Міністерство освіти і науки України  
НТУУ «КПІ ім. І. Сікорського»

Кафедра цифрових технологій в енергетиці

Лабораторна робота № 4

«Комп’ютерне моделювання»

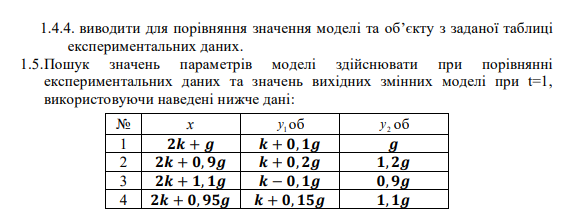
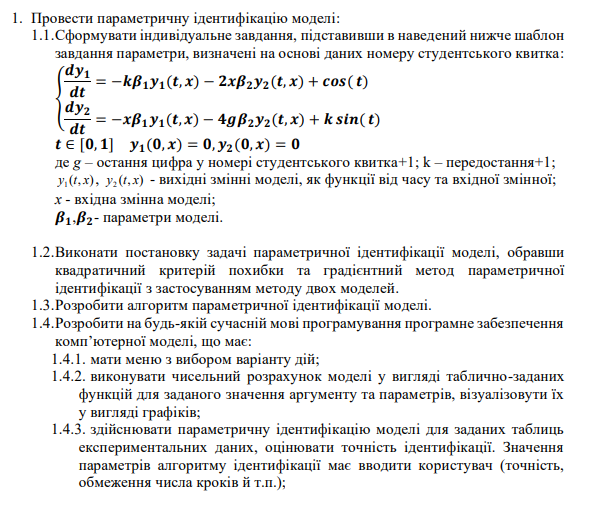
Варіант №25

Тема: «Параметрична ідентифікація моделей»

Виконав:  
студент 3-го курсу НН ІАТЕ  
групи ТР-12  
Руденко Владислав Ігорович

Київ – 2023

**Завдання**

****

*Коефіцієнти згідно до варіанту: g-6, k-4*

**Результати виконання**

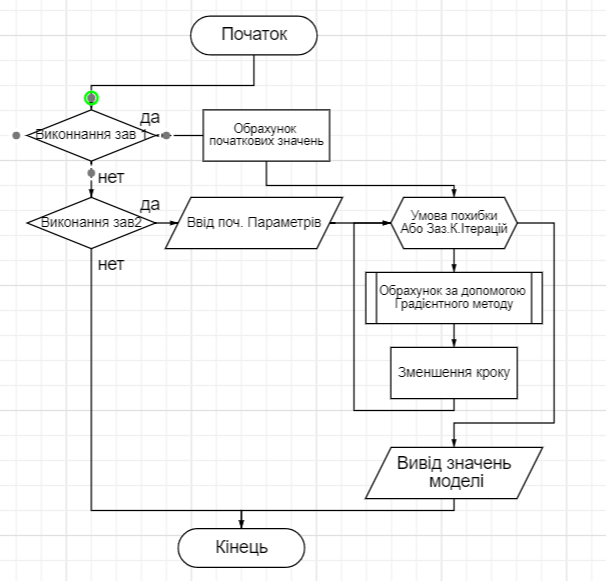
****

Рисунок Блок Схема Алгоритму Ідентифікації Моделі

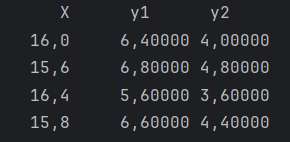
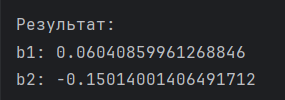
****

