Relazione di “Metodi del Calcolo Scientifico”

Simon Vocella
Matricola: 718289

13 luglio 2012

1 Lu Decomposition

1.1 Teoria

1.2 Jama

1.3 Programma lu-decomposition

Listing 1: lu-decompostion

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.Scanner;

import Jama.EigenvalueDecomposition;
import Jama.Matrix;

class Main {
 public static String path = ”/home/simon/” +
              ”projects/metodi_calcolo_scientifico/LuEig/matrice/”;

 /*
  * Method that permit to read a file and
  * create a Jama Matrix
  */
 public static Matrix readMatrix(String file) {
  Matrix matrix = null;
  Scanner sc;
  try {
   sc = new Scanner(new File(file));
   int size = sc.nextInt();
   matrix = new Matrix(size, size);

   for (int i = 0; i < size * size; i++) {
    int x = sc.nextInt();
    int y = sc.nextInt();
   double d = Double.parseDouble(sc.next());
   matrix.set(x - 1, y - 1, d);
  }

  sc.close();
 } catch (FileNotFoundException e) {
  e.printStackTrace();
 }

 return matrix;
}

/*
 * Method that permit to read a file and
 * create a Jama Matrix with 1 columnn (an array)
 */
public static Matrix readArray(String file) {
 Matrix matrix = null;
 Scanner sc;
 try {
  sc = new Scanner(new File(file));
  int size = sc.nextInt();
  matrix = new Matrix(size, 1);

  for (int i = 0; i < size; i++) {
   int x = sc.nextInt();
   double d = Double.parseDouble(sc.next());
   matrix.set(x - 1, 0, d);
  }

  sc.close();
 } catch (FileNotFoundException e) {
  e.printStackTrace();
 }

 return matrix;
}

/*
 * Method that permit to get all eigenvalues
 * ordered from a Jama Matrix
 */
public static ArrayList<Double> getOrderedEigenValues(Matrix A) {
 double[][] values = A.getArray();
 ArrayList<Double> eigenValues = new ArrayList<Double>();

 for (int i = 0; i < values.length; i++) {
  eigenValues.add(values[i][i]);
 }

 Collections.sort(eigenValues);
 return eigenValues;
}

public static void main(String[] argv) {
 System.out
   .println("TEST           \t" +
              "ERRORE RELATIVO \t" +
              "ERRORE PRIMA COMP.\t" +
              "AUTOVALORE NUMERO 7\t" +
              "TIME TO SOLVE\t" +
              "TIME TO EIGEN");
 System.out
   .println("————————————\t" +
              "————————————————\t" +
              "——————————————————\t" +
              "———————————————————\t" +
              "—————————————\t" +
              "—————————————");

 /* Name of all files */
 String[] names = { "easy-10", "easy-100", "easy-1000", "bad-10",
   "bad-100", "bad-500", "bad-1000", "verybad-10", "verybad-100",
   "verybad-500", "verybad-1000", "rand-10", "rand-100",
   "rand-1000", "rand-5000", "eig-10", "eig-20", "eig-30",
   "eig-40", "eig-50", "eig-100", "eig-1000", "eig-2000",
   "eig-5000" };

 String prefix = "matrice-";
 String postfix = ".dat";

 for (int i = 0; i < names.length; i++) {
  String nameFile = prefix + names[i] + postfix;
  String file = path + nameFile;
  Matrix A = readMatrix(file);

  String erroreRelativo = "n.a.";
  String errorePrimaComp = "n.a.";
  String settimoAutovalore = "n.a.";
  long inizioS = 0;
  long fineS = 0;
  long inizioE = 0;
  long fineE = 0;

  /* If the matrix isn't eig type than solve with lu decomposition */
  if (!nameFile.startsWith("matrice-eig")) {
   String fileNoti = file.replace("matrice", "terminenoto");
   Matrix b = readArray(fileNoti);
   int size = A.getArray().length;
   Matrix x_esatta = new Matrix(size, 1, 1.0);

   inizioS = new Date().getTime();
   Matrix x_calcolata = A.solve(b);
```

```
    fineS = new Date().getTime();
    Matrix diff = x_esatta.minus(x_calcolata);

    double erroreRelativoD = diff.normF() / x_esatta.normF();
    double errorePrimaCompD = Math.abs(x_calcolata.get(0, 0) - 1.0);

    erroreRelativo = String.format("%e", erroreRelativoD);
    errorePrimaComp = String.format("%e", errorePrimaCompD);
   }

