**MN**

**LISTA 10**

**WIOLETTA ŁUPKOWSKA**

**244831, CZW. 9:15**

**Zad.1**

**import** org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
**import** org.apache.commons.math3.ode.FirstOrderIntegrator;  
**import** org.apache.commons.math3.ode.nonstiff.EulerIntegrator;  
  
**import** java.util.ArrayList;  
  
**public class** TestLotkaVolterra {  
  
 **public static void** main(String[] args) {  
 **double**[] params= **new double**[]{1.5,1,1,3,1,2,0.2,0.4};  
 **double**[] xStart= **new double**[]{50,10};  
 **double**[] xStop= **new double**[]{0,0};  
  
  
 FirstOrderDifferentialEquations lotkaVolterraODE = **new** LotkaVolterraODE(params);  
 FirstOrderIntegrator eulerInt = **new** EulerIntegrator(0.01);  
 LotkaVolterraPath lotkaVolterraPath = **new** LotkaVolterraPath();  
 eulerInt.addStepHandler(lotkaVolterraPath);  
 eulerInt.integrate(lotkaVolterraODE,0,xStart,50,xStop);  
  
 FirstOrderDifferentialEquations lotkaVolterraODE2 = **new** LotkaVolterraODE2(params);  
 FirstOrderIntegrator eulerInt2 = **new** EulerIntegrator(0.01);  
 LotkaVolterraPath lotkaVolterraPath2 = **new** LotkaVolterraPath();  
 eulerInt2.addStepHandler(lotkaVolterraPath2);  
 eulerInt2.integrate(lotkaVolterraODE2,0,xStart,100,xStop);  
  
  
 FirstOrderDifferentialEquations lotkaVolterraODE3 = **new** LotkaVolterraODE3(params);  
 FirstOrderIntegrator eulerInt3 = **new** EulerIntegrator(0.01);  
 LotkaVolterraPath lotkaVolterraPath3 = **new** LotkaVolterraPath();  
 eulerInt3.addStepHandler(lotkaVolterraPath3);  
 eulerInt3.integrate(lotkaVolterraODE3,0,xStart,100,xStop);  
  
 System.***out***.println(**"t,preys,predators"**);  
 *//System.out.println(lotkaVolterraPath3.getTime());  
 //System.out.println(lotkaVolterraPath3.getPreys());* System.***out***.println(lotkaVolterraPath3.getPredators());  
  
  
  
 }  
  
  
}

**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.ode.sampling.StepHandler;  
**import** org.apache.commons.math3.ode.sampling.StepInterpolator;  
  
**import** java.util.ArrayList;  
  
**public class** LotkaVolterraPath **implements** StepHandler {  
  
 ArrayList<Double> **time** = **new** ArrayList<>();  
 ArrayList<Double> **preys** = **new** ArrayList<>();  
 ArrayList<Double> **predators** = **new** ArrayList<>();  
  
 **public** ArrayList<Double> getTime() {  
 **return time**;  
 }  
  
 **public** ArrayList<Double> getPreys() {  
 **return preys**;  
 }  
  
 **public** ArrayList<Double> getPredators() {  
 **return predators**;  
 }  
  
 @Override  
 **public void** init(**double** t0, **double**[] y0, **double** t) {  
  
 }  
  
 @Override  
 **public void** handleStep(StepInterpolator interpolator, **boolean** isLast) **throws** MaxCountExceededException {  
  
 **time**.add(interpolator.getCurrentTime());  
 **double**[] x = interpolator.getInterpolatedState();  
 **preys**.add(x[0]);  
 **predators**.add(x[1]);  
  
  
  
 }  
}

**import** org.apache.commons.math3.exception.DimensionMismatchException;  
**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
  
**public class** LotkaVolterraODE **implements** FirstOrderDifferentialEquations {  
  
 **private double**[] **params**;  
  
 **public** LotkaVolterraODE(**double**[] params){  
 **this**.**params** = params;  
 }  
  
 @Override  
 **public int** getDimension() {  
 **return** 2;  
 }  
  
 @Override  
 **public void** computeDerivatives(**double** t, **double**[] x, **double**[] dxdt) **throws** MaxCountExceededException, DimensionMismatchException {  
 dxdt[0]=(**params**[0]-**params**[1]\*x[1])\*x[0];  
 dxdt[1]=(**params**[2]\*x[0]-**params**[3])\*x[1];  
 }  
}

**import** org.apache.commons.math3.exception.DimensionMismatchException;  
**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
  
**public class** LotkaVolterraODE2 **implements** FirstOrderDifferentialEquations {  
  
 **private double**[] **params**;  
  
 **public** LotkaVolterraODE2(**double**[] params){  
 **this**.**params** = params;  
 }  
  
 @Override  
 **public int** getDimension() {  
 **return** 2;  
 }  
  
 *//params={a,b,c,d,e,f}  
 // 0,1,2,3,4,5* @Override  
 **public void** computeDerivatives(**double** t, **double**[] x, **double**[] dxdt) **throws** MaxCountExceededException, DimensionMismatchException {  
 dxdt[0]=(**params**[0]-**params**[1]\*x[1])\*x[0] - **params**[4]\*x[0];  
 dxdt[1]=(**params**[2]\*x[0]-**params**[3])\*x[1]- **params**[5]\*x[1];  
 }  
}

