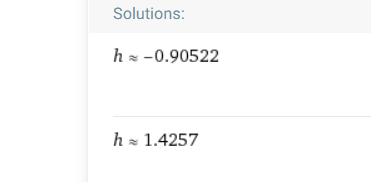
MN KART 1

ŁUPKOWSKA WIOLETTA, 244831

//poprawione równanie w zadaniu pierwszym//



Za wartość dokładną przyjmuję drugie h, ponieważ nie jest możliwe, aby wysokość h była <0.

1.1

**public interface** Funkcja {  
  
 **double** funkcjaZ(**double** x);  
}

**public class** LiczFunkcje **implements** Funkcja {  
  
**double roK**=200;  
**double roW**=1000;  
  
 @Override  
 **public double** funkcjaZ(**double** h) {  
 **return** 4./3.-Math.*pow*(h,2)\*((3-h)/3)-4./3.\*(200./1000.);  
 }  
}

**public class** Main {  
  
 **public static double** *xrDokladne* = 1.4257;  
  
 **public static void** main(String[] args) {  
  
 RegulaFalsi regulaFalsi = **new** RegulaFalsi(**new** LiczFunkcje());  
 RegulaPunktuStalego regulaPunktuStalego = **new** RegulaPunktuStalego(**new** LiczFunkcje());  
 MetodaStycznych metodaStycznych = **new** MetodaStycznych((**new** LiczFunkcje()));  
 MetodaSiecznych metodaSiecznych = **new** MetodaSiecznych((**new** LiczFunkcje()));  
  
  
 **double** solutionRegulaPunktu = regulaPunktuStalego.solver(0, *xrDokladne*, 0.001);  
 System.***out***.println(**"Regula Punktu :"**);  
 System.***out***.println(solutionRegulaPunktu );  
  
 **double** solutionRegulaFalsi = regulaFalsi.solver(0, 1, *xrDokladne*, 0.001);  
 System.***out***.println(**"Miejsce zerowe regula falsi:"**);  
 System.***out***.println(solutionRegulaFalsi );  
  
  
 **double** solutionMetodaStycznych = metodaStycznych.solver(0, *xrDokladne*, 0.001);  
 System.***out***.println(solutionMetodaStycznych + **" solution styczne"**);  
 System.***out***.println();  
  
 **double** solutionMetodaSiecznych = metodaSiecznych.solver(0, *xrDokladne*, 0.001);  
 System.***out***.println(solutionMetodaSiecznych + **" solution sieczne"**);  
 System.***out***.println();  
  
  
  
 }  
  
}

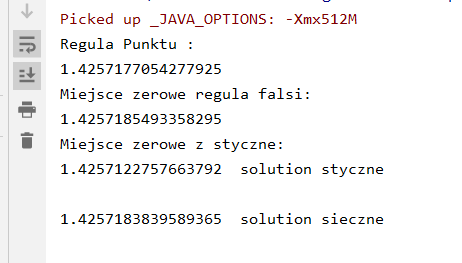
**import** java.util.ArrayList;  
  
**public class** MetodaSiecznych {  
  
 **private** Funkcja **f**;  
 **public** MetodaSiecznych(Funkcja f){  
 **this**.**f** = f;  
 }  
  
 **public double** liczEa (**double** xrNew, **double** xrOld){  
 **return** Math.*abs*((xrNew-xrOld)/xrNew)\*100;  
 }  
 ArrayList<Double> **listEa** = **new** ArrayList<Double>();  
 ArrayList<Double> **listXr** = **new** ArrayList<Double>();  
  
 **public double** solver(**double** x0, **double** xrDokladne, **double** zadanyBlad){  
  
 LiczFunkcje licz = **new** LiczFunkcje();  
 **double** xrOld=xrDokladne;  
 **double** xrOlder;  
 **double** xr=x0;  
 **double** xrNew=0;  
 **double** ea=1;  
 **double** xminusjeden=0;  
  