Рисунок Експерементальні Дані



**Робота програми:**

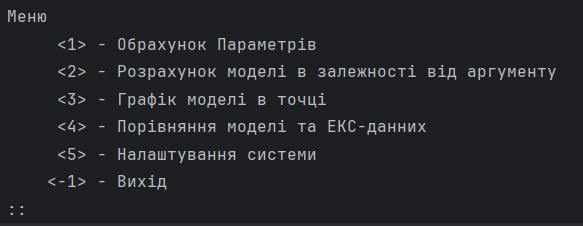
****

Рисунок Меню

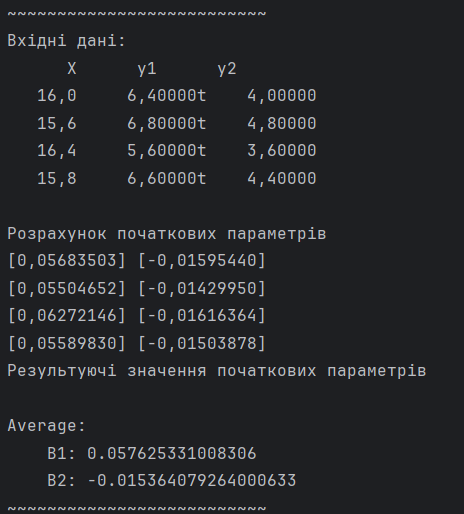
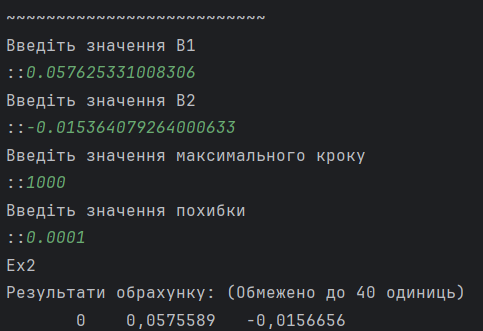
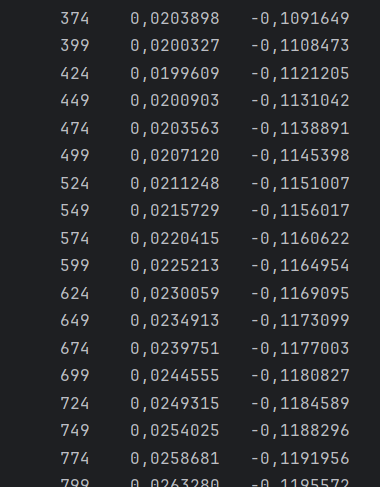
****

