Metody Obliczeniowe Zadanie 1.14

Uniwersytet Gdański, Informatyka III rok

Hallman , Górski, Rzeppa

Zadanie: - Pobrać wartości funkcji $f^{(n)}(x)$ w punktach węzłowych siatki. W węzłach x_k, $k = 0,\ldots,n-1$ o krotności a_k koniecznie podać $f^{(j)}(x_k)$, $j = 0,\ldots,a_k$ - podać wielomian interpolacyjny Hermite - Narysować wielomian f(x) i wielomian interpolacyjny na jednym wykresie

Wielomian interpolacyjny Hermite'a funkcji f

Twierdzenie:

Niech dany będzie ciąg liczb $u_0 \leq u_1, \leq \ldots \leq u_n$. Dla dowolnego ciągu liczb $c_0, c_1, \ldots, c_n$, istnieje dokładnie jeden wielomian W stopnia co najwyżej n, taki że jeśli liczba u_k w ciągu $u_0, \ldots, u_n$ występuje r razy, a dokładniej, jeśli $u_k = \cdots = u_{k+r-1}$ i z $k > 0$ wynika $u_{k-1} \neq u_k$ oraz z $k + r \leq n$ wynika $u_{k+r-1} \neq u_{k+r}$ (liczbę u_k nazywamy wtedy węzłem r–krotnym), który spełnia:

$$W(u_k) = c_k,\ W'(u_k) = c_{k+1},\ \ldots,\ W^{(r-1)}(u_k) = c_{k+r-1}$$

Definicja:

Wielomian W stopnia co najwyżej n, nazywamy wielomianem interpolacyjnym Hermite'a funkcji f, jeśli w każdym r–krotnym węźle u_k spełnia równania:

$$W(u_k) = f(u_k),\ W'(u_k) = f'(u_k),\ \ldots,\ W^{(r-1)}(u_k) = f^{(r-1)}(u_k).$$

Z powyższej definicji widzimy, iż pojęcie wielomianu interpolacyjnego Hermite'a jest uogólnieniem pojęcia „zwykłego" wielomianu interpolacyjnego Lagrange'a, który narzucał jedynie równość wartości wielomianu i funkcji w danych punktach. Przedstawione wcześniej twierdzenie gwarantuje, że wielomian taki wyznaczony jest w sposób jednoznaczny.

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.

- Kolejne punkty znajdujące się w podanym przedziale - program wysyła komunikat do użytkownika: „*Podaj kolejno węzły znajdujące się w przedziale...",* porównuje wprowadzone dane do *double*

- Krotności punktów - program wysyła komunikat do użytkownika: „*Podaj krotność węzła",* porównuje wprowadzone dane do *int*

- Aby rozpocząć liczenie wielomianu, program wymaga od użytkownika wpisania polecenia: „*licz"*

Wyjście:

- Wielomian - program wypisuje wynik obliczeń w postaci nieuporządkowanego wielomianu.

Klasy:

- Derivative

- Hermite

- Interpolation

Derivative:

Klasa zajmująca się wyliczaniem wartości pochodnych w punkcie.

public Derivative(Point point) - metoda implementująca stałą funkcję

public ArrayList<Double> makeDerivatives() - uzupełnia tabelę wartościami wyliczanymi z podanych przez użytkownika danych

Hermite:

Klasa wczytująca dane i porządkująca je.

public Hermite readPoints() - metoda zczytuje przedział oraz punkty

public Hermite fillDirevativesValues() - przygotowuje dane wpisując wartości

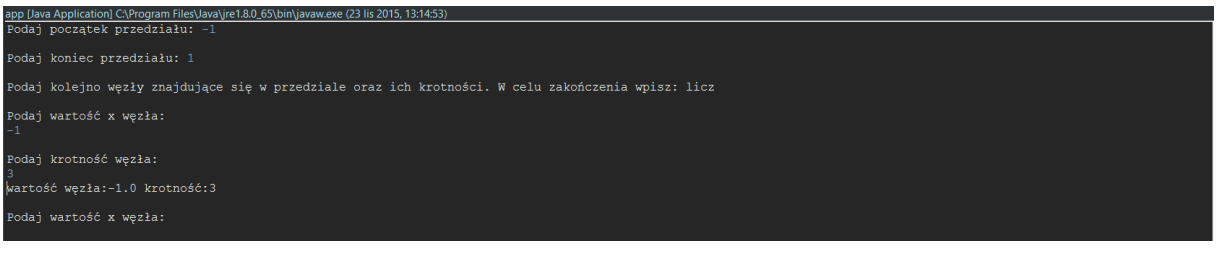
Interpolation:

