Diogo Bogas (201202482), Diogo Fernandes (201403232),

Iúri Pena (201304604), João Ferreira (201304824)

Análise Numérica

Unidade Prática 3



Departamento de Matemática

Faculdade de Ciências da Universidade do Porto

Março de 2017

1. Dado e um conjunto de pontos de abcissas distintas

Ambos os programas foram testados, pelo menos, com os exemplos dos slides da disciplina.

Existe algum erro associado aos resultados devido à conversão de frações para números decimais.

* 1. Método de Lagrange

Compilação: javac trab3a.java

Execução: java trab3a

**import** **java.util.Scanner**;

**public** **class** **trab3a** {

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

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

System.out.println("Inserir o valor de n");

**int** n = in.nextInt();

**float** table[][] = **new** **float**[n+**1**][**2**];

/\* ler os dados \*/

**for**(**int** i=**0**; i<n+**1**; i++) {

System.out.println("Para i = " +i+ ": inserir ni f(ni)");

table[i][**0**] = in.nextFloat();

table[i][**1**] = in.nextFloat();

}

System.out.println("Deseja:\n1. Constuir polinómio interpolador\n2. Calcular para x dado");

**int** flag = in.nextInt();

/\* imprimir polinómio \*/

**if**(flag == **1**) {

System.out.print("p"+n+"(x) = ");

**for**(**int** i=**0**;i<n+**1**;i++) {

printPol(n, i, table);

**if**(i==n)

System.out.println(" \* "+table[i][**1**]);

**else**

System.out.print(" \* "+table[i][**1**]+" + ");

}

}

/\* calcular polinómio \*/

**else** {

System.out.println("Indicar o valor de X");

**float** x = in.nextFloat();

**float** res = **0**;

**for**(**int** i=**0**; i<n+**1**;i++)

res += calcPol(n, i, table, x) \* table[i][**1**];

System.out.println(res);

}

}

/\* imprimir o polinómio Lk \*/

**static** **void** **printPol**(**int** n, **int** k, **float** [][] table) {

System.out.print("(");

**for**(**int** i=**0**; i<n+**1**;i++)

**if**(i!=k)

System.out.print("(x-"+table[i][**0**]+")");

System.out.print(")/(");

**for**(**int** i=**0**; i<n+**1**;i++)

**if**(i!=k)

System.out.print("("+table[k][**0**]+"-"+table[i][**0**]+")");

System.out.print(")");

}

/\* calcular o valor de Lk \*/

**static** **float** **calcPol**(**int** n, **int** k, **float** [][] table, **float** x) {

**float** num = **1**, den = **1**;

**for**(**int** i=**0**;i<n+**1**;i++)

**if**(i!=k) {

num \*= (x-table[i][**0**]);

den \*= (table[k][**0**]-table[i][**0**]);

}

**return** num/den;

}

}

Este programa não só calcula o polinómio interpolador como também o imprime, caso o utilizador indicar.

De notar que os valores a inserir correspondem aos valores de .

* 1. Spline cúbico natural

Compilação: javac trab3b.java && javac SplineInterpolator.java

Execução: java trab3b

Ficheiro trab3b.java:

**import** **java.util.Scanner**;

**import** **java.util.List**;

**import** **java.util.ArrayList**;

**class** **trab3b** {

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

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

System.out.println("Valor de n");

**int** n = in.nextInt();

/\* n+1 pontos da função \*/

**float** pontos[][] = **new** **float**[n+**1**][**2**];

**for**(**int** i=**0**; i<n+**1**; i++) {

System.out.println("Para i = " +i+ ": inserir ni f(ni)");

pontos[i][**0**] = in.nextFloat();

pontos[i][**1**] = in.nextFloat();

}

/\* sistema que define o spline | valores de M | spline \*/

**float**[][] system = makeSystem(n,pontos);

//float[] m = calcSystem(n, pontos);

//float[][] spline = makeSpline(n, pontos, m);

System.out.println("Deseja:\n1. Constuir spline cúbico natural\n2. Calcular para x dado");

