Relazione di "Metodi del Calcolo Scientifico"

Simon Vocella
Matricola: 718289

23 luglio 2012

Indice

1 Lu Decomposition

1.1 Teoria

Sia A una matrice invertibile. Allora A può essere decomposta come

$$\mathbf{PA} = \mathbf{LU}$$

dove P è una matrice di permutazione, L è una matrice triangolare inferiore a diagonale unitaria ($l_{ii} = 1, \forall i$) e U é una matrice triangolare superiore.

La decomposizione LU è simile all'algoritmo di Gauss. Nell'eliminazione gaussiana si prova a risolvere l'equazione matriciale

$$\mathbf{Ax} = \mathbf{b}$$

Il processo di eliminazione produce una matrice triangolare superiore U e trasforma il vettore b in b'

$$\mathbf{Ux} = \mathbf{b}'$$

Poichè U è una matrice triangolare superiore, questo sistema di equazioni si può risolvere facilmente tramite sostituzione all'indietro. Durante la decomposizione LU, però, b non è trasformata e l'equazione può essere scritta come

$$\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$$

così possiamo riusare la decomposizione se vogliamo risolvere lo stesso sistema per una differente b.

Nel caso più generale, nel quale la fattorizzazione della matrice comprende anche l'utilizzo di scambi di riga nella matrice, viene introdotta anche una matrice di permutazione P, ed il sistema diventa:

$$\begin{cases} Ly = Pb \\ Ux = y \end{cases}$$

La risoluzione di questo sistema permette la determinazione del vettore x cercato.

1.2 Jama

JAMA è un pacchetto di algebra lineare di base scritto in Java. Fornisce le classi a livello di utente per la costruzione e la manipolazione di matrici reali, dense. È pensato per fornire funzionalità sufficienti per problemi di routine.

Sono disponibili cinque scomposizioni fondamentali della matrice JAMA:

- La decomposizione di Cholesky, matrici simmetriche definite positive
- Decomposizione LU (eliminazione gaussiana) di matrici rettangolari (Quella che usiamo in questo caso)
- Decomposizione QR di matrici rettangolari
- Decomposizione autovalore di matrici quadrate e simmetriche sia nonsymmetric
- Valore Singolare decomposizione di matrici rettangolari

Funzionalità non coperte: JAMA non è affatto un' ambiente completo algebra lineare. Per esempio, non supporta matrici con struttura particolare o per ulteriori decomposizioni specializzati (ad esempio Shur, autovalore generalizzato). Matrici complesse non sono incluse.

1.3 Programma lu-decomposition

Per il progetto ho deciso di utilizzare Jama, perchè sembra essere veloce e semplice da implementare. Non supporta matrici i valori complessi e per questo non ho potuto calcolare gli autovalori delle matrici rand, che ho lasciato ad n.a.
Qui è listato il programma principale.