   /* If the matrix isn't rand type than calculate eigenvalues */
   if (!nameFile.startsWith("matrice-rand")) {
    inizioE = new Date().getTime();
    EigenvalueDecomposition eg = new EigenvalueDecomposition(A);
    fineE = new Date().getTime();
    ArrayList<Double> eigenValues = getOrderedEigenValues(eg.getD());

    double settimoAutovaloreD = eigenValues.get(6);
    settimoAutovalore = String.format("%f", settimoAutovaloreD);
   }

   /* Print results */
   String printName = nameFile.replace("matrice-", "")
     .replace("-symm", "").replace(".dat", "");

   if (nameFile.startsWith("matrice-eig")) {
    System.out.printf("%12s\t%16s\t%18s\t%19s\t%12s\t\t%2dms%n",
      printName, erroreRelativo, errorePrimaComp,
      settimoAutovalore, "-", (fineE - inizioE));
   } else if (nameFile.startsWith("matrice-rand")) {
    System.out.printf("%12s\t%16s\t%18s\t%19s\t%2dms\t\t%12s%n",
      printName, erroreRelativo, errorePrimaComp,
      settimoAutovalore, (fineS - inizioS), "-");
   } else {
    System.out
      .printf("%12s\t%16s\t%18s\t%19s\t%2dms\t\t%2dms%n",
        printName, erroreRelativo, errorePrimaComp,
        settimoAutovalore, (fineS - inizioS),
        (fineE - inizioE));
   }
  }
 }
}
```

TEST	ERRORE RELATIVO	ERRORE PRIMA COMP	AUTOVALORE N 7	TIME TO SOLVE
easy-10	3.510833e-16	2.220446e-16	7.000000	0m
easy-100	2.853360e-15	1.110223e-15	7.000000	1m
easy-1000	3.174443e-14	3.264056e-14	7.000000	670m
rand-10	4.711062e-15	8.659740e-15	n.a.	0m
rand-100	1.001901e-13	8.237855e-14	n.a.	1m
rand-1000	2.547025e-12	4.767298e-13	n.a.	647m
rand-5000	9.787832e-12	1.521339e-11	n.a.	101418m
bad-10	3.118816e-07	2.176155e-07	6.000000	0m
bad-100	2.258293e-05	2.394252e-05	6.000000	1m
bad-500	4.622912e-05	4.654016e-05	6.000000	69m
bad-1000	2.279306e-04	2.257559e-04	6.000000	645m
verybad-10	4.993346e-04	4.179128e-04	6.000000	0m
verybad-100	3.283544e-03	3.125627e-03	6.000000	2m
verybad-500	1.065932e-02	1.067230e-02	6.000000	70m
verybad-1000	2.926093e-02	2.903832e-02	6.000000	644m
eig-10	n.a.	n.a.	1.212788	0m
eig-20	n.a.	n.a.	0.616452	0m
eig-30	n.a.	n.a.	0.386165	0m
eig-40	n.a.	n.a.	0.251158	0m
eig-50	n.a.	n.a.	0.183589	0m
eig-100	n.a.	n.a.	0.105820	0m
eig-1000	n.a.	n.a.	0.005791	0m
eig-2000	n.a.	n.a.	0.004094	0m
eig-5000	n.a.	n.a.	0.001635	0m

1.4 Risultati e conclusioni

2 Discrete Cosine Transform

2.1 Teoria

2.2 JTransform

2.3 Programma discrete-cosine-transform

Listing 2: discrete-cosine-transform

```
import java.util.Date;

import edu.emory.mathcs.jtransforms.dct.DoubleDCT_2D;

public class Dct {

 /*
  * Method that permit to print a Matrix
  * for debug purpose
  */
 public static void printMatrix(double[][] z) {
  int n = z.length;
  int m = z[0].length;