**int** flag = in.nextInt();

/\* imprimir sistema e spline \*/

**if**(flag == **1**) {

System.out.println("\nSistema que determina o spline:");

printSystem(n, system);

System.out.println("\nS(x):");

// printSpline(n, spline);

}

/\* calcular sistema e spline \*/

**else** {

List<Float> mX = **new** ArrayList<>();

List<Float> mY = **new** ArrayList<>();

**for**(**int** i=**0**; i<n+**1**; i++) {

mX.add(pontos[i][**0**]);

mY.add(pontos[i][**1**]);

}

System.out.println("Indicar o valor de X");

**float** x = in.nextFloat();

SplineInterpolator s = SplineInterpolator.createMonotoneCubicSpline(mX, mY);

System.out.println("Resultado: "+s.interpolate(x));

/\* TODO

check interval

calcular sistema

calcular spline

calcular equação com o valor de x

\*/

}

}

/\*\*

\* Construir o sistema que determina o spline.

\*

\* h\_i/6 \* M\_i-1 + (h\_i+h\_i+1)/3 \* M\_i + h\_i+1/6 \* M\_i+1 =

\* = (f\_i+1 - f\_i)/h\_i+1 - (f\_i - f\_i-1)/h\_i

\*

\* Para i=1 até n-1

\*/

**static** **float**[][] **makeSystem**(**int** n, **float** t[][]) {

**float** s[][] = **new** **float**[n][n+**2**];

/\* M0 = 0 --> a linha 0 fica a 0.

Mn = 0 --> a linha n fica a 0.

M\_1 a M\_n-1

\*/

**for**(**int** i=**1**; i<n; i++) {

**float** hi = t[i][**0**]-t[i-**1**][**0**];

**float** hi1 = t[i+**1**][**0**]-t[i][**0**];

/\* parte direita da equação \*/

**float** right = ( (t[i+**1**][**1**]-t[i][**1**]) / hi1 ) - ( (t[i][**1**]-t[i-**1**][**1**]) / hi );

**for**(**int** j=**0**; j<n+**2**; j++)

**if**(j == i-**1**)

s[i][j] = hi/**6**;

**else** **if**(j == i)

s[i][j] = (hi+hi1)/**3**;

**else** **if**(j == i+**1**)

s[i][j] = hi1/**6**;

**else** **if**(j == n+**1**)

s[i][j] = right;

}

**return** s;

}

**static** **void** **printSystem**(**int** n, **float** s[][]) {

**for**(**int** i=**1**; i<n; i++){

System.out.print(i-**1**+": ");

**for**(**int** j=**0**; j<n+**2**; j++)

**if**(j == n+**1**)

System.out.println(s[i][j]);

**else** **if**(j==n)

System.out.print(s[i][j]+"\*M"+(j)+" = ");

**else**

System.out.print(s[i][j]+"\*M"+(j)+" + ");

}

System.out.println(n-**1**+": M0 = 0");

System.out.println(n+": M"+n+" = 0");

}

}

O que comentários com “/\*\*/” são efetivamente comentários, enquanto que os comentários com “//” correspondem a partes não implementadas do programa. Inicialmente tinha-se como objetivo escrever todo o programa, contudo surgiram várias dificuldades aquando do cálculo do sistema que define o spline e, por este motivo, recorremos a uma outra class (SplineInterpolator) - encontrando-se sob a licença Apache 2.0 e não tendo sido feitas quaisquer alterações à mesma - para efetuar o cálculo do spline interpolador para um valor de dado. Desta forma, é apenas possível imprimir o sistema que define o spline.

Ficheiro SplineInterpolator.java

/\*

\* Copyright (C) 2012 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

**import** **java.util.List**;

**class** **SplineInterpolator** {

**private** **final** List<Float> mX;

**private** **final** List<Float> mY;

**private** **final** **float**[] mM;

**private** **SplineInterpolator**(List<Float> x, List<Float> y, **float**[] m) {

mX = x;

mY = y;

mM = m;

}

/\*\*

\* Creates a monotone cubic spline from a given set of control points.

