Metody Obliczeniowe Zadanie 1.2

Uniwersytet Gdański, Informatyka III rok

Hallman , Górski, Rzeppa

Równanie:



Zadanie:

Napisać program realizujący metodę siecznych dla powyższego zadania. Uruchamiać program dla różnych punktów startowych metody. - Przeanalizować metodę stycznych i metodę iteracji prostych dla tego równania i jeśli okaże się to możliwe, zaimplementować te metody. Program uruchamiać z różnymi punktami startowymi i po wykonaniu metody wyświetlać stosowne komunikaty dotyczące wyniku, tj, czy metoda jest zbieżna, czy rozbieżna dla danego warunku początkowego i z jakiego powodu.

**Metoda siecznych**

Jeśli zmodyfikujemy metodę Newtona przez zamianę na iloraz to otrzymamy tzw. metodę siecznych:



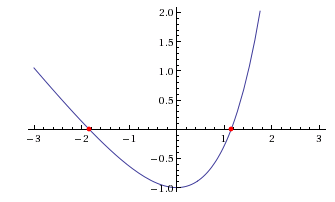
Metoda ta wymaga ustalenia w przedziale [a,b] dwóch punktów startowych 

**Interpretacja metody siecznych**

Na każdym kroku to miejsce zerowe siecznej wykresu w punktach oraz , tzn prostej o równaniu:



**Wykres funkcji**



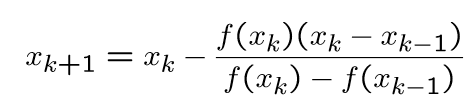
**Metoda siecznych:**

**Punkty startowe:**

x0 = -2

x1 = 2

**Wzór:**



**Obliczenia:**

x2 = 2 - ((3.389)(2-(-2)))\((3.389)-(0.135))= -2.166

f(-2.66) = 0.280 =/= 0

**Metoda iteracji prostych:**

**Wzór:**

ITERACJE.png

**Metoda stycznych:**

**Wzór:**

**styczne.png**

**Wejście**

- Początek przedziału - program wysyła komunikat do użytkownika: “Podaj początek przedziału”, porównuje typ wprowadzonych danych do double

- Koniec pedziału - program wysyła komunikat do użytkownika: “Podaj koniec przedziału”, porównuje typ wprowadzonych danych do double, sprawdza czy koniec przedziału jest większy niż początek.

**Wyjście**

- Kolejne wyniki: metodą iteracji prostych, siecznych, stycznych z dopuszczalnym błędem 0,0001.

**Klasy**

- Function.java – klasa wyliczająca i tworząca funkcje.

\* **public** **static** **double** **round**(**double** value, **int** places) – metoda zaokrąglająca dane używane w programie.

\***public** **double** **getDerivativeValue**(**double** x) – wyliczanie pochodnej funkcji.

\***double** **value**(**double** x) – wyliczanie wartości funkcji.

\***double** **getTangentValue**(**double** x) – przekształcanie funkcji dla metod iteracji prostych.

- Tangent.java – klasa licząca metodą stycznych.

\***public** **void** **solveTangent**() – metoda licząca metodą stycznych.

- Main.java – klasa główna uruchamiająca program.

\***public** **static** **void** **main**(**String**[] args) – metoda statyczna rozruchowa programu.

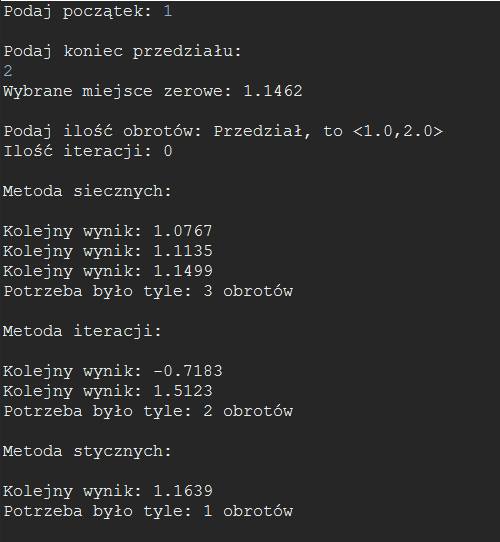
- Secant.java – klasa licząca metodą siecznych.

\***public** **void** **solveSecant**() – metoda licząca metodą siecznych.

- SimpleIteretion.java – klasa licząca metodą iteracji prostych.

\***public** **void** **solveIteration**() - klasa licząca metodą iteracji prostych.

