+=*mainMatrix*[i][0];  
 m20+=*mainMatrix*[i][1];  
 m30+=*mainMatrix*[i][2];  
 m40+=*mainMatrix*[i][3];  
 m50+=*mainMatrix*[i][4];  
 m60+=*mainMatrix*[i][5];  
 m70+=*mainMatrix*[i][6];  
  
  
 m01+=*mainMatrix*[i][0];  
 m11+=*mainMatrix*[i][0]\**mainMatrix*[i][0];  
 m21+=*mainMatrix*[i][3];  
 m31+=*mainMatrix*[i][4];  
 m41+=Math.*pow*(*mainMatrix*[i][0],2)\**mainMatrix*[i][1];  
 m51+=Math.*pow*(*mainMatrix*[i][0],2)\**mainMatrix*[i][2];  
 m61+=*mainMatrix*[i][6];  
 m71+=Math.*pow*(*mainMatrix*[i][0],2) \* *mainMatrix*[i][1] \* *mainMatrix*[i][2];  
 m81+=*mainMatrix*[i][0] \* *mainMatrix*[i][7];  
 m91+= *mainMatrix*[i][0] \* *mainMatrix*[i][8];  
 m101+= *mainMatrix*[i][0] \* *mainMatrix*[i][9];  
  
  
 m02+= *mainMatrix*[i][1];  
 m12+= *mainMatrix*[i][3];  
 m22+= Math.*pow*(*mainMatrix*[i][1],2);  
 m32+= *mainMatrix*[i][5];  
 m42+= *mainMatrix*[i][0]\*Math.*pow*(*mainMatrix*[i][1],2);  
 m52+= *mainMatrix*[i][6];  
 m62+=Math.*pow*(*mainMatrix*[i][1],2)\**mainMatrix*[i][2];  
 m72+=Math.*pow*(*mainMatrix*[i][1],2) \* *mainMatrix*[i][0] \* *mainMatrix*[i][2];  
 m82+= *mainMatrix*[i][1] \* *mainMatrix*[i][7];  
 m92+= *mainMatrix*[i][1] \* *mainMatrix*[i][8];  
 m102+= *mainMatrix*[i][1] \* *mainMatrix*[i][9];  
  
  
 m03+= *mainMatrix*[i][2];  
 m13+= *mainMatrix*[i][4];  
 m23+= *mainMatrix*[i][5];  
 m33+= Math.*pow*(*mainMatrix*[i][1],2);  
 m43+= *mainMatrix*[i][6];  
 m53+= *mainMatrix*[i][0]\*Math.*pow*(*mainMatrix*[i][2],2);  
 m63+= *mainMatrix*[i][1]\*Math.*pow*(*mainMatrix*[i][2],2);  
 m73+=Math.*pow*(*mainMatrix*[i][2],2) \* *mainMatrix*[i][0] \* *mainMatrix*[i][1];  
 m83+= *mainMatrix*[i][2] \* *mainMatrix*[i][7];  
 m93+= *mainMatrix*[i][2] \* *mainMatrix*[i][8];  
 m103+= *mainMatrix*[i][2] \* *mainMatrix*[i][9];  
  
 m04+= *mainMatrix*[i][3];  
 m14+= *mainMatrix*[i][1]\*Math.*pow*(*mainMatrix*[i][0],2);  
 m24+= *mainMatrix*[i][0]\*Math.*pow*(*mainMatrix*[i][1],2);  
 m34+= *mainMatrix*[i][6];  
 m44+= Math.*pow*(*mainMatrix*[i][0],2)\*Math.*pow*(*mainMatrix*[i][1],2);  
 m54+=Math.*pow*(*mainMatrix*[i][0],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][1];  
 m64+=Math.*pow*(*mainMatrix*[i][1],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][0];  
 m74+=Math.*pow*(*mainMatrix*[i][1],2) \* *mainMatrix*[i][2] \* Math.*pow*(*mainMatrix*[i][0],2);  
 m84+= *mainMatrix*[i][3] \* *mainMatrix*[i][7];  
 m94+= *mainMatrix*[i][3] \* *mainMatrix*[i][8];  
 m104+= *mainMatrix*[i][3] \* *mainMatrix*[i][9];  
  