Рисунок Обрахунок початкових параметрів моделі

****

****

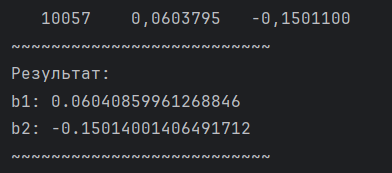
****

Рисунок Параметрична ідентифікація моделі

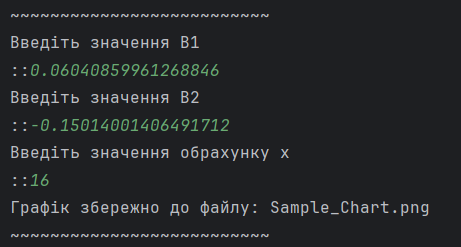
****

Рисунок Розрахунок графіку для заданих аргументів моделі

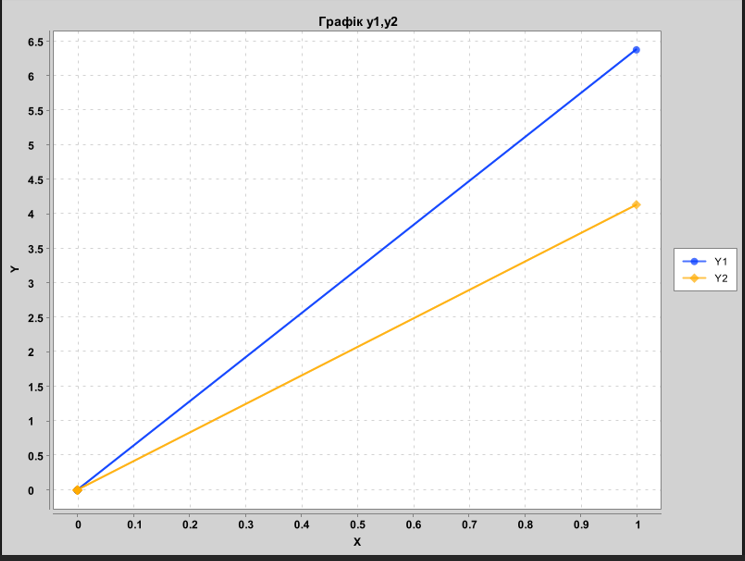


Рисунок Графік

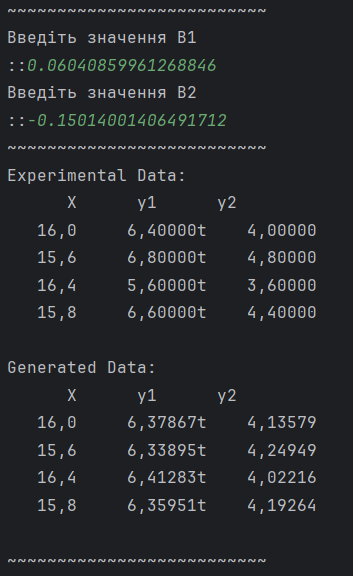
****

Рисунок Порівнянн моделі та експерементних даних

**Лістинг Коду**

import org.apache.commons.math3.linear.Array2DRowRealMatrix;  
import org.apache.commons.math3.linear.LUDecomposition;  
import org.apache.commons.math3.linear.RealMatrix;  
import org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
import org.apache.commons.math3.ode.FirstOrderIntegrator;  
import org.apache.commons.math3.ode.nonstiff.ClassicalRungeKuttaIntegrator;  
  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
import org.apache.commons.math3.exception.DimensionMismatchException;  
import org.apache.commons.math3.exception.MaxCountExceededException;  
import org.knowm.xchart.BitmapEncoder;  
import org.knowm.xchart.XYChart;  
import org.knowm.xchart.XYChartBuilder;  
  
import static java.lang.Math.\*;  
  
class Values  
{  
 public boolean Filled = false;  
 public boolean Lock = false;  
  
 //Settings  
 //Коеф-ти  
 public int  
 k =6,  
 g=4;  
 //Обрахунок  
 public double  
 maxIter = 100000,  
 error = 0.0001,  
 h=0.000001;  
 int Limiter = 40;  
  
 double[][] ExArray = new double[][]  
 {  
 {2\*k+g,k+0.1\*g,g},  
 {2\*k+0.9\*g,k+0.2\*g,1.2\*g},  
 {2\*k+1.1\*g,k-0.1\*g,0.9\*g},  
 {2\*k+0.95\*g,k+0.15\*g,1.1\*g}  
 };  
 double[][] WorkBArray = new double[4][2];  
 double[] AverageBArray = new double[2];  
 public double[][] getExArray(){return ExArray;}  
  
}  
  
class RK4  
{  
 private static final double *y1result* = 0;  
 private static final double *y2result* = 0;  
  
 public static double CalculationRootMeanSquareError(double[] current, double[] precision){  
 double sumSquaredDifferences = 0.0;  
 for (int i = 0; i < current.length; i++) {  
 double difference = current[i] - precision[i];  
 sumSquaredDifferences += Math.*pow*(difference, 2);  
 }  
 return Math.*sqrt*(sumSquaredDifferences / current.length);  
 }  
  
 private static double[] solveDifferentialEquation(FirstOrderIntegrator integrator, FirstOrderDifferentialEquations equations, double initialX, double initialY, double startTime, double endTime) {  
 double[] initialValues = new double[]{initialX, initialY};  
 integrator.integrate(equations, startTime, initialValues, endTime, initialValues);  
 return initialValues;  
 }  
  
 public static double[] RK45Lib\_LastY(double k, double g, double x, double b1, double b2){  
 List<double[]> result = new ArrayList<double[]>();  
 *RK45Lib\_Function*(k,g,x,b1,b2,result);  
 return result.get(result.size()-1);  
 }  
  
 public static List<double[]> RK45Lib(double k, double g, double x, double b1, double b2){  
 List<double[]> result = new ArrayList<double[]>();  
 *RK45Lib\_Function*(k,g,x,b1,b2,result);  
 return result;  
 }  
  
 public static List<double[]> RK45Lib\_Function(double k, double g, double x, double b1, double b2, List<double[]> result){  
 FirstOrderDifferentialEquations equations = new OverritedFirstOrderDifferentialEquations(k,g,x,b1,b2);  
 double stepShift =0.1;  
 double[] rungeKuttaRes = {*y1result*,*y2result*}, oldRungeKuttaRes;  
 result.add(rungeKuttaRes);  
 do {  
 oldRungeKuttaRes = rungeKuttaRes;  
 rungeKuttaRes = *solveDifferentialEquation*(new ClassicalRungeKuttaIntegrator(stepShift), equations,*y1result*,*y2result*,0,1);  
 stepShift/=2;  
 result.add(rungeKuttaRes);  
 }while (*CalculationRootMeanSquareError*(rungeKuttaRes, oldRungeKuttaRes) > 0.1);  
  
 return result;  
 }  
}  
  
class OverritedFirstOrderDifferentialEquations implements FirstOrderDifferentialEquations{  
 double k,g,x,b1,b2;  
  