\*

\* The spline is guaranteed to pass through each control point exactly. Moreover, assuming the control points are

\* monotonic (Y is non-decreasing or non-increasing) then the interpolated values will also be monotonic.

\*

\* This function uses the Fritsch-Carlson method for computing the spline parameters.

\* http://en.wikipedia.org/wiki/Monotone\_cubic\_interpolation

\*

\* @param x

\* The X component of the control points, strictly increasing.

\* @param y

\* The Y component of the control points

\* @return

\*

\* @throws IllegalArgumentException

\* if the X or Y arrays are null, have different lengths or have fewer than 2 values.

\*/

**public** **static** SplineInterpolator **createMonotoneCubicSpline**(List<Float> x, List<Float> y) {

**if** (x == **null** || y == **null** || x.size() != y.size() || x.size() < **2**) {

**throw** **new** **IllegalArgumentException**("There must be at least two control "

+ "points and the arrays must be of equal length.");

}

**final** **int** n = x.size();

**float**[] d = **new** **float**[n - **1**]; // could optimize this out

**float**[] m = **new** **float**[n];

// Compute slopes of secant lines between successive points.

**for** (**int** i = **0**; i < n - **1**; i++) {

**float** h = x.get(i + **1**) - x.get(i);

**if** (h <= **0**f) {

**throw** **new** **IllegalArgumentException**("The control points must all "

+ "have strictly increasing X values.");

}

d[i] = (y.get(i + **1**) - y.get(i)) / h;

}

// Initialize the tangents as the average of the secants.

m[**0**] = d[**0**];

**for** (**int** i = **1**; i < n - **1**; i++) {

m[i] = (d[i - **1**] + d[i]) \* **0.5f**;

}

m[n - **1**] = d[n - **2**];

// Update the tangents to preserve monotonicity.

**for** (**int** i = **0**; i < n - **1**; i++) {

**if** (d[i] == **0**f) { // successive Y values are equal

m[i] = **0**f;

m[i + **1**] = **0**f;

} **else** {

**float** a = m[i] / d[i];

**float** b = m[i + **1**] / d[i];

**float** h = (**float**) Math.hypot(a, b);

**if** (h > **9**f) {

**float** t = **3**f / h;

m[i] = t \* a \* d[i];

m[i + **1**] = t \* b \* d[i];

}

}

}

**return** **new** **SplineInterpolator**(x, y, m);

}

/\*\*

\* Interpolates the value of Y = f(X) for given X. Clamps X to the domain of the spline.

\*

\* @param x

\* The X value.

\* @return The interpolated Y = f(X) value.

\*/

**public** **float** **interpolate**(**float** x) {

// Handle the boundary cases.

**final** **int** n = mX.size();

**if** (Float.isNaN(x)) {

**return** x;

}

**if** (x <= mX.get(**0**)) {

**return** mY.get(**0**);

}

**if** (x >= mX.get(n - **1**)) {

**return** mY.get(n - **1**);

}

// Find the index 'i' of the last point with smaller X.

// We know this will be within the spline due to the boundary tests.

**int** i = **0**;

**while** (x >= mX.get(i + **1**)) {

i += **1**;

**if** (x == mX.get(i)) {

**return** mY.get(i);

}

}

// Perform cubic Hermite spline interpolation.

**float** h = mX.get(i + **1**) - mX.get(i);

**float** t = (x - mX.get(i)) / h;

**return** (mY.get(i) \* (**1** + **2** \* t) + h \* mM[i] \* t) \* (**1** - t) \* (**1** - t)

+ (mY.get(i + **1**) \* (**3** - **2** \* t) + h \* mM[i + **1**] \* (t - **1**)) \* t \* t;

}

// For debugging.

**@Override**

**public** String **toString**() {

StringBuilder str = **new** StringBuilder();

**final** **int** n = mX.size();

str.append("[");

**for** (**int** i = **0**; i < n; i++) {

**if** (i != **0**) {

str.append(", ");

}

str.append("(").append(mX.get(i));

str.append(", ").append(mY.get(i));

str.append(": ").append(mM[i]).append(")");

}

str.append("]");

**return** str.toString();

}

}