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));
    }
   }
  }
}
```

1.4 Risultati e conclusioni

Qui presentiamo i valori dei test precedenti con cui dovremmo confrontare i valori ottenuti.
Quando viene calcolato l'errore relativo e l'errore sulla prima componente, manteniamo lo stesso ordine di grandezza nella maggior parte di casi e, a volte, perdiamo o guadagniamo un ordine di grandezza nel caso di matrici bad e very bad.
Nel caso del calcolo del settimo autovalore i risultati sono praticamente identici.
Nella tabella dei valori ottenuti abbiamo aggiunto due colonne: Time to solve che è il tempo di calcolo della lu-decomposition e il tempo di calcolo degli autovalori.
Entrambi i tempi si matengono bassi tranne nel caso di rand-5000 per il tempo di calcolo della lu-decomposition e eig-5000 nel caso del calcolo dei suoi autovalori.

Tabella 1: Valori dei test precedenti

Test	Errore Relativo	Errore Prima Comp	Autovalore n.7
easy-10	4.242156e-016	6.661338e-016	7.000000
easy-100	2.019904e-015	3.330669e-015	7.000000
easy-1000	2.654198e-014	1.554312e-014	7.000000
rand-10	5.607561e-015	1.132427e-014	n.a.
rand-100	8.760746e-014	2.065015e-014	n.a.
rand-1000	3.827585e-012	5.317968e-013	n.a.
rand-5000	1.102259e-011	1.227463e-012	n.a.
bad-10	6.287577e-007	4.387160e-007	5.999999
bad-100	9.304715e-006	9.864899e-006	6.000000
bad-500	8.763526e-005	8.822490e-005	5.999999
bad-1000	7.556303e-005	7.484208e-005	5.999999
verybad-10	5.264102e-005	4.405734e-005	6.000000
verybad-100	1.197916e-003	1.140304e-003	6.000000
verybad-500	2.451563e-003	2.454550e-003	6.000000
verybad-1000	1.921933e-002	1.907311e-002	6.000000
eig-10	n.a.	n.a.	1.212788
eig-20	n.a.	n.a.	0.616452
eig-30	n.a.	n.a.	0.386165
eig-40	n.a.	n.a.	0.251158
eig-50	n.a.	n.a.	0.183589
eig-100	n.a.	n.a.	0.105820
eig-1000	n.a.	n.a.	0.005791
eig-2000	n.a.	n.a.	0.004094
eig-5000	n.a.	n.a.	0.001635

Tabella 2: Valori Ottenuti

Test	Errore Relativo	Errore Prima Comp	Autovalore n.7	Time to solve	Time to calc. eigen
easy-10	3.510833e-16	2.220446e-16	7.000000	0ms	0ms
easy-100	2.853360e-15	1.110223e-15	7.000000	1ms	0ms
easy-1000	3.174443e-14	3.264056e-14	7.000000	670ms	9ms
rand-10	4.711062e-15	8.659740e-15	n.a.	0ms	0ms
rand-100	1.001901e-13	8.237855e-14	n.a.	1ms	0ms
rand-1000	2.547025e-12	4.767298e-13	n.a.	647ms	0ms
rand-5000	9.787832e-12	1.521339e-11	n.a.	101418ms	0ms
bad-10	3.118816e-07	2.176155e-07	6.000000	0ms	0ms
bad-100	2.258293e-05	2.394252e-05	6.000000	1ms	0ms
bad-500	4.622912e-05	4.654016e-05	6.000000	69ms	2ms
bad-1000	2.279306e-04	2.257559e-04	6.000000	645ms	13ms
verybad-10	4.993346e-04	4.179128e-04	6.000000	0ms	0ms
verybad-100	3.283544e-03	3.125627e-03	6.000000	2ms	0ms
verybad-500	1.065932e-02	1.067230e-02	6.000000	70ms	1ms
verybad-1000	2.926093e-02	2.903832e-02	6.000000	644ms	8ms
eig-10	n.a.	n.a.	1.212788	0ms	0ms
eig-20	n.a.	n.a.	0.616452	0ms	0ms
eig-30	n.a.	n.a.	0.386165	0ms	0ms
eig-40	n.a.	n.a.	0.251158	0ms	0ms
eig-50	n.a.	n.a.	0.183589	0ms	0ms
eig-100	n.a.	n.a.	0.105820	0ms	1ms
eig-1000	n.a.	n.a.	0.005791	0ms	7ms
eig-2000	n.a.	n.a.	0.004094	0ms	30ms
eig-5000	n.a.	n.a.	0.001635	0ms	967ms

2 Discrete Cosine Transform

2.1 Teoria

Consideriamo N vettori di $\mathbb{R}^N$ definiti da

$$(w_k)_i = cos(k\pi x_i)$$

dove $x_i = \frac{i}{N} + \frac{1}{2}(\frac{1}{N}) = \frac{2i+1}{2N}$, i = 0..$N-1$.
Definiamo

$$a_k = \frac{1}{||w_k||} = \begin{cases} \sqrt{\frac{1}{N}} & k = 0 \\ \sqrt{\frac{2}{N}} & k >= 1 \end{cases}$$

in modo che

$$(\tilde{w_k})_i = a_k w_k$$

sia una base ortonormale.

La DCT (Discrete Cosine Transform) consiste nel calcolare i coefficenti di un vettore $y = (y_0, ..., y_N - 1)$ nella base $\{\tilde{w_k}\}$.

2.2 Caso bidimensionale

Si opera con i cosiddetti prodotti tensiorale.
Definiamo:

$$x_i = \frac{i}{N} + \frac{1}{2}(\frac{1}{N}) = \frac{2i+1}{2N}, \text{ i} = 0..N-1.$$
$$y_j = \frac{j}{M} + \frac{1}{2}(\frac{1}{M}) = \frac{2j+1}{2M}, \text{ j} = 0..M-1.$$

Definiamo gli e_{ij} è una matrice con 1 in posizione (ij) e zero altrove. In altre parole gli NM $\{e_{ij}\}$ sono ortonormali. Definiamo poi:

$$(w_{kl})_{ij} = cos(k\pi x_i)cos(l\pi y_j)$$

ed anche i NM $\{w_{kl}\}$ sono ortogonali. Se definiamo quindi $\tilde{w_{kl}} = a_{kl}w_{kl}$ abbiamo una base ortonormale. Da questo troviamo il calcolo della dct2:

$$c_{kl} = {a_{kl}}^2 \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} z_{ij}cos(k\pi x_i)cos(l\pi y_j)$$

Possiamo scrivere la formula come

$$c_{kl} = a_{kl} \sum_{i=0}^{N-1} [a_{kl} \sum_{j=0}^{M-1} z_{ij}cos(l\pi y_j)]cos(k\pi x_i)$$

Nella parentesi quadra ci sono N DCT monodimensionale fatte su un campione lungo M e poi restano M DCT monodimensionale su un campione lungo N. Quindi:

$$N \cdot DCT(M) + M \cdot DCT(N