**import** org.apache.commons.math3.exception.DimensionMismatchException;  
**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
  
**public class** LotkaVolterraODE3 **implements** FirstOrderDifferentialEquations {  
  
 **private double**[] **params**;  
  
 **public** LotkaVolterraODE3(**double**[] params){  
 **this**.**params** = params;  
 }  
  
 @Override  
 **public int** getDimension() {  
 **return** 2;  
 }  
  
 *//params={a,b,c,d,e,f,g,h}  
 // 0,1,2,3,4,5,6,7* @Override  
 **public void** computeDerivatives(**double** t, **double**[] x, **double**[] dxdt) **throws** MaxCountExceededException, DimensionMismatchException {  
 dxdt[0]=(**params**[0]-**params**[1]\*x[1])\*x[0] - **params**[6]\*Math.*pow*(x[0],2);  
 dxdt[1]=(**params**[2]\*x[0]-**params**[3])\*x[1]- **params**[7]\*Math.*pow*(x[1],2);  
 }  
}

**W MATLABIE**

[t,x] = ode45(@odefunLV,[0 30],[50 10],[],[1.5 1 1 3 ]);

figure;

plot(t,x);

legend('ofiary','drapiezniki')

figure

plot(x(:,1),x(:,2));

%%

[t2,x2] = ode45(@odefunLV2,[0 60],[50 10],[],[1.5 1 1 3 1 2]);

figure;

plot(t2,x2);

legend('ofiary','drapiezniki')

figure

plot(x2(:,1),x2(:,2));

%%

[t3,x3] = ode45(@odefunLV3,[0 60],[50 10],[],[1.5 1 1 3 1 2 0.2 0.4]);

figure;

plot(t3,x3);

legend('ofiary','drapiezniki')

figure

plot(x3(:,1),x3(:,2));

function dxdt = odefunLV(t,x,params)

dxdt = [(params(1)-params(2)\*x(2))\*x(1); ((params(3))\*x(1)-params(4))\*x(2)];

end

function dxdt = odefunLV2(t,x,params)

dxdt = [(params(1)-params(2)\*x(2))\*x(1)-params(5)\*x(1); ((params(3))\*x(1)-params(4))\*x(2)-params(6)\*x(2)];

end

function dxdt = odefunLV3(t,x,params)

dxdt = [(params(1)-params(2)\*x(2))\*x(1)-params(7)\*x(1)\*x(1); ((params(3))\*x(1)-params(4))\*x(2)-params(8)\*x(2)\*x(2)];

end

**Zad.1 wyniki**

**1a**

****

Rysunek :Wykres populacji drapieżników i ofiar w funkcji czasu, **Matlab.**



Rysunek : Wykres populacji drapieżników i ofiar w funkcji czasu, **java.**



Rysunek : przestrzeń fazowa narysowana przy uzyciu wyników z pragramu **MatLab**

****

Rysunek :przestrzeń fazowa narysowana przy uzyciu wyników wygenerowanych na podstawie kodu j**ava**

**1b (e=1, f=2)**

****

Rysunek :Wykres populacji drapieżników i ofiar w funkcji czasu, **matlab.**



Rysunek : Wykres populacji drapieżników i ofiar w funkcji czasu**, java.**

****

Rysunek : Przestrzeń fazowa, **matlab.**

****

Rysunek :Przestrzen fazowa, **java.**

**Wykresy otrzymane z javy wydają mi się błędne ze względu na wzrastającą wysokość pików w czasie obu populacji.**

**1c**

**Wyniki Matlab:**

****

Rysunek : Przestrzeń fazowa, matlab.

****

Rysunek : Wykres ofiar, drapieżników w czasie.

**Wyniki java:**

****

Rysunek 11: Przestrzeń fazowa, matlab.

****

Rysunek 1211: Wykres ofiar, drapieżników w czasie.