**Zrzut ekranu dokumentujący działanie programu**



**Kod programu**

Main.java

**package** mhallman.methods.secant;

**import** java.util.Scanner;

**public** **class** **main** {

**public** **static** **void** **main**(**String**[] args) {

**Secant** **secant** = **new** Secant();

**SimpleIterations** **simpleIter** = **new** SimpleIterations();

**Tangent** **tangent** = **new** Tangent();

**double** **startingPoint**=0;

**double** **endPoint**=0;

**int** **numberOfLoop**=0;

**double** **zeroValue**= -1.8414;

**double** **zeroValue1**= 1.1462;

**double** **mainZeroValue** = 0;

**Scanner** **in** = **new** Scanner(**System**.***in***);

**System**.***out***.print("Podaj początek: ");

**if**(in.hasNextDouble()){

startingPoint=in.nextDouble();

secant.setPs(startingPoint);

simpleIter.setStartingPoint(startingPoint);

tangent.setStartingPoint(startingPoint);

}**else**{

**System**.***err***.println("Podano nieprawidłową wartość!!");

**System**.*exit*(0);

}

**System**.***out***.println();

**System**.***out***.print("Podaj koniec przedziału: ");

**if**(in.hasNextDouble()){

endPoint=in.nextDouble();

secant.setPk(endPoint);

**if**(secant.getPs()==secant.getPk()/\* || (secant.getPs()\*secant.getPk() > 0)\*/){

**System**.***err***.println("Podano nieprawidłową wartość!!");

**System**.*exit*(0);

}

}

**if**(zeroValue>=startingPoint && zeroValue<=endPoint){

mainZeroValue = zeroValue;

secant.setMainZeroValue(mainZeroValue);

}

**else** **if**(zeroValue1>=startingPoint && zeroValue1<=endPoint && mainZeroValue!=0){

**System**.***out***.println("Błąd");

**System**.*exit*(0);

}**else** **if**(zeroValue1>=startingPoint && zeroValue1<=endPoint){

mainZeroValue = zeroValue1;

secant.setMainZeroValue(mainZeroValue);

simpleIter.setMainZeroValue(mainZeroValue);

tangent.setMainZeroValue(mainZeroValue);

}

**System**.***out***.println("Wybrane miejsce zerowe: "+mainZeroValue);

**System**.***out***.println();

**System**.***out***.print("Podaj ilość obrotów: ");

**System**.***out***.println("Przedział‚ to <"+secant.getPs()+","+secant.getPk()+">");

**System**.***out***.println("Ilość iteracji: "+numberOfLoop);

**System**.***out***.println();

**System**.***out***.println("Metoda siecznych:");

**System**.***out***.println();

secant.solveSecant();

**System**.***out***.println();

**System**.***out***.println("Metoda iteracji:");

**System**.***out***.println();

simpleIter.solveIteration();

**System**.***out***.println();

**System**.***out***.println("Metoda stycznych:");

**System**.***out***.println();

tangent.solveTangent();

}

}

Function.java

**package** mhallman.methods.secant;

**import** java.lang.Math;

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**import** org.apache.commons.math3.analysis.differentiation.DerivativeStructure;

**public** **class** **Function** {

**private** **static** **double** *E* = java.lang.**Math**.***E***;

**private** **double** x;

**private** **DerivativeStructure** f;

**private** **DerivativeStructure** fun;

**private** **DerivativeStructure** fun1;

**private** **DerivativeStructure** fun2;

**private** **DerivativeStructure** free;

**private** **DerivativeStructure** ex;

**double** minusX;

**public** **Function**(){

}

**public** **static** **double** **round**(**double** value, **int** places) {

**if** (places < 0) **throw** **new** IllegalArgumentException();

**BigDecimal** **bd** = **new** BigDecimal(value);

bd = bd.setScale(places, *RoundingMode*.***HALF\_UP***);

**return** bd.doubleValue();

}

**public** **double** **getDerivativeValue**(**double** x){

f = **new** DerivativeStructure(1,0, 0, *E*);

**this**.x=x;

**this**.ex = **this**.f.pow(*round*(x,3));

**this**.free = **this**.f.pow(0);

//function e^x - 2 - x

**this**.fun1 = **new** DerivativeStructure(1,ex,-1,free);

**return** *round*(fun1.getValue(),4);

}

**double** **value**(**double** x){

f = **new** DerivativeStructure(1,0, 0, *E*);

**this**.x=x;

**this**.ex = **this**.f.pow(*round*(x,3));

**this**.free = **this**.f.pow(0);

**this**.minusX = -1 \* x;

//function e^x - 2 - x

**this**.fun = **new** DerivativeStructure(1,ex,-2,free,minusX,free);

**return** *round*(**this**.fun.getValue(),4);

}

**double** **getTangentValue**(**double** x){

f = **new** DerivativeStructure(1,0, 0, *E*);

**this**.x=x;

**this**.ex = **this**.f.pow(*round*(x,3));

**this**.free = **this**.f.pow(0);

**this**.minusX = -1 \* x;

//function e^x - 2 - x

**this**.fun2 = **new** DerivativeStructure(2,free,-1,ex);

**return** *round*(**this**.fun2.getValue(),4);

}

}

Secant.java

**package** mhallman.methods.secant;

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**public** **class** **Secant** {

**private** **double** ps;//starting point

**private** **double** pk;//ending point

**double** fx1;

**double** fx0;

**double** x2;

**int** i=0;

**double** zeroValue= -1.8414;

**double** zeroValue1= 1.1462;

**double** mainZeroValue=0;

**private** **Function** function;

**private** **int** iterations;

**private** **double** nextValues[];

**public** **Secant**(){

function = **new** Function();