  System.out.print(”[”);
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < m; j++) {
    System.out.print(z[i][j]);
    if (j != m - 1)
     System.out.print(” ”);
   }
   if (i != n - 1)
    System.out.println();
  }
  System.out.println(”]”);
 }

 /*
  * Method that calculate dct2 in two dimensions directly
  * just as described here:
  * http://www.mathworks.it/help/toolbox/images/ref/dct2.html
  */
 public static double[][] dct2in2dimension(double[][] z, double offset)
   throws Exception {
  if (z.length == 0)
   throw new Exception(”z empty”);

  if (z[0].length == 0)
   throw new Exception(”z row empty”);

 int n = z.length;
 int m = z[0].length;

 double[][] c = new double[n][m];
 double[] alf1 = new double[n];
 double[] alf2 = new double[m];

 alf1[0] = 1. / Math.sqrt(n);
 for (int k = 1; k < n; k++) {
  alf1[k] = Math.sqrt(2. / n);
 }

 alf2[0] = 1. / Math.sqrt(m);
 for (int l = 1; l < m; l++) {
  alf2[l] = Math.sqrt(2. / m);
 }

 double sum;
 for (int k = 0; k < n; k++) {
  for (int l = 0; l < m; l++) {
   sum = 0;
   for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
     sum += (z[i][j] + offset)
       * Math.cos((Math.PI * (2 * i + 1) * k)
         / (2 * n))
       * Math.cos((Math.PI * (2 * j + 1) * l)
         / (2 * m));
    }
   }
   c[k][l] = alf1[k] * alf2[l] * sum;
   System.out.println(k + " " + l + ": " + sum + "*" + alf1[k]
     + "*" + alf2[l] + " -> " + c[k][l]);
  }
 }

 return c;
}

/*
 * Method that calculate idct2 in two dimensions directly
 * just as described here:
 * http://www.mathworks.it/help/toolbox/images/ref/idct2.html
 */
public static double[][] idct2in2dimension(double[][] z, double offset)
  throws Exception {
 if (z.length == 0)
  throw new Exception("z empty");

 if (z[0].length == 0)
  throw new Exception("z row empty");
  int n = z.length;
  int m = z[0].length;

  double[][] c = new double[n][m];
  double[] alf1 = new double[n];
  double[] alf2 = new double[m];

  alf1[0] = 1. / Math.sqrt(n);
  for (int k = 1; k < n; k++) {
   alf1[k] = Math.sqrt(2. / n);
  }

  alf2[0] = 1. / Math.sqrt(m);
  for (int l = 1; l < m; l++) {
   alf2[l] = Math.sqrt(2. / m);
  }

  for (int k = 0; k < n; k++) {
   for (int l = 0; l < m; l++) {
    c[k][l] = 0;
    for (int i = 0; i < n; i++) {
     for (int j = 0; j < m; j++) {
      c[k][l] += alf1[i]
        * alf2[j]
        * z[i][j]
        * Math.cos((Math.PI * (2 * k + 1) * i)
          / (2 * n))
        * Math.cos((Math.PI * (2 * l + 1) * j)
          / (2 * m));
     }
    }
    c[k][l] += offset;
    System.out.println(k + " " + l + ": " + c[k][l]);
   }
  }

  return c;
 }

 /*
  * Method that calculate dct2 in two dimensions, first
  * calculate dct in row and after calculate dct in column
  */
 public static double[][] dct2(double[][] z, double offset) throws Exception {
  if (z.length == 0)
   throw new Exception("z empty");

  if (z[0].length == 0)
   throw new Exception("z row empty");

  int n = z.length;
  int m = z[0].length;
  double[][] c = new double[n][m];
  double[][] c2 = new double[n][m];
  double alfa;
  double sum;

  for (int k = 0; k < n; k++) {
   for (int l = 0; l < m; l++) {
    sum = 0;
    for (int i = 0; i < n; i++) {
     sum += (z[i][l] + offset)
       * Math.cos((Math.PI * (2. * i + 1.) * k) / (2. * n));
    }
    alfa = k == 0 ? 1. / Math.sqrt(n) : Math.sqrt(2. / n);
    c[k][l] = alfa * sum;
   }
  }