**Minimum ofiar w przybliżeniu przypada na maksimum drapieżników.**

**Zad. 2**

**package** Zad2;  
  
**import** org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
**import** org.apache.commons.math3.ode.FirstOrderIntegrator;  
**import** org.apache.commons.math3.ode.nonstiff.EulerIntegrator;  
  
**import** java.util.ArrayList;  
  
**public class** TestLotkaVolterra {  
  
 **public static void** main(String[] args) {  
 **double**[] params= **new double**[]{3,13,1000};  
 **double**[] xStart= **new double**[]{90,90,10};  
 **double**[] xStop= **new double**[]{0,0,0};  
  
 **double** max;  
   
 FirstOrderDifferentialEquations lotkaVolterraODE = **new** LotkaVolterraODE(params);  
 FirstOrderIntegrator eulerInt = **new** EulerIntegrator(0.01);  
 LotkaVolterraPath lotkaVolterraPath = **new** LotkaVolterraPath();  
 eulerInt.addStepHandler(lotkaVolterraPath);  
 eulerInt.integrate(lotkaVolterraODE,0,xStart,50,xStop);  
   
 ArrayList<Double> narazeni = lotkaVolterraPath.getNarazeni();  
 ArrayList<Double> chorzy = lotkaVolterraPath.getChorzy();  
 ArrayList<Double> odporni = lotkaVolterraPath.getOdporni();  
 ArrayList<Double> time = lotkaVolterraPath.getTime();  
 *//System.out.println(chorzy)* max=*max*(chorzy);  
 System.***out***.println(**"beta: "** + params[0]);  
 System.***out***.println(**"gamma: "** + params[1]);  
 System.***out***.println(**"dzień, w którym było najwięcej chorych: "**+max);  
 System.***out***.println(chorzy);  
   
 }  
  
 **public static double** max(ArrayList<Double> array){  
 **double** max=0;  
  
 **for**(**int** i= 0; i<array.size()-3; i++){  
 **if**(array.get(i)-array.get(i+1)<0 && array.get(i+2)-array.get(i+3)<0 ){  
 max= i+2;  
 }  
 }  
 **return** max;  
 };  
  
}

**package** Zad2;  
  
**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.ode.sampling.StepHandler;  
**import** org.apache.commons.math3.ode.sampling.StepInterpolator;  
  
**import** java.util.ArrayList;  
  
**public class** LotkaVolterraPath **implements** StepHandler {  
  
 ArrayList<Double> **narazeni** = **new** ArrayList<>();  
 ArrayList<Double> **chorzy** = **new** ArrayList<>();  
 ArrayList<Double> **odporni** = **new** ArrayList<>();  
 ArrayList<Integer> **czas**= **new** ArrayList<>();  
 ArrayList<Double> **time**= **new** ArrayList<>();  
  
 **public** ArrayList<Integer> getCzas() {  
 **return czas**;  
 }  
  
 **public** ArrayList<Double> getNarazeni() {  
 **return narazeni**;  
 }  
  
 **public** ArrayList<Double> getChorzy() {  
 **return chorzy**;  
 }  
  
 **public** ArrayList<Double> getOdporni() {**return odporni**; }  
  
 **public** ArrayList<Double> getTime() {**return time**; }  
  
 @Override  
 **public void** init(**double** t0, **double**[] y0, **double** t) {  
  
 }  
  
 @Override  
 **public void** handleStep(StepInterpolator interpolator, **boolean** isLast) **throws** MaxCountExceededException {  
  
 **time**.add(interpolator.getCurrentTime()\*100);  
 **double**[] x = interpolator.getInterpolatedState();  
 **narazeni**.add(x[0]);  
 **chorzy**.add(x[1]);  
 **odporni**.add(x[2]);  
  
  
 }  
}

**package** Zad2;  
  
**import** org.apache.commons.math3.exception.DimensionMismatchException;  
**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.ode.FirstOrderDifferentialEquations;  
  
**public class** LotkaVolterraODE **implements** FirstOrderDifferentialEquations {  
  
 **private double**[] **params**;  
  
 **public** LotkaVolterraODE(**double**[] params){  
 **this**.**params** = params;  
 }  
  
 @Override  
 **public int** getDimension() {  
 **return** 3;  
 }

*// params={beta, gamma, N}  
 // 0 1 2  
 // x[0] --- x  
 // x[1] --- y  
 // x[2] --- z* @Override  
 **public void** computeDerivatives(**double** t, **double**[] x, **double**[] dxdt) **throws** MaxCountExceededException, DimensionMismatchException {  
 dxdt[0]=-(**params**[0]/**params**[2])\*x[0]\*x[1]; *//zdrowi* dxdt[1]=(**params**[0]/**params**[2])\*x[0]\*x[1]- (1/**params**[1])\*x[1]; *//chorzy* dxdt[2]=(1/**params**[1])\*x[1]; *//odporni* }  
}

**Zad.2 wyniki**

**Parametry={1,10,1000} : (os/dzień, dni na wyleczenie, populacja)**

****

**Parametry={3,13,1000} :**

****

Liczba chorych w 1 przypadku nie ma piku (myślę, że zarażanie 1 os. na dzień powinno spowodować choćby nieznaczny wzrost liczby chorych ze względu na czas zdrowienia wynoszący 10 dni), ilość zdrowych, narażonych na chorobę maleje wolniej niż w przypadku poniżej, krzywa przedstawiająca osoby odporne jest bardzo zbliżona w obu przypadkach. Zmiana pierwszego parametru na więcej niż 1 powoduje powstanie piku na krzywej chorych.