 @Override  
 public int getDimension() {  
 return 2;  
 }  
  
 OverritedFirstOrderDifferentialEquations(double k,double g,double x,double b1,double b2)  
 {  
 this.b1=b1;  
 this.b2=b2;  
 this.k=k;  
 this.g=g;  
 this.x=x;  
 }  
  
 @Override  
 public void computeDerivatives(double t, double[] doubles, double[] doublesArray) throws MaxCountExceededException, DimensionMismatchException {  
  
 doublesArray[0] = -k \* b1 \* doubles[0] - 2 \* x \* b2 \* doubles[1] + *cos*(t);  
 doublesArray[1] = -x \* b1 \* doubles[0] - 4 \* g \* b2 \* doubles[1] + k \* *sin*(t);  
 }  
}  
  
  
public class Main {  
 public static void main(String[] args) {  
 *UI*();  
 }  
 public static void UI()  
 {  
 System.*out*.println("Початок Виконання:");  
 Values values = new Values();  
 boolean auto = false;  
 int auto\_counter = 0;  
 Scanner scan = new Scanner(System.*in*);  
 boolean end = true;  
 do{  
 *PrintMainUI*();  
 int CurrentTODOValue = !auto ? (scan.nextInt()) : auto\_counter++;  
 switch (CurrentTODOValue)  
 {  
 case 1:  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 *CreatePar*(values);  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 break;  
 case 2:  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 *CulcModel*(values);  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 break;  
 case 3:  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 *DisplayGraph*(values, auto);  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 break;  
 case 4:  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 *ComperisonTable*(values);  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 break;  
 case 5:  
 *PrintSettingsMainUI*(values, auto);  
 auto\_counter = 0;  
 break;  
 case -1:  
 end=false;  
 break;  
  
 default:  
  
 break;  
 }  
  
 }while (end);  
 }  
  
 public static void PrintMainUI()  
 {  
 System.*out*.println("Меню");  
 System.*out*.println("\t <1> - Обрахунок Параметрів");  
 System.*out*.println("\t <2> - Розрахунок моделі в залежності від аргументу");  
 System.*out*.println("\t <3> - Графік моделі в точці");  
 System.*out*.println("\t <4> - Порівняння моделі та ЕКС-данних");  
 System.*out*.println("\t <5> - Налаштування системи");  
 System.*out*.println("\t<-1> - Вихід");  
 System.*out*.print("::");  
 }  
  
 public static void PrintSettingsMainUI(Values values, boolean auto)  
 {  
 if(auto)  
 {  
 auto = false;  
 System.*out*.println("Авто-Мод вимкнено");  
 }  
  
 System.*out*.println("Меню Нашатувань");  
 System.*out*.println("\t <1> - Встановити базові параметри k,g");  
 System.*out*.println("\t <2> - Налаштування параметрів виводу");  
 System.*out*.println("\t <3> - Увімкнення авто-моду");  
 System.*out*.println("\t <4> - Налаштування параметрів авто-моду");  
 System.*out*.println("\t <5> - Активаці/Деактивація Заборони на використання існуючих даних");  
 System.*out*.println("\t<-1> - Вихід");  
 System.*out*.print("::");  
 Scanner scan = new Scanner(System.*in*);  
 int choose = scan.nextInt();  
 switch (choose)  
 {  
 case -1:  
 return;  
 default:  
 return;  
 case 1:  
 System.*out*.printf("Введіть значення k\n::");  
 values.k = scan.nextInt();  
 System.*out*.printf("Введіть значення g\n::");  
 values.g = scan.nextInt();  
 System.*out*.println("Успіх.");  
 break;  
 case 2:  
 System.*out*.printf("Введіть значення кількості значеннь при виводі\n::");  
 values.Limiter = 40;  
 System.*out*.println("Успіх.");  
 return;  
 case 3:  
 System.*out*.println("Успіх.");  
 auto = true;  
 values.Lock = false;  
 break;  
 case 4:  
 System.*out*.printf("Введіть значення Максимальної іттерації\n::");  
 values.maxIter = 100000;  
 System.*out*.printf("Введіть значення похибки\n::");  
 values.error = 0.0001;  
 System.*out*.printf("Введіть значення кроку\n::");  
 values.h=0.000001;  
 System.*out*.printf("Введіть значення кількості значеннь при виводі\n::");  
 values.Limiter = 40;  
 System.*out*.println("Успіх.");  
 break;  
 case 5:  
 auto = false;  
 values.Lock = !values.Lock;  
 System.*out*.println("Успіх.");  
 break;  
 }  
 }  
 private static void ComperisonTable(Values values) {  
 if(!values.Filled || values.Lock)  
 {  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.printf("Введіть значення B1\n::");  
 double B1 = Double.*parseDouble*(scan.nextLine());  
 System.*out*.printf("Введіть значення B2\n::");  
 double B2= Double.*parseDouble*(scan.nextLine());  
 values.AverageBArray= new double[]{B1,B2};  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 }  
 *printYTable*(values.getExArray(),"Experimental Data");  
 double[][] OwnYTable = new double[4][3];  
 for(int i= 0;i<4;i++)  
 {  
 OwnYTable[i][0] = values.getExArray()[i][0];  
 OwnYTable[i][1] = RK4.*RK45Lib\_LastY*(values.k,values.g,OwnYTable[i][0],values.AverageBArray[0],values.AverageBArray[1])[0];  
 OwnYTable[i][2] = RK4.*RK45Lib\_LastY*(values.k,values.g,OwnYTable[i][0],values.AverageBArray[0],values.AverageBArray[1])[1];  
 }  
 *printYTable*(OwnYTable,"Generated Data");  
 }  
  