  
 m05+= *mainMatrix*[i][4];  
 m15+= *mainMatrix*[i][2]\*Math.*pow*(*mainMatrix*[i][0],2);  
 m25+= *mainMatrix*[i][6];  
 m35+= *mainMatrix*[i][0]\*Math.*pow*(*mainMatrix*[i][2],2);  
 m45+=Math.*pow*(*mainMatrix*[i][0],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][1];  
 m55+= Math.*pow*(*mainMatrix*[i][0],2)\*Math.*pow*(*mainMatrix*[i][2],2);  
 m65+=Math.*pow*(*mainMatrix*[i][2],2) \* *mainMatrix*[i][0] \* *mainMatrix*[i][1];  
 m75+=Math.*pow*(*mainMatrix*[i][2],2) \* *mainMatrix*[i][1] \* Math.*pow*(*mainMatrix*[i][0],2);  
 m85+= *mainMatrix*[i][4] \* *mainMatrix*[i][7];  
 m95+= *mainMatrix*[i][4] \* *mainMatrix*[i][8];  
 m105+= *mainMatrix*[i][4] \* *mainMatrix*[i][9];  
  
 m06+= *mainMatrix*[i][5];  
 m16+= *mainMatrix*[i][6];  
 m26+= *mainMatrix*[i][2]\*Math.*pow*(*mainMatrix*[i][1],2);  
 m36+= *mainMatrix*[i][1]\*Math.*pow*(*mainMatrix*[i][2],2);  
 m46+=Math.*pow*(*mainMatrix*[i][1],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][0];  
 m56+=Math.*pow*(*mainMatrix*[i][2],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][1];  
 m66+= Math.*pow*(*mainMatrix*[i][1],2)\*Math.*pow*(*mainMatrix*[i][2],2);  
 m76+=Math.*pow*(*mainMatrix*[i][2],2) \* *mainMatrix*[i][0] \* Math.*pow*(*mainMatrix*[i][1],2);  
 m86+= *mainMatrix*[i][5] \* *mainMatrix*[i][7];  
 m96+= *mainMatrix*[i][5] \* *mainMatrix*[i][8];  
 m106+= *mainMatrix*[i][5] \* *mainMatrix*[i][9];  
  
  
 m07+= *mainMatrix*[i][6];  
 m17+=Math.*pow*(*mainMatrix*[i][0],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][1];  
 m27+=Math.*pow*(*mainMatrix*[i][1],2) \* *mainMatrix*[i][2] \* *mainMatrix*[i][0];  
 m37+=Math.*pow*(*mainMatrix*[i][2],2) \* *mainMatrix*[i][1] \* *mainMatrix*[i][0];  
 m47+= Math.*pow*(*mainMatrix*[i][1],2)\*Math.*pow*(*mainMatrix*[i][0],2)\**mainMatrix*[i][2];  
 m57+= Math.*pow*(*mainMatrix*[i][2],2)\*Math.*pow*(*mainMatrix*[i][0],2)\**mainMatrix*[i][1];  
 m67+= Math.*pow*(*mainMatrix*[i][2],2)\*Math.*pow*(*mainMatrix*[i][1],2)\**mainMatrix*[i][0];  
 m77+= Math.*pow*(*mainMatrix*[i][2],2)\*Math.*pow*(*mainMatrix*[i][1],2)\*Math.*pow*(*mainMatrix*[i][0],2);  
 m87+= *mainMatrix*[i][6] \* *mainMatrix*[i][7];  
 m97+= *mainMatrix*[i][6] \* *mainMatrix*[i][8];  
 m107+= *mainMatrix*[i][6] \* *mainMatrix*[i][9];  
  