}

**public** **static** **double** **round**(**double** value, **int** places) {

**if** (places < 0) **throw** **new** IllegalArgumentException();

**BigDecimal** **bd** = **new** BigDecimal(value);

bd = bd.setScale(places, *RoundingMode*.***HALF\_UP***);

**return** bd.doubleValue();

}

**public** **int** **getIterations**() {

**return** iterations;

}

**public** **void** **setIterations**(**int** iterations) {

**this**.iterations = iterations;

}

/\*\*

\* **@return** the ps

\*/

**public** **double** **getPs**() {

**return** ps;

}

/\*\*

\* **@param** ps the ps to set

\*/

**public** **void** **setPs**(**double** ps) {

**this**.ps = ps;

}

/\*\*

\* **@return** the pk

\*/

**public** **double** **getPk**() {

**return** pk;

}

/\*\*

\* **@param** pk the pk to set

\*/

**public** **void** **setPk**(**double** pk) {

**this**.pk = pk;

}

**public** **double** **nextValue**(**double** x0, **double** x1){

fx1=**this**.function.value(x1);

fx0=**this**.function.value(x0);

**double** **value** = (x1 - (fx1\*(x1-x0))/(fx1-fx0));

**return** *round*(value,4);

}

**public** **void** **solveSecant**(){

**double** **x0**=**this**.getPs();

**double** **x1**=**this**.getPk();

nextValues = **new** **double**[100];

**do**{

x2=**this**.nextValue(x0,x1);

x0=x1;

x1=x2;

nextValues[i]=x2;

i++;

**System**.***out***.println("Kolejny wynik: "+x2);

}**while**(mainZeroValue-x2>0.0001);

**System**.***out***.println("Potrzeba było tyle: "+i+" obrotów");

}

**public** **double** **getMainZeroValue**() {

**return** mainZeroValue;

}

**public** **void** **setMainZeroValue**(**double** mainZeroValue) {

**this**.mainZeroValue = mainZeroValue;

}

SimpleIterations.java

**package** mhallman.methods.secant;

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**public** **class** **SimpleIterations** {

**private** **double** startingPoint;

**private** **int** stepNumber;

**private** **double** functionResult;

**private** **double** result;

**int** i=0;

**double** mainZeroValue=0;

**private** **Function** function = **new** Function();

**public** **static** **double** **round**(**double** value, **int** places) {

**if** (places < 0) **throw** **new** IllegalArgumentException();

**BigDecimal** **bd** = **new** BigDecimal(value);

bd = bd.setScale(places, *RoundingMode*.***HALF\_UP***);

**return** bd.doubleValue();

}

**public** **int** **getStepNumber**() {

**return** stepNumber;

}

**public** **void** **setStepNumber**(**int** stepNumber) {

**this**.stepNumber = stepNumber;

}

**public** **double** **getStartingPoint**() {

**return** startingPoint;

}

**public** **void** **setStartingPoint**(**double** startingPoint) {

**this**.startingPoint = startingPoint;

}

**public** **void** **solveIteration**() {

**do**{

functionResult = function.getTangentValue(startingPoint);

startingPoint = *round*(functionResult,4);

**System**.***out***.println("Kolejny wynik: "+startingPoint);

i++;

}**while**(mainZeroValue-startingPoint>0.0001);

**System**.***out***.println("Potrzeba było tyle: "+i+" obrotów");

}

**public** **double** **getMainZeroValue**() {

**return** mainZeroValue;

}

**public** **void** **setMainZeroValue**(**double** mainZeroValue) {

**this**.mainZeroValue = mainZeroValue;

}

}

Tangent.java

**package** mhallman.methods.secant;

**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**public** **class** **Tangent** {

**private** **double** startingPoint;

**private** **int** stepNumber;

**private** **double** functionResult;

**double** mainZeroValue=0;

**int** i=0;

**private** **Function** function = **new** Function();

**public** **static** **double** **round**(**double** value, **int** places) {

**if** (places < 0) **throw** **new** IllegalArgumentException();

**BigDecimal** **bd** = **new** BigDecimal(value);

bd = bd.setScale(places, *RoundingMode*.***HALF\_UP***);

**return** bd.doubleValue();

}

**public** **double** **getStartingPoint**() {

**return** startingPoint;

}

**public** **void** **setStartingPoint**(**double** startingPoint) {

**this**.startingPoint = startingPoint;

}

**public** **int** **getStepNumber**() {

**return** stepNumber;

}

**public** **void** **setStepNumber**(**int** stepNumber) {

**this**.stepNumber = stepNumber;

}

**public** **void** **solveTangent**() {

**do**{

functionResult = startingPoint - (function.value(startingPoint) / function.getDerivativeValue(startingPoint));

startingPoint = *round*(functionResult,4);

**System**.***out***.println("Kolejny wynik: "+startingPoint);

i++;

}**while**(mainZeroValue-startingPoint>0.0001);

**System**.***out***.println("Potrzeba było tyle: "+i+" obrotów");

}

**public** **double** **getMainZeroValue**() {

**return** mainZeroValue;

}

**public** **void** **setMainZeroValue**(**double** mainZeroValue) {

**this**.mainZeroValue = mainZeroValue;

}

}