  for (int l = 0; l < m; l++) {
   for (int k = 0; k < n; k++) {
    sum = 0;
    for (int j = 0; j < m; j++) {
     sum += c[k][j]
       * Math.cos((Math.PI * (2. * j + 1.) * l) / (2. * m));
    }
    alfa = l == 0 ? 1. / Math.sqrt(m) : Math.sqrt(2. / m);
    c2[k][l] = alfa * sum;
   }
  }

  return c2;
 }

 /*
  * Method that calculate idct2 in two dimensions, first
  * calculate idct in row and after calculate idct in column
  */
 public static double[][] idct2(double[][] z, double offset)
   throws Exception {
  if (z.length == 0)
   throw new Exception(”z empty”);

  if (z[0].length == 0)
   throw new Exception(”z row empty”);

  int n = z.length;
  int m = z[0].length;
  double[][] c = new double[n][m];
  double[][] c2 = new double[n][m];
  double alfa;

  for (int k = 0; k < n; k++) {
   for (int l = 0; l < m; l++) {
    c[k][l] = 0;
    for (int i = 0; i < n; i++) {
     alfa = i == 0 ? 1. / Math.sqrt(n) : Math.sqrt(2. / n);
     c[k][l] += alfa * z[i][l]
       * Math.cos((Math.PI * (2 * k + 1) * i) / (2 * n));
    }
   }
  }

  for (int l = 0; l < m; l++) {
   for (int k = 0; k < n; k++) {
    c2[k][l] = 0;
    for (int j = 0; j < m; j++) {
     alfa = j == 0 ? 1. / Math.sqrt(m) : Math.sqrt(2. / m);
     c2[k][l] += alfa * c[k][j]
       * Math.cos((Math.PI * (2 * l + 1) * j) / (2 * m));
    }
    c2[k][l] += offset;
   }
  }

  return c2;
 }

 /*
  * test from example 1
  *
  * %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% esempio 1
  *
  * z = [1 2 3
  *      4 5 6];
  *
  * dct2(z) = [+8.5732   -2.0000    0.0000
  *            -3.6742         0         0];
  */
 public static void test1() throws Exception {
  double[][] vals = { { 1., 2., 3. }, { 4., 5., 6. } };
  double offset = 0;
  System.out.println("vals: ");
  printMatrix(vals);

  long startS = new Date().getTime();
  double[][] result = dct2(vals, offset);
  long endS = new Date().getTime();

  System.out.println("dct2 result: ");
  printMatrix(result);

  System.out.println("time: " + (endS - startS));

  double[][] ivals = idct2(result, -offset);
  System.out.println("idct2 result: ");
  printMatrix(ivals);

  System.out.println(”jtransform dct2 result: ”);
  int n = vals.length;
  int m = vals[0].length;
  for (int k = 0; k < n; k++) {
   for (int l = 0; l < m; l++) {
    vals[k][l] += offset;
   }
  }

  long startO = new Date().getTime();
  DoubleDCT_2D dct_2d = new DoubleDCT_2D(n, m);
  dct_2d.forward(vals, true);
  long endO = new Date().getTime();
  printMatrix(vals);