  
 private static void printYTable(double[][] Input2DArray, String Massage)  
 {  
 System.*out*.println(Massage+":");  
 System.*out*.printf("%7.2s\t%7.5s\t%7.5s\n","X","y1","y2");  
 for (double[] tempValueArray:Input2DArray) {  
 System.*out*.printf("%7.1f \t%7.5ft\t%7.5f\n",tempValueArray[0],tempValueArray[1],tempValueArray[2]);  
 }  
 System.*out*.println();  
 }  
  
 private static void DisplayGraph(Values values, boolean auto)  
 {  
 if(!values.Filled || values.Lock)  
 {  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.printf("Введіть значення B1\n::");  
 double B1 = Double.*parseDouble*(scan.nextLine());  
 System.*out*.printf("Введіть значення B2\n::");  
 double B2= Double.*parseDouble*(scan.nextLine());  
 values.AverageBArray= new double[]{B1,B2};  
 }  
 Scanner scan = new Scanner(System.*in*);  
 int staticvalue = 16;  
 if(!auto)  
 {  
 System.*out*.printf("Введіть значення обрахунку x\n::");  
 }  
 int x = !auto?scan.nextInt():staticvalue;  
 List<double[]> result = RK4.*RK45Lib*(values.k,values.g,x,values.AverageBArray[0],values.AverageBArray[1]);  
  
 int RequestSize = 10;  
 double scalefactor = (double)result.size()/RequestSize;  
 double[] newResultPosition = new double[RequestSize];  
 double[] newResultY2 = new double[RequestSize];  
 double[] newResultY1 = new double[RequestSize];  
 if(RequestSize>result.size())  
 {  
 for(int i=0;i<result.size()-1;i++)  
 {  
 newResultPosition[i] = i;  
 newResultY1[i] = result.get(i)[0];  
 newResultY2[i] = result.get(i)[1];  
 }  
 }  
 else  
 {  
 for(int i=0;i<RequestSize;i++)  
 {  
 int newIndex = (int) Math.*floor*(i\*scalefactor);  
 if(newIndex >= result.size()){  
 newIndex = result.size()-1;  
 }  
 newResultPosition[i] = i;  
 newResultY1[i] = result.get(newIndex)[0];  
 newResultY2[i] = result.get(newIndex)[1];  
 }  
 }  
  
  
  
 XYChart chart = new XYChartBuilder().width(800).height(600).title("Графік y1,y2").xAxisTitle("X").yAxisTitle("Y").build();  
 chart.addSeries("Y1", newResultPosition, newResultY1);  
 chart.addSeries("Y2", newResultPosition, newResultY2);  
 try {  
 BitmapEncoder.*saveBitmap*(chart, "./Sample\_Chart", BitmapEncoder.BitmapFormat.*PNG*);  
 System.*out*.println("График сохранен в файл: Sample\_Chart.png");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
 private static void CreatePar(Values values)  
 {  
 *printYTable*(values.ExArray,"Вхідні дані");  
 double[][] TempExValue = values.getExArray();  
 double[] avgb1b2 = new double[]{0,0};  
  