 **while**(ea>zadanyBlad){  
 xrOlder=xrOld;  
 xrOld=xr;  
 xr= xr-((licz.funkcjaZ(xr)\*(xrOlder-xr))/(licz.funkcjaZ(xrOlder)-licz.funkcjaZ(xr))); *//przyblizenie rozwiazania* xrNew=xr;  
 ea= liczEa(xrNew,xrOld);  
 **listEa**.add(ea);  
 **listXr**.add(xr);  
 }  
  
 **return listXr**.get(**listXr**.size()-1);  
  
 }  
  
  
  
}

**import** java.util.ArrayList;  
  
**public class** MetodaStycznych {  
  
 **private** Funkcja **f**;  
 **public** MetodaStycznych(Funkcja f){  
 **this**.**f** = f;  
 }  
  
 **public double** liczEa (**double** xrNew, **double** xrOld){  
 **return** Math.*abs*((xrNew-xrOld)/xrNew)\*100;  
 }  
  
 ArrayList<Double> **listEa** = **new** ArrayList<Double>();  
 ArrayList<Double> **listXr** = **new** ArrayList<Double>();  
  
 **public double** solver(**double** x0, **double** xrDokladne, **double** zadanyBlad){  
  
 LiczFunkcje licz = **new** LiczFunkcje();  
 **double** xrOld=xrDokladne;  
 **double** xr=x0;  
 **double** xrNew=0;  
 **double** ea=1;  
  
 **while**(ea>zadanyBlad){  
 xrOld=xr;  
 xr= xr-(licz.funkcjaZ(xr)/(-Math.*exp*(-xr)-1)); *//przyblizenie rozwiazania* xrNew=xr;  
 **double** et = Math.*abs*(xr-1.4257)/1.4257;  
 ea= liczEa(xrNew,xrOld);  
 **listEa**.add(ea);  
 **listXr**.add(xr);  
 }  
 System.***out***.println(**"Miejsce zerowe z styczne:"**);  
 **return listXr**.get(**listXr**.size()-1);  
  
 }  
  
  
}

**import** java.util.ArrayList;  
  
**public class** RegulaFalsi {  
  
 **private** Funkcja **f**;  
  
 **public** RegulaFalsi(Funkcja f) { *//tym ustawiam funkcję jakiej miejsc zerowych chce szukac* **this**.**f** = f;  
 }  
  
 **public double** liczEa (**double** xrNew, **double** xrOld){  
 **return** Math.*abs*((xrNew-xrOld)/xrNew)\*100;  
 }  
  
 ArrayList<Double> **listEa** = **new** ArrayList<Double>();  
 ArrayList<Double> **listXr** = **new** ArrayList<Double>();  
  
 **public double** solver(**double** xl, **double** xu, **double** xrDokladne, **double** zadanyBlad){  
  
 LiczFunkcje licz = **new** LiczFunkcje();  
 **double** xrOld=xrDokladne;  
 **double** xr=0;  
 **double** xrNew=0;  
 **double** ea=1;  
  
 **while**(ea>zadanyBlad){  
 xrOld=xr;  
 xr= xu-(licz.funkcjaZ(xu)\*(xl-xu))/(licz.funkcjaZ(xl)-licz.funkcjaZ(xu));  
 xrNew=xr;  
 **double** et = Math.*abs*(xr-1.4257)/1.4257;  
 ea= liczEa(xrNew,xrOld);  
 **listEa**.add(ea);  
 **listXr**.add(xr);  
  
  
 **if** (licz.funkcjaZ(xl) \* licz.funkcjaZ(xu)<0) {  
 xu = xr;  
 }**else if** (licz.funkcjaZ(xl)\*licz.funkcjaZ(xu)>0) {  
 xl = xr;  
 }**else if** (licz.funkcjaZ(xl)\*licz.funkcjaZ(xr)==0){  
 System.***out***.println(**"MAM PIERWIASTEK"**);  
  
 }  
 }  
 **return listXr**.get(**listXr**.size()-1);  
 }  
}

**import** java.util.ArrayList;  
  
**public class** RegulaPunktuStalego {  
  
 **private** Funkcja **f**;  
  
 **public** RegulaPunktuStalego(Funkcja f) { *//tym ustawiam funkcję jakiej miejsc zerowych chce szukac* **this**.**f** = f;  
 }  
  