  
  
 m08+=*mainMatrix*[i][7];  
 m18+=*mainMatrix*[i][7]\**mainMatrix*[i][0];  
 m28+=*mainMatrix*[i][7]\**mainMatrix*[i][1];  
 m38+=*mainMatrix*[i][7]\**mainMatrix*[i][2];  
 m48+=*mainMatrix*[i][7]\**mainMatrix*[i][3];  
 m58+=*mainMatrix*[i][7]\**mainMatrix*[i][4];  
 m68+=*mainMatrix*[i][7]\**mainMatrix*[i][5];  
 m78+=*mainMatrix*[i][7]\**mainMatrix*[i][6];  
 m88+=*mainMatrix*[i][7]\**mainMatrix*[i][7];  
 m98+=*mainMatrix*[i][7]\**mainMatrix*[i][8];  
 m108+=*mainMatrix*[i][7]\**mainMatrix*[i][9];  
  
 m09+=*mainMatrix*[i][8];  
 m19+=*mainMatrix*[i][8]\**mainMatrix*[i][0];  
 m29+=*mainMatrix*[i][8]\**mainMatrix*[i][1];  
 m39+=*mainMatrix*[i][8]\**mainMatrix*[i][2];  
 m49+=*mainMatrix*[i][8]\**mainMatrix*[i][3];  
 m59+=*mainMatrix*[i][8]\**mainMatrix*[i][4];  
 m69+=*mainMatrix*[i][8]\**mainMatrix*[i][5];  
 m79+=*mainMatrix*[i][8]\**mainMatrix*[i][6];  
 m89+=*mainMatrix*[i][8]\**mainMatrix*[i][7];  
 m99+=*mainMatrix*[i][8]\**mainMatrix*[i][8];  
 m109+=*mainMatrix*[i][8]\**mainMatrix*[i][9];  
  
 m010+=*mainMatrix*[i][9];  
 m110+=*mainMatrix*[i][9]\**mainMatrix*[i][0];  
 m210+=*mainMatrix*[i][9]\**mainMatrix*[i][1];  
 m310+=*mainMatrix*[i][9]\**mainMatrix*[i][2];  
 m410+=*mainMatrix*[i][9]\**mainMatrix*[i][3];  
 m510+=*mainMatrix*[i][9]\**mainMatrix*[i][4];  
 m610+=*mainMatrix*[i][9]\**mainMatrix*[i][5];  
 m710+=*mainMatrix*[i][9]\**mainMatrix*[i][6];  
 m810+=*mainMatrix*[i][9]\**mainMatrix*[i][7];  
 m910+=*mainMatrix*[i][9]\**mainMatrix*[i][8];  
 m1010+=*mainMatrix*[i][9]\**mainMatrix*[i][9];  
 }  
 }  
  
 return new Matrix(new double[][]{  
 {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},  
  
 });  
  
 }  
  
 private static Matrix generateZamenMatrix(){  
 double k0 = 0,k1 = 0,k2 = 0,k3 = 0,k4 = 0,k5 = 0,k6 = 0,k7 = 0,k8 = 0, k9 = 0, k10 = 0;  
 for(int i = 0; i < *mainMatrix*.length; i++){  
 for(int j = 0; j < *mainMatrix*[i].length; j++){  
 k0+= *mainMatrix*[i][*mainMatrix*[i].length-1];  
 k1+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][0];  
 k2+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][1];  
 k3+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][2];  
 k4+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][3];  
 k5+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][4];  
 k6+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][5];  
 k7+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][6];  
 k8+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][6];  
 k9+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][6];  
 k10+= *mainMatrix*[i][*mainMatrix*[i].length-1] \* *mainMatrix*[i][6];  
 }  
 }  
 return new Matrix(new double[][] {  
 {k0},  
 {k1},  
 {k2},  
 {k3},  
 {k4},  
 {k5},  
 {k6},  
 {k7},  
 {k8},  
 {k9},  
 {k10}  
 });  
 }  
  
  
 private static void regressionEquation(){  
 double mainDet = *generateStandartMatrix*().det();  
  