 int i=0;  
 for (double[] tempArrayLine:TempExValue) {  
 double dy1dt = *dydt*(tempArrayLine[1],0,tempArrayLine[0],0);  
 double dy2dt = *dydt*(tempArrayLine[2],0,tempArrayLine[0],0);  
  
 dy1dt-=*cos*(1);  
 dy2dt-=values.k\**sin*(1);  
  
  
 double[] b1b2 = *gause*(new double[]{dy1dt,dy2dt},new double[][]{  
 {-values.k\*tempArrayLine[1],-2\*tempArrayLine[0]\*tempArrayLine[2]},  
 {-tempArrayLine[0]\*tempArrayLine[1],-4\*values.g\*tempArrayLine[2]}  
 });  
 values.WorkBArray[i][0] = b1b2[0];  
 values.WorkBArray[i][1] = b1b2[1];  
  
  
 avgb1b2[0] += b1b2[0];  
 avgb1b2[1] += b1b2[1];  
 i++;  
 }  
  
 values.AverageBArray[0] = avgb1b2[0]/4;  
 values.AverageBArray[1] = avgb1b2[1]/4;  
  
  
 {  
 System.*out*.println("Розрахунок початкових параметрів");  
 for (double[] tempb1b2Value:values.WorkBArray) {  
 System.*out*.printf("[%7.8f] [%7.8f]\n",tempb1b2Value[0],tempb1b2Value[1]);  
 }  
 System.*out*.println("Результуючі значення початкових параметрів");  
 System.*out*.println("\nAverage:\n\tB1: "+values.AverageBArray[0]+"\n\tB2: "+values.AverageBArray[1]);  
 }  
 values.Filled = true;  
 }  
  
 public static double dydt(double y1x1, double y2x2, double x1, double x2)  
 {  
 return (y1x1-y2x2)/(x1-x2);  
 }  
  
 public static double[] gause(double[] constants, double[][] coef)  
 {  
 RealMatrix matrix = new Array2DRowRealMatrix(coef);  
 LUDecomposition luDecomposition = new LUDecomposition(matrix);  
  
 RealMatrix constantsMatrix = new Array2DRowRealMatrix(new double[][] {constants});  
 RealMatrix solution = luDecomposition.getSolver().solve(constantsMatrix.transpose());  
  
 double[] double\_solution = new double[solution.getRowDimension()];  
 for (int i = 0; i < solution.getRowDimension(); i++) {  
 double\_solution[i] = solution.getEntry(i, 0);  
 }  
 return double\_solution;  
 }  
  
 private static void CulcModel(Values values) {  
 List<double[]> result = new ArrayList<>();  
 if(!values.Filled || values.Lock)  
 {  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.printf("Введіть значення B1\n::");  
 double B1 = Double.*parseDouble*(scan.nextLine());  
 System.*out*.printf("Введіть значення B2\n::");  
 double B2= Double.*parseDouble*(scan.nextLine());  
 System.*out*.printf("Введіть значення максимального кроку\n::");  
 values.maxIter = Integer.*parseInt*(scan.nextLine());  
 System.*out*.printf("Введіть значення похибки\n::");  
 values.error = Double.*parseDouble*(scan.nextLine());  
 values.AverageBArray= new double[]{B1,B2};  
 }  
  
 /// Req  
 int maxIter = (int)values.maxIter;  
 double error = values.error;  
 //  
  
 double maxError = Double.*MAX\_VALUE*, preError, h=values.h;  
 int Iteration = 1;  
  
 do {  
 preError = error;  
 error = *twoSquereError*(values.k,values.g,values.getExArray(),values.AverageBArray[0],values.AverageBArray[1]);  
 if(preError<error)  
 {  
 h/=2;  
 } else if (preError==error) {  
 break;  
 }  
 double[] grad = *solveGradient*(values.k,values.g,values.getExArray(),values.AverageBArray[0],values.AverageBArray[1]);  
 values.AverageBArray[0] -= h\*grad[0];  
 values.AverageBArray[1] -= h\*grad[1];  
 result.add(new double[] {values.AverageBArray[0],values.AverageBArray[1]});  
 Iteration++;  
  
