Project 2

CS 301-01

Pavankumar Aiyar
Professor Raheja

November 20, 2012

1 Source

```java
/**
 * @author Pavan Aiyar
 *
 *          File: Driver.java
 *
 *          Description: Driver class to demonstrate Newton Divided Difference
 *          program.
 *
 */
public class Driver {
    public static void main(String... args) {

        Newton n = new Newton("test.txt"); // pass input file
        n.solve();

        System.out.println("Divided Differences Table: ");
        n.printFormatted();

        System.out.println("Interpolating polynomial:");
        n.toPolynomial();

        System.out.println("\nSimplified polynomial:");
        n.toSimplified();

    }
}
```

Driver.java

```java
/**
 * @author Pavan Aiyar
 *
 *          File: Newton.java
 *
 *          Description: loads from file a line containing x values, and a line
 *          containing f(x) values, constructs and formats corresponding divided
 *          difference table, interpolating polynomial, and simplified polynomial
 *
 *          Note: ACCURATE TO 3 DECIMAL PLACES in respect to formatting. Change
 *          all instances of .3 and the round() method if you want a different accuracy
 *          but this will look terrible for larger (~50) node input.
 */

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class Newton {

    private double[][] table;
    private ArrayList<Double> coef;
    private String[][] f;
    private String filename;

    public Newton(String filename) {
        this.filename = filename;
        this.coef = new ArrayList<Double>();
        this.loadTable();
    }

    /**
     * Load table from file
     */
    public void loadTable() {
        FileInputStream fstream = null;
        try {
            fstream = new FileInputStream(this.filename);
        }
        catch (FileNotFoundException fn) {
```

```
            System.out.printf("File %s not found\n", this.filename);
        }
        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new InputStreamReader(in));

        String[] x = null;
        String[] y = null;

        try {
            x = br.readLine().split("\\s+");
            y = br.readLine().split("\\s+");
        }
        catch (IOException ie) {
            System.out.printf("Error reading contents of %s\n", this.filename);
        }

        if (x.length != y.length) {
            System.out.println("Error: Number of nodes do not match.");
            System.exit(0);
        }

        this.table = new double[x.length][x.length + 1];

        for (int i = 0; i < x.length; ++i) {
            table[i][0] = Double.parseDouble(x[i]);
        }// x to first column

        for (int i = 0; i < y.length; ++i) {
            table[i][1] = Double.parseDouble(y[i]);
        }// y to second column
    }

    /**
     * Solve the divided difference table
     */
    public void solve() {
        int n = this.table[0].length; // column length

        for (int j = 2; j < n; ++j) {
            for (int i = 0; i < n - j; ++i) {
                this.table[i][j] = (this.table[i + 1][j - 1] - this.table[i][j - 1])
                        / (this.table[i + (j - 1)][0] - this.table[i][0]);
            }
        }

        for (int i = 1; i < table[0].length; ++i) {
            this.coef.add(table[0][i]);
        }
    }

    // 3 decimal place rounding
    private double round(double a) {
        return (double) Math.round(a * 1000) / 1000;
    }

    /**
     * Convert the table into an interpolating polynomial, and print
     */
    public void toPolynomial() {
        ArrayList<String> xToken = new ArrayList<String>();
        String p = "";
        for (int i = 0; i < this.table.length - 1; ++i) {
            double t = this.table[i][0];

            if (t < 0) {
                p = "+";
            }
            else if (t > 0) {
                p = "-";
            }
            if (round(t) == 0) {
                xToken.add("(x)");
            }
            else {
                xToken.add(String.format("(x%s%.3f)", p, t));
            }
```

```
        }

        String poly = String.format("%.3f", this.coef.get(0));

        for (int i = 1; i < xToken.size() + 1; ++i) {