 Matrix rem = *generateStandartMatrix*();  
 rem.setMatrix(0,7,new int[]{0},*generateZamenMatrix* ());  
 double b0 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,7,new int[]{1},*generateZamenMatrix*());  
 double b1 = rem.det()/mainDet;  
  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{2},*generateZamenMatrix*());  
 double b2 = rem.det()/mainDet;  
  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{3},*generateZamenMatrix*());  
 double b3 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{4},*generateZamenMatrix*());  
 double b12 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{5},*generateZamenMatrix*());  
 double b13 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{6},*generateZamenMatrix*());  
 double b23 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{7},*generateZamenMatrix*());  
 double b123 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{8},*generateZamenMatrix*());  
 double b11 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{9},*generateZamenMatrix*());  
 double b22 = rem.det()/mainDet;  
  
 rem = *generateStandartMatrix*();  
 rem.setMatrix(0,0,new int[]{10},*generateZamenMatrix*());  
 double b33 = rem.det()/mainDet;  
  
 System.*out*.println("Regression equation");  
 System.*out*.printf("y = %.2f + %.2fx1 + %.2fx2 + %.2fx3 + %.2fx1x2 + %.2fx1x3 + %.2fx2x3 + %.2f x1x2x3 + %.2f x1^2 + %.2f x2^2 + %.2f x3^2\n",b0,b1,b2,b3,b12,b13,b23,b123,b11,b22,b33);  
 System.*out*.println("f(x1,x2,x3) = 5,0 + 4,7\*x1 + 3,6\*x2 + 6,4\*x3 + 6,8\*x1\*x1 + 0,3\*x2\*x2 + 5,3\*x3\*x3 + 3,2\*x1\*x2 + 0,9\*x1\*x3 + 2,7\*x2\*x3 + 0,1\*x1\*x2\*x3");  
 System.*out*.println("Зоряне плече l = k^(1/2) тому що всі зоряні точки знаходяться на однаковій відстані від початку експерименту\n l = 3^(1/2) = 1.73");  
 }  
  
 private static ArrayList<Double> dispersions(){  
 ArrayList<Double> dis = new ArrayList<>();  
 double temp = 0;  
 for(int i = 0; i < *mainMatrix*.length; i++){  
 for(int j = 10; j < *mainMatrix*[i].length-1; j++){  
 temp += Math.*pow*(*mainMatrix*[i][j] - *mainMatrix*[i][*mainMatrix*[i].length-1],2);  
 }  
 dis.add(temp/*m*);  
 temp = 0;  
 }  
 return dis;  
 }  
  
 private static void critKohren(){  
 System.*out*.println();  
 double r = 0;  
 Iterator<Double> disIter = *dispersions*().iterator();  
 for(;disIter.hasNext();){  
 r+= disIter.next();  
 }  
 double Gp = Collections.*max*(*dispersions*())/r;  
 System.*out*.println("\nKohren criterion:");  
 double f1 = *m*-1;  
 System.*out*.printf("f1 = %.2f\n" +  
 "Gp = %.2f\n",f1,Gp);  
 if(Gp < 0.5157) System.*out*.println("Dispersion is homogeneous!");  
 else {  
 System.*out*.println("Dispersion is not homogeneous!");  
 *m*+=1;  
 *generateMatrix*();  
 *critKohren*();  
 }  
 }  
  