 }while (Iteration < maxIter && maxError>error);  
 *printLimitModelResult*(values,result, "Ex2");  
 }  
  
 private static void printLimitModelResult(Values values, List<double[]> result, String message) {  
 System.*out*.println(message);  
 int Limiter = values.Limiter;  
 System.*out*.println("Результати обрахунку: "+(Limiter<0?"(Без обмежень)":"(Обмежено до "+Limiter+" одиниць)"));  
 for(int i=0;i<(Limiter<0?result.size()-1:40);i++)  
 {  
 if(Limiter>0)  
 {  
 int newIndex = (int) Math.*floor*(i\*(float)result.size()/Limiter);  
 if(newIndex >= result.size()){  
 newIndex = result.size()-1;  
 }  
 System.*out*.printf("%8.0f\t%8.7f\t%8.7f\n",(float)newIndex,result.get(newIndex)[0],result.get(newIndex)[1]);  
 }  
 else  
 {  
 System.*out*.printf("%8.0f\t%8.7f\t%8.7f\n",(float)i,result.get(i)[0],result.get(i)[1]);  
 }  
 }  
 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~");  
 System.*out*.println("Результат: ");  
 System.*out*.println("b1: "+result.get(result.size()-1)[0]+"\nb2: "+result.get(result.size()-1)[1]);  
  
 }  
  
 private static double[] solveGradient(int k, int g, double[][] ExArray, double b1, double b2) {  
 return new double[] {*solveGradientFirstPartFunction*(k,g,ExArray,b1,b2),*solveGradientSecondPartFunction*(k,g,ExArray,b1,b2)};  
 }  
  
 private static double solveGradientFirstPartFunction(int k, int g, double[][] ExArray, double b1, double b2) {  
 double miss = 0.000001;  
 double e =0;  
 int i =0;  
 for(double[] tempD : ExArray)  
 {  
 double[] modelres = RK4.*RK45Lib\_LastY*(k,g,tempD[0],b1,b2);  
 double[] twoModels = *getDoubles*(k,g,ExArray,b1,b2,i,miss,RK4.*RK45Lib\_LastY*(k,g,ExArray[i][0],b1+miss,b2));  
 e+=(modelres[0]-tempD[1])\*twoModels[0]+(modelres[1]-tempD[2])\*twoModels[1];  
 i++;  
 }  
 return e;  
 }  
  
 private static double solveGradientSecondPartFunction(int k, int g, double[][] ExArray, double b1, double b2) {  
 double e =0;  
 int i =0;  
 double miss = 0.000001;  
 for(double[] tempD : ExArray)  
 {  
 double[] modelres = RK4.*RK45Lib\_LastY* (k,g,tempD[0],b1,b2);  
 double[] twoModels = *getDoubles*(k,g,ExArray,b1,b2,i,miss,RK4.*RK45Lib\_LastY*(k,g,ExArray[i][0],b1,b2+miss));  
 e+=(modelres[0]-tempD[1])\*twoModels[0]+(modelres[1]-tempD[2])\*twoModels[1];  
 i++;  
 }  
 return e;  
 }  
  
 private static double[] getDoubles(int k, int g, double[][] ExArray, double b1, double b2, int i, double db, double[] modelres) {  
 double[] newModelRes = RK4.*RK45Lib\_LastY*(k,g,ExArray[i][0],b1,b2);  
 return new double[]  
 {  
 (modelres[0]-newModelRes[0])/db,  
 (modelres[1]-newModelRes[1])/db,  
 };  
 }  
  
 private static double twoSquereError(int k, int g, double[][] exArray, double b1, double b2) {  
 double[] xValues = new double[exArray.length];  
 double[] y1Observeo = new double[exArray.length];  
 double[] y2Observeo = new double[exArray.length];  
  
 for(int i=0;i<exArray.length;i++)  
 {  
 xValues[i] = exArray[i][0];  
 y1Observeo[i] = exArray[i][1];  
 y2Observeo[i]= exArray[i][2];  
 }  
  
 double[] y1Predict = new double[exArray.length];  
 double[] y2Predict = new double[exArray.length];  
 for(int i=0;i<exArray.length;i++)  
 {  
 double[] modelRes = RK4.*RK45Lib\_LastY*(k, g, xValues[i], b1, b2);  
 y1Predict[i] = modelRes[0];  
 y2Predict[i] = modelRes[1];  
 }  
 double error = RK4.*CalculationRootMeanSquareError*(y1Predict,y1Observeo)+RK4.*CalculationRootMeanSquareError*(y2Predict,y2Observeo);  
 return error/2;  
 }  
}