 **public double** liczEa (**double** xrNew, **double** xrOld){  
 **return** Math.*abs*((xrNew-xrOld)/xrNew)\*100;  
 }  
  
 ArrayList<Double> **listEa** = **new** ArrayList<Double>();  
 ArrayList<Double> **listXr** = **new** ArrayList<Double>();  
  
 **public double** solver(**double** x0, **double** xrDokladne, **double** zadanyBlad){  
  
 LiczFunkcje licz = **new** LiczFunkcje();  
 **double** xrOld=xrDokladne;  
 **double** xr=x0;  
 **double** xrNew=0;  
 **double** ea=1;  
  
  
 **while**(ea>zadanyBlad){  
 xrOld=xr;  
 xr= licz.funkcjaZ(xr)+xr; *//przyblizenie rozwiazania* xrNew=xr;  
 ea= liczEa(xrNew,xrOld);  
 **listXr**.add(xr);  
  
  
 }  
  
 **return listXr**.get(**listXr**.size()-1);  
 }  
  
}

1.2



2.1

**public interface** FirstOrderODE {  
  
 **double** f(**double** t, **double** x);  
  
}

**public class** FirstOrderODESolver { *//rozwiazywanie rownania rozniczkowego 1 st* **private** ODESingleStep **odeSingleStep**;  
 **private** StepHandler **stepHandler**;  
  
 **public** FirstOrderODESolver(ODESingleStep odeSingleStep) { *//konstruktor pobierajacy krok, ustawia krok* **this**.**odeSingleStep** = odeSingleStep;  
 }  
  
 **public double** integrate(FirstOrderODE ode, **double** tStart, **double** xStart, **double** tStop, **int** n){ *//First order ode daje f(t,x), n=liczba krokow* **double** h = (tStop - tStart)/n; *//dlugosc kroku* **double** x= xStart; *//xPocz x0* **double** t = tStart; *//tPocz t0* **for**(**int** i = 0; i<n; i++){ *//i<liczba krokow* **if**(**stepHandler** != **null**)  
 **stepHandler**.handler(t,x);  
 x = **odeSingleStep**.singleStep(ode, t, x, h); *//ODESingleStep liczony inaczej dla kazdej z metod (euler, mofyfEuler, rk)* t += h;  
  
 }  
 **if**(**stepHandler** != **null**)  
 **stepHandler**.handler(t,x);  
 **return** x;  
 }  
  
  
}

**import** java.util.ArrayList;  
  
**public class** Main {  
  
  
 **public static int** *n*=500;  
 **public static double** *k*=0.026;  
 **public static double** *xMax*=12\*Math.*pow*(10,9);  
 **public static double** *x0*=2.555\*Math.*pow*(10,9);  
  
  
  
 **public static void** main(String[] args) {  
  
  
 FirstOrderODESolver solverRK3 = **new** FirstOrderODESolver((**new** RK3SingleStep()));  
  
 **double** xEndRK3 = solverRK3.integrate( (t,x) -> *k*\*(1-x/*xMax*)\*x , 0,*x0*,50,*n*);  
  
 System.***out***.println(**"solution Rungeg-Kutt3:"**);  
 System.***out***.println(xEndRK3);  
  
  
 }  
  
}

**public interface** ODESingleStep {  
  
 **double** singleStep(FirstOrderODE ode, **double** t, **double** x, **double** h);  
}

**public class** RK3SingleStep **implements** ODESingleStep {  
 **double k1**;  
 **double k2**;  
 **double k3**;  
 **double xEnd**;  
  
 @Override  
 **public double** singleStep(FirstOrderODE ode, **double** t, **double** x, **double** h) {  
  
 **k1** = ode.f(t,x);  
 **k2** = ode.f(t+h/2, x+**k1**\*(h/2));  
 **k3** = ode.f(t+h, x-h\***k1**+2\*h\***k2**);  
 **double** nach = (1./6.)\* (**k1**+4\***k2**+**k3**);  
 **xEnd** = x + h\*nach;  
 **return xEnd**;  
 }  
}

**public interface** StepHandler {  
 **void** handler(**double** t, **double** x);  
}

2.2