Klasa zajmująca się obliczaniem ilorazów różnicowych oraz interpolowaniem metodą Hermite'a.

public double[] quotients(Hermite hermit, double[] nodes) - oblicza ilorazy różnicowe

public Interpolation(Hermite hermit) - metoda wyliczająca oraz wypisująca wielomian

Zrzuty ekranu dokumentujące działanie programu



*Podawanie przedziałów i początkowego punktu

```
app [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (23 lis 2015, 13:14:53)

Podaj wartość x węzła:
0

Podaj krotność węzła:
3
wartość węzła:0.0 krotność:3

Podaj wartość x węzła:
1

Podaj krotność węzła:
3
wartość węzła:1.0 krotność:3
```

*Dalsze podawanie punktów oraz krotności

```
<terminated> app [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (23 lis 2015, 13:14:53)
Wybany przedział to [-1,1]
wartość węzła:-1.0 krotność:3
wartość węzła:0.0 krotność:3
wartość węzła:1.0 krotność:3
Wartość funkcji w punkcie: -1.0 wynosi =7.0
Wartość pochodnej rzędu:1 dla funkcji w punkcie: -1.0 wynosi =-24.0
Wartość pochodnej rzędu:2 dla funkcji w punkcie: -1.0 wynosi =66.0

Wartość funkcji w punkcie: 0.0 wynosi =1.0
Wartość pochodnej rzędu:1 dla funkcji w punkcie: 0.0 wynosi =2.0
Wartość pochodnej rzędu:2 dla funkcji w punkcie: 0.0 wynosi =6.0

Wartość funkcji w punkcie: 1.0 wynosi =11.0
Wartość pochodnej rzędu:1 dla funkcji w punkcie: 1.0 wynosi =28.0
Wartość pochodnej rzędu:2 dla funkcji w punkcie: 1.0 wynosi =66.0
```

*Drukowanie wprowadzonych przez użytkownika danych oraz wyliczenie wartości funkcji/pochodnych

```
<terminated> app [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (23 lis 2015, 13:14:53)
Wartość t[0]: -1.0
Wartość t[1]: -1.0
Wartość t[2]: -1.0
Wartość t[3]: 0.0
Wartość t[4]: 0.0
Wartość t[5]: 0.0
Wartość t[6]: 1.0
Wartość t[7]: 1.0
Wartość t[8]: 1.0
```

*Drukowanie węzłów potrzebnych do wyliczenia ilorazów

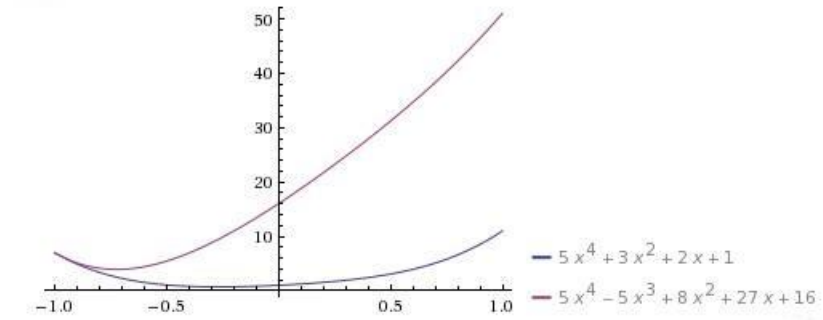
```
f[t0][t1]:-24.0  f[t0][t2]:33.0  f[t0][t3]:-15.0  f[t0][t4]:5.0  f[t0][t5]:0.0  f[t0][t6]:0.0  f[t0][t7]:0.0  f[t0][t8]:0.0
f[t1][t1]:-24.0  f[t1][t2]:18.0  f[t1][t3]:-10.0  f[t1][t4]:5.0  f[t1][t5]:0.0  f[t1][t6]:0.0  f[t1][t7]:0.0
f[t2][t1]:-6.0  f[t2][t2]:8.0  f[t2][t3]:-5.0  f[t2][t4]:5.0  f[t2][t5]:0.0  f[t2][t6]:0.0
f[t3][t1]:2.0  f[t3][t2]:3.0  f[t3][t3]:5.0  f[t3][t4]:5.0  f[t3][t5]:0.0
f[t4][t1]:2.0  f[t4][t2]:8.0  f[t4][t3]:10.0  f[t4][t4]:5.0
f[t5][t1]:10.0  f[t5][t2]:18.0  f[t5][t3]:15.0
f[t6][t1]:28.0  f[t6][t2]:33.0
f[t7][t1]:28.0
```