  System.out.println(”time: ” + (endO - startO));
 }

 /*
  * test from example 2
  *
  * %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% esempio 2
  * % -> tratto dall'articolo di Wallace
  * %
  * % attenzione: prima di calcolare la DCT2 tutti
  * % i coefficienti sono stati abbassati di 128
  * % (come prescrive lo standard) per equilibrare
  * % la frequenza (0,0)
  *
  * z = [139 144 149 153 155 155 155 155144 151 153 156 159 156 156 156
  *      150 155 160 163 158 156 156 156159 161 162 160 160 159 159 159
  *      159 160 161 162 162 155 155 155161 161 161 161 160 157 157 157
  *      162 162 161 163 162 157 157 157162 162 161 161 163 158 158 158];
  *
  * dct2(z-128) =
  *
  *   235.6250   -1.0333  -12.0809   -5.2029    2.1250   -1.6724   -2.7080    1.3238
  *   -22.5904  -17.4842   -6.2405   -3.1574   -2.8557   -0.0695    0.4342   -1.1856
  *   -10.9493   -9.2624   -1.5758    1.5301    0.2029   -0.9419   -0.5669   -0.0629
  *    -7.0816   -1.9072    0.2248    1.4539    0.8963   -0.0799   -0.0423    0.3315
  *    -0.6250   -0.8381    1.4699    1.5563   -0.1250   -0.6610    0.6088    1.2752
  *     1.7541   -0.2029    1.6205   -0.3424   -0.7755    1.4759    1.0410   -0.9930
  *    -1.2825   -0.3600   -0.3169   -1.4601   -0.4900    1.7348    1.0758   -0.7613
  *    -2.5999    1.5519   -3.7628   -1.8448    1.8716    1.2139   -0.5679   -0.4456
  *
  */
 public static void test2() throws Exception {
  double[][] vals = { { 139., 144., 149., 153., 155., 155., 155., 155. },
    { 144., 151., 153., 156., 159., 156., 156., 156. },
    { 150., 155., 160., 163., 158., 156., 156., 156. },
    { 159., 161., 162., 160., 160., 159., 159., 159. },
    { 159., 160., 161., 162., 162., 155., 155., 155. },
    { 161., 161., 161., 161., 160., 157., 157., 157. },
    { 162., 162., 161., 163., 162., 157., 157., 157. },
    { 162., 162., 161., 161., 163., 158., 158., 158. } };

  double offset = −128;
  System.out.println(”vals: ”);
  printMatrix(vals);

  long startS = new Date().getTime();
  double[][] result = dct2(vals, offset);
  long endS = new Date().getTime();

  System.out.println(”dct2 result: ”);
  printMatrix(result);

  System.out.println(”time: ” + (endS − startS));

  double[][] ivals = idct2(result, −offset);
  System.out.println(”idct2 result: ”);
  printMatrix(ivals);

  System.out.println(”jtransform dct2 result: ”);
  int n = vals.length;
  int m = vals[0].length;
  for (int k = 0; k < n; k++) {
   for (int l = 0; l < m; l++) {
    vals[k][l] += offset;
   }
  }

  long startO = new Date().getTime();
  DoubleDCT_2D dct_2d = new DoubleDCT_2D(n, m);
  dct_2d.forward(vals, true);
  long endO = new Date().getTime();
  printMatrix(vals);

  System.out.println(”time: ” + (endO − startO));
 }

 /*
  * test from example 3
  *
  * %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% esempio 3
  *
  * z =  [3      7     −5
  *       8     −9      7];
  *
  * dct2(z) =
  *
  *      4.4907    4.5000    4.9075
```

```
  *      -0.4082      3.5000   -14.1451
  *
  */
 public static void test3() throws Exception {
  double[][] vals = { { 3., 7., -5. }, { 8., -9., 7. } };
  double offset = 0;
  System.out.println("vals: ");
  printMatrix(vals);

  long startS = new Date().getTime();
  double[][] result = dct2(vals, offset);
  long endS = new Date().getTime();

  System.out.println("dct2 result: ");
  printMatrix(result);

  System.out.println("time: " + (endS - startS));

  double[][] ivals = idct2(result, -offset);
  System.out.println("idct2 result: ");
  printMatrix(ivals);

  System.out.println("jtransform dct2 result: ");
  int n = vals.length;
  int m = vals[0].length;
  for (int k = 0; k < n; k++) {
   for (int l = 0; l < m; l++) {
    vals[k][l] += offset;
   }
  }

  long startO = new Date().getTime();
  DoubleDCT_2D dct_2d = new DoubleDCT_2D(n, m);
  dct_2d.forward(vals, true);
  long endO = new Date().getTime();
  printMatrix(vals);

  System.out.println("time: " + (endO - startO));
 }

 public static void main(String[] args) throws Exception {
  System.out.println("TEST1");
  test1();
  System.out.println("\nTEST2");
  test2();
  System.out.println("\nTEST3");
  test3();
 }
}
```

2.4 Risultati e conclusioni