            double t = this.coef.get(i);
            if (t != 0) {
                if (t > 0) {
                    p = "+";
                }
                else {
                    p = "-";
                }
                String varx = "";
                for (int j = 0; j < i; ++j) {
                    varx += xToken.get(j);
                }
                poly += String.format(" %s %.3f%s", p, Math.abs(t), varx);
            }
        }
        System.out.println(poly);
    }

    /**
     * Simplify the table by using the polynomial class, and print
     */
    public void toSimplified() {
        Polynomial p = new Polynomial();
        ArrayList<Double> t = new ArrayList<Double>();

        ArrayList<ArrayList<Double>> matrix = new ArrayList<ArrayList<Double>>();
        for (int i = 0; i < this.table[0].length - 1; ++i) {
            t.add(0.0);
        }
        t.add(0, this.coef.get(0));
        matrix.add(t);

        for (int i = 1; i < this.coef.size(); ++i) {
            t = new ArrayList<Double>();
            double c = this.coef.get(i);
            for (int j = 0; j < i; ++j) {
                t.add(this.table[j][0]);
            }
            matrix.add(p.expandPoly(c, t, this.table[0].length));
        }

        t = p.compress(matrix);

        System.out.println(toSimpString(t));
    }

    private String toSimpString(ArrayList<Double> c) {
        String poly = "";
        String power = "";
        for (int i = 0; i < c.size(); ++i) {
            Double f = c.get(i);
            power = String.format("x^%d", i);
            if (f != 0) {
                if (i == 0) {
                    poly += String.format(" %+.3f", f);
                }
                else {
                    poly += String.format(" %+.3f%s", f, power);
                }
            }
        }

        return poly;
    }

    /**
     * Format table in side-ways pyramid form
     */
    private void formatTable() {
        int h = 2 * this.table.length;
        int w = this.table[0].length;
```

```
197
198         this.f = new String[h][w];
199
200         for (int i = 0; i < this.f.length; ++i) {
201             for (int j = 0; j < this.f[i].length; ++j) {
202                 this.f[i][j] = String.format("%8s", " ");
203             }
204         }
205
206         // Transfer column 1
207         int q = 0; // offset
208         for (int i = 0; i < this.table.length; ++i) {
209             this.f[q][0] = String.format("%8.3f", this.table[i][0]);
210             q += 2;
211         }
212         // Transfer column 2
213         q = 0; // offset
214         for (int i = 0; i < this.table.length; ++i) {
215             this.f[q][1] = String.format("%8.3f", this.table[i][1]);
216             q += 2;
217         }
218
219         int n = this.table[0].length;
220         for (int col = 2; col < n; ++col) {
221             q = (col - 1);
222             for (int row = 0; row < n - col; ++row) {
223                 f[q][col] = String.format("%8.3f", this.table[row][col]);
224                 q += 2;
225             }
226         }
227
228     }
229
230     // Print table in in side-ways pyramid form
231     public void printFormatted() {
232         formatTable();
233         for (String[] s : f) {
234             for (String b : s) {
235                 System.out.print(b + " ");
236             }
237             System.out.println();
238         }
239     }
240
241     public double[][] getTable() {
242         return this.table;
243     }
244
245     public ArrayList<Double> getCoef() {
246         return this.coef;
247     }
248
249     // DEBUG METHOD TO PRINT COEFFICIENTS
250     public void printCoef() {
251         for (Double d : this.coef) {
252             System.out.printf("%8.3f ", d);
253         }
254     }
255
256     // DEBUG METHOD TO PRINT ENTIRE TABLE
257     public void printTable() {
258         for (int row = 0; row < this.table.length; ++row) {
259             for (int column = 0; column < this.table[row].length; ++column) {
260                 System.out.printf("%8.3f ", this.table[row][column]);
261             }
262             System.out.println("\n");
263         }
264     }
265
266 }
```