 private static void critStudent(){  
 double Sbs = Math.*sqrt*(((*dispersions*().get(0) + *dispersions*().get(1) + *dispersions*().get(2) + *dispersions*().get(3)) / 4)/15\**m*);  
 double [] B = new double[11];  
 for(int i = 0; i < *mainMatrix*.length; i++){  
 B[0] += *mainMatrix*[i][*mainMatrix*.length -1];  
 B[1] += i < 4 ? -*mainMatrix*[i][*mainMatrix*.length-1] : *mainMatrix*[i][*mainMatrix*.length-1];  
 B[2] += i % 2 == 0 ? -*mainMatrix*[i][*mainMatrix*.length-1] : *mainMatrix*[i][*mainMatrix*.length-1];  
 B[3] += i >= 4 ? -*mainMatrix*[i][*mainMatrix*.length-1] : *mainMatrix*[i][*mainMatrix*.length-1];  
 B[4] -= *mainMatrix*[i][*mainMatrix*.length-1];  
 B[5] -= *mainMatrix*[i][*mainMatrix*.length-1];  
 B[6] -= *mainMatrix*[i][*mainMatrix*.length-1];  
 B[7] += *mainMatrix*[i][*mainMatrix*.length-1];  
 B[8] -= *mainMatrix*[i][*mainMatrix*.length-1];  
 B[9] += *mainMatrix*[i][*mainMatrix*.length-1];  
 B[10] -= *mainMatrix*[i][*mainMatrix*.length-1];  
  
 }  
 for(int i = 0; i < B.length; i++){  
 B[i]/=8;  
 }  
  
 double f3 = (*m*-1) \* 15;  
 System.*out*.println("\nStudent criterion");  
 System.*out*.printf("f3 = %.2f\n" +  
 "tk = 2.120\n",f3);  
  
 if(B[0]/Sbs < 2.12) System.*out*.println("b0 is unnecessary");  
 if(B[1]/Sbs > 2.12) System.*out*.print("");  
 if(B[2]/Sbs < 2.12) System.*out*.print("");  
 if(B[3]/Sbs > 2.12) System.*out*.print("");  
 if(B[4]/Sbs < 2.12) System.*out*.print("");  
 if(B[5]/Sbs < 2.12) System.*out*.println("b1b3 is unnecessary");  
 if(B[6]/Sbs < 2.12) System.*out*.print("");  
 if(B[7]/Sbs < 2.12) System.*out*.println("b1b2b3 is unnecessary");  
 if(B[8]/Sbs < 2.12) System.*out*.print("");  
 if(B[9]/Sbs < 2.12) System.*out*.println("b22 is unnecessary");  
 if(B[10]/Sbs < 2.12) System.*out*.println("b33 is unnecessary");  
 }  
  
  
 private static void critFisher() {  
 double s2 = ((*dispersions*().get(0) + *dispersions*().get(1) + *dispersions*().get(2) + *dispersions*().get(3) + *dispersions*().get(4) +  
 *dispersions*().get(5) + *dispersions*().get(6) + *dispersions*().get(7)) / 8) / (8 \* *m*);  
 double s2ad = 6.58;  
 double Fp = s2ad / s2;  
 System.*out*.println("\nFisher criterion");  
 System.*out*.printf("Fp = %.2f\n", Fp);  
 System.*out*.println("f4 = N - d = 8-3 = 5\n" +  
 "f3 = f1 \* f2 = (m-1) \* N = 2 \* 8 = 16\n" +  
 "Ft = 2.9");  
 if (Fp > 4.1) System.*out*.println("Regression equation is inadequate to the original!");  
 else System.*out*.println("Regression equation is adequate to the original!");  
 }  
  
 private static void crit(){  
 *critKohren*();  
 *critStudent*();  
 *critFisher*();  
 }  
  
 public static void runExperiment(){  
 *generateMatrix*();  
 *getMatrix*();  
 *regressionEquation*();  
 *crit*();  
 }  
  
  
  
  
}

**Результати виконання програми:**



**Висновок:** Під час виконання роботи проблем не виникало. Отримані результати збігаються з очікуваними. Необхідно рівняння з квадратичними членами, щоб модель була адекватна. Дисперсія завжди однорідна.