*Drukowanie ilorazów wyliczonych przez program. Wyróżniona jest górna krawędź zawierająca współczynniki wielomianu

```
P(x)= 7.0 + -24.0(x + 1.0) + 33.0(x + 1.0)(x + 1.0) + -15.0(x)(x + 1.0)(x + 1.0) + 5.0(x)(x)(x + 1.0)(x + 1.0) + 0.0(x)(x)(x)(x + 1.0)(x + 1.0) + 0.0(x - 1.0)(x)(x)(x)(x + 1.0)(x + 1.0)
```

*Drukowanie nieuporządkowanego wielomianu

plot $\frac{5x^4+3x^2+2x+1}{5x^4-5x^3+8x^2+27x+16}$ $x=-1$ to 1



app.java

```
package methods.hermite.main;

public class app {

	public static void main(String[] args) {
		Hermite hermite = new Hermite();
		hermite.readPoints();
		System.out.println("Wybany przedział to
["+hermite.getP()+","+hermite.getK()+"]");
		for(int i=0;i<hermite.getPointsNumber();i++){
			System.out.println(hermite.getPoint(i));
		}
		hermite.fillDirevativesValues();
		Interpolation interpolation = new Interpolation(hermite);
	}
```

Derivative.java

```
package methods.hermite.main;

import java.util.ArrayList;

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

public class Derivative {
	private double x;
	private int a;
	private DerivativeStructure polymonStruct;
	private DerivativeStructure x8;
	private DerivativeStructure x3;
	private DerivativeStructure x2;
	private DerivativeStructure x1;
	private DerivativeStructure free;
	private DerivativeStructure polymon;

	/**
	 * Constructor with argument
	 * @param point
	 */
	public Derivative(Point point){
		this.x=point.getX();
		this.a=point.getA();
		polymonStruct = new DerivativeStructure(1,a, 0, x);
		x3 = polymonStruct.pow(4);
		x2 = polymonStruct.pow(2);
		x1 = polymonStruct.pow(1);
		free = polymonStruct.pow(0);
		//function is 5x^4 + 3x^2 + 2x + 1
		polymon = new DerivativeStructure(5.0, x3, 3.0, x2, 2.0, x1,
1.0,free);

		//function is x^8 + 1
		//x8 = polymonStruct.pow(8);
		//polymon = new DerivativeStructure(1.0, x8, 1.0,free);
	}

	/**
	 *
	 * @return
	 */
	public ArrayList<Double> makeDerivatives(){
		ArrayList<Double> listY = new ArrayList<Double>();
		double value = 0;
		value=this.polymon.getValue();
		System.out.println("Wartość funkcji w punkcie: "+this.x+"
wynosi ="+value);
		listY.add(value);
		for(int i = 1; i<this.a;i++){
			value=this.polymon.getPartialDerivative(i);
			System.out.println("Wartość pochodnej rzędu:"+i+" dla
funkcji w punkcie: "+this.x+" wynosi ="+value);
			listY.add(value);
		}
		System.out.println();

		return listY;
```

Hermite.java

```
package methods.hermite.main;

import java.util.ArrayList;
import java.util.Scanner;
import org.apache.commons.math3.analysis.interpolation.HermiteInterpolator;


public class Hermite {

	private int p;
	private int k;
	private ArrayList<Point> points = new ArrayList<Point>();
	

	public Hermite(){
		
	}
	
	public int getPointsNumber(){
		return this.points.size();
	}
	

	public Point getPoint(int i){
		return this.points.get(i);
	}
	
	/**
	 * @return the p
	 */
	public int getP() {
		return p;
	}

	/**
	 * @param p the p to set
	 */
	public void setP(int p) {
		this.p = p;
	}

	/**
	 * @return the k
	 */
	public int getK() {
		return k;
	}

	/**
	 * @param k the k to set
	 */
	public void setK(int k) {
		this.k = k;
	}
	
	// funkcja poszukiwania j-tej pochodnej w punkcie x
    public double getYfromX(double x, int j) {
        for (int i=0;i<this.points.size();i++)
            if (this.points.get(i).getX() == x)
```

```
                return this.points.get(i).getNthY(j);
        return 0;
    }

    /**
     * Reads input points
     * @return
     */
    public Hermite readPoints(){

            Scanner in = new Scanner(System.in);
            System.out.print("Podaj początek przedziału: ");
            if(in.hasNextInt()){
                  this.p=in.nextInt();
            }else{
                  System.err.println("Podano nieprawidłową wartość!!");
                  System.exit(0);
            }
            System.out.println();
            System.out.print("Podaj koniec przedziału: ");
            if(in.hasNextInt()){
                  this.k=in.nextInt();
                  if(this.getK()==this.getP() || this.getK()<this.getP()){
                        System.err.println("Podano nieprawidłową
wartość!!");
                        System.exit(0);
                  }
            }
            System.out.println();
            System.out.println("Podaj kolejno węzły znajdujące się w
przedziale oraz ich krotności. W celu zakończenia wpisz: licz \n");
            System.out.println("Podaj wartość x węzła: ");
            while(in.hasNextDouble()){
                              double x = in.nextDouble();
                              if(x>=this.getP() && x<= this.getK()){
                              Point point = new Point();
                              point.setX(x);
                              System.out.println();
                              System.out.println("Podaj krotność węzła: ");
                              if(in.hasNextInt()){
                                    int a = in.nextInt();
                                    point.setA(a);
                                    points.add(point);
                                    System.out.println(point);
                                    System.out.println();
                              }
                              System.out.println("Podaj wartość x węzła: ");
                  }else{
                        System.out.println("Liczba nie mieści się w
przedziale, program kończy działanie!!");
                        System.exit(0);
                  }
            }
            return this;
      }

      /**
       * Fills list of values with partial derivatives
       * @return
       */
      public Hermite fillDirevativesValues(){
            HermiteInterpolator polynomial = new HermiteInterpolator();
            for(int i=0;i<this.getPointsNumber();i++){
                  Derivative diver = new Derivative(this.getPoint(i));
                  this.getPoint(i).setY(diver.makeDerivatives());
            }
```

```
            return this;
      }
```

Point.java

```
package methods.hermite.main;

import java.util.ArrayList;

public class Point {

      private double x;
      private int a;
      private ArrayList<Double> y;


      /**
       * Constructor without arguments
       */
      public Point(){

      }


      /**
       * Constructor with one argument
       * @param x
       */
      public Point(double x, int a){
            super();
            this.x=x;
            this.a=a;
      }

      /**
       * @return the x
       */
      public double getX() {
            return x;
      }

      /**
       * @param x the x to set
       */
      public void setX(double x) {
            this.x = x;
      }

      /**
       * @return the y
       */
      public ArrayList<Double> getY() {
            return y;
      }

      /**
       * @param y the y to set
       */
      public void setY(ArrayList<Double> y) {
            this.y = y;
      }


      /**
       * @param y
```

```
	 */
	public void addY(double y){
		this.y.add(y);
	}
	/**
	 * @param i
	 * @return the n-th element from y list
	 */
	public double getNthY(int i){
		return this.y.get(i);
	}
	/**
	 * @return the size of y list
	 */
	public int getSize(){
		return this.y.size();
	}

	/**
	 * @return the a
	 */
	public int getA() {
		return a;
	}

	/**
	 * @param a the a to set
	 */
	public void setA(int a) {
		this.a = a;
	}
	@Override
	public String toString(){
		return "wartość węzła:"+this.getX()+" krotność:"+this.getA()+"
";
	}
}
```

Interpolation.java

```
package methods.hermite.main;

import java.util.ArrayList;

import
org.apache.commons.math3.analysis.differentiation.DerivativeStructure;
import org.apache.commons.math3.analysis.interpolation.HermiteInterpolator;

public class Interpolation {
	private String polymonial;

	public Interpolation(){

	}
	/**
	 * @return the polynomial
```

```
	 */
	public String getPolymonial() {
		return polymonial;
	}

	/**
	 * @param polynomial the polynomial to set
	 */
	public void setPolymonial(String polymonial) {
		this.polymonial = polymonial;
	}
	// obliczanie ilorazow roznicowych
    public double[] quotients(Hermite hermit,  double[] nodes) throws
Exception {
	int nodesNumber=0; // ilosc punktow
	int k = 0;  // wypelnianie pierwszego wiersza
	  for (int i = 0; i < hermit.getPointsNumber();i++) {
		nodesNumber += hermit.getPoint(i).getSize(); // liczymy ilosc
pochodnych
	    }
	    System.out.println();
	    double[][] quotients = new double[nodesNumber][]; // tworzenie
kolumn ilorazów jak w tej tabelce ze skryptu
	    for (int i = 0; i < nodesNumber; i++){ // tworzenie wierszy
		quotients[i] = new double[nodesNumber-i]; //tworzy sie
tablicowa piramida
	    }
	    for(int i = 0; i < hermit.getPointsNumber(); i++) {
		for (int j = 0; j < hermit.getPoint(i).getSize(); j++) {
		    quotients[k][0] = hermit.getPoint(i).getNthY(0);
//wypełniamy pierwszy wiersz wartościamy funkcji f(x) dla danego x
		    k++;
		}
	    }
	    int factorial = 1; // wypelnianie reszty wierszy
	    for (int j = 1; j < nodesNumber; j++) {
		factorial *= j;
		for (int i = 0; i < nodesNumber - j; i++) {
		    if (nodes[i+j] - nodes[i] == 0) { // jezeli w mianowniku
zero
			quotients[i][j] = hermit.getYfromX(nodes[i+j],j) /
factorial; // metoda z silnią i pochodnymi
		    }
		    else { // zwykla metoda
			quotients[i][j] = (quotients[i+1][j-1] - quotients[i][j-
1]) / (nodes[i+j] - nodes[i]); //inaczej normalna metoda
		    }
		}
	    }
	    int r=1;
	    for (int i = 0; i < nodesNumber; i++) {
		for (int j = 1; j < nodesNumber-i; j++) {
		    System.out.print("  f[t"+i+"][t"+j+"]:"+quotients[i][j]);
		}
		System.out.println();
		}
	    double[] diagonal = quotients[0]; // zwrocenie gornej przekatnej z
której budujemy wielomian. Chodzi o współczynniki
	    String polynom = "P(x)= ";
	    for(int i=0;i<diagonal.length;i++){//testowe wypisanie wielomianu
		polynom += diagonal[i]+"";
		int l=i;
		while(l>0){
		    if(nodes[l]>0){
			polynom += "(x - "+nodes[l]+")";
		    }
```

```
                else if(nodes[l]<0){
                        polynom += "(x + "+nodes[l]*(-1)+")";
                }else{
                        polynom += "(x)";
                }
                l--;
            }
            /*if(i<diagonal.length){
                if(diagonal[i+1]<0){
                        polynom+=" - ";
                }else if(diagonal[i+1]>0){
                        polynom+=" + ";
                }else{
                        i++;
                }
            }*/
            polynom+=" + ";
        }
        System.out.print(polynom);
       /* for(int i=0;i<diagonal.length;i++){
            System.out.print(diagonal[i]+" ");
        }*/
        System.out.println();
        Polynomial w = Polynomial.NewtonFormToPolynomial(nodes, diagonal);
        System.out.println();
        System.out.println(w);
        return diagonal;
    }

          // interpolacja Hermite'a
         public Interpolation(Hermite hermit){
           //System.out.println(hermit.getPointsNumber());
           int k = 0;  // wypelnianie tablicy
           int n=0;
           for (int i = 0; i < hermit.getPointsNumber(); i++) {
                for (int j = 0; j <hermit.getPoint(i).getSize(); j++) {
                    n++;//zliczanie węzłów
                }
            }
           double[] nodes = new double[n]; //tutaj beda te nasze wezly t0
t1 itp
             for (int i = 0; i < hermit.getPointsNumber(); i++) {
                 for (int j = 0; j <hermit.getPoint(i).getSize(); j++) {
                     nodes[k] = hermit.getPoint(i).getX();
                     k++;
                 }
             }
             for(int l=0;l<nodes.length;l++){
                 System.out.println("Wartość t["+l+"]:
"+nodes[l]);//wypisywanie węzłów
             }
             try {
                     double[] quotientss = quotients(hermit,nodes);
                 } catch (Exception e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                 } // obliczenie ilorazow roznicowych
             this.setPolymonial("P(X) = "+" ");
         }
```