Newton.java

```
1 /**
2  * @author Pavan Aiyar
```

```
 *
 *          File: Polynomial.java
 *
 *          Description: Class defines methods to expand polynomials strictly in
 *          the form c(x - x0)(x-x1)...(x-xn) for the purpose of providing a
 *          simplified polynomial representation of a given interpolating
 *          polynomial
 */

import java.util.ArrayList;

public class Polynomial {

    /**
     * Expand polynomial provided in form c*(x-t0)...(x-tn) return list of
     * coefficients x^0 ... x^n power
     */
    public ArrayList<Double> expandPoly(double c, ArrayList<Double> t,
            int minSize) {

        ArrayList<ArrayList<Double>> matrix = new ArrayList<ArrayList<Double>>();

        // coefficient array
        ArrayList<Double> g = new ArrayList<Double>();
        for (int i = 0; i < t.size() + 1; ++i) {
            g.add(0.0);
        }
        g.add(0, c); // put coefficient into this array

        for (int i = 0; i < t.size(); ++i) {
            matrix.add(push(g));
            matrix.add(mult(g, -t.get(i)));
            g = compress(matrix);
            matrix.clear();
        }

        int n = g.size();
        for (int i = 0; i < minSize - n; ++i) {
            g.add(0.0); // pad with 0s
        }

        return g;
    }

    // multiply by x (shift right)
    private ArrayList<Double> push(ArrayList<Double> a) {
        ArrayList<Double> t = new ArrayList<Double>();
        t.add(0.0);
        for (int i = 0; i < a.size() - 1; ++i) {
            t.add(a.get(i));
        }
        return t;
    }

    // multiply a by constant x
    private ArrayList<Double> mult(ArrayList<Double> a, double x) {
        ArrayList<Double> t = new ArrayList<Double>();
        for (double d : a) {
            t.add(d * x);
        }
        return t;
    }

    // Add like terms in coefficient matrix
    public ArrayList<Double> compress(ArrayList<ArrayList<Double>> a) {

        ArrayList<Double> compressed = new ArrayList<Double>();
        for (int i = 0; i < a.get(0).size(); ++i) {
            double sum = 0.0;
            for (int j = 0; j < a.size(); ++j) {
                sum += a.get(j).get(i);
            }
            compressed.add(sum);
        }

        return compressed;
```

```
    }

}
```

Polynomial.java

2 Test Cases

2.1 Test 1 Input.txt

```
1 1.5 0 2
3 3.25 3 1.67
```

input.txt

2.1.1 Input.txt Output

```
Divided Differences Table:
   1.000    3.000
                      0.500
   1.500    3.250              0.333
                      0.167             -1.997
   0.000    3.000            -1.663
                     -0.665
   2.000    1.670

Interpolating polynomial:
3.000 + 0.500(x-1.000) + 0.333(x-1.000)(x-1.500) - 1.997(x-1.000)(x-1.500)(x)

Simplified polynomial:
 +3.000 -3.328x^1 +5.325x^2 -1.997x^3
```

out.dat

2.2 Test 2 x^2 with 4 nodes

```
1 2 3 8
1 4 9 64
```

test2.in

2.2.1 Test 2 Output

```
Divided Differences Table:
   1.000    1.000
                      3.000
   2.000    4.000              1.000
                      5.000              0.000
   3.000    9.000              1.000
                     11.000
   8.000   64.000

Interpolating polynomial:
1.000 + 3.000(x-1.000) + 1.000(x-1.000)(x-2.000)

Simplified polynomial:
 +1.000x^2
```

test2.out

2.3 Test 3 50 nodes

Input file of 50 nodes generated by the function $f(x) = 7x^4 - 4x^3 + 3x^2 - 5$

```
...omitted with respect to page length
```

2.3.1 Input.txt Output

Table omitted ... input file provided in hand-in

```
Interpolating polynomial:
1.000 + 86.000(x-1.000) + 154.000(x-1.000)(x-2.000) + 66.000(x-1.000)(x-2.000)(x-3.000) + 7.000(x-1.000)(x
    -2.000)(x-3.000)(x-4.000)

Simplified polynomial:
 -5.000 +3.000x^2 -4.000x^3 +7.000x^4
```

test3.out

3 Compile Instructions

Should compile and run via:

```
javac Driver.java
java Driver
```

4 Notes

Formatting is optimized for given test cases. If a function requires more than 3 decimal places of precision, `Newton.java` must change all instances of ".3" to the desired accuracy, and modify the `round()` method accordingly. [1]

[1] Just do a search and replace ".3" with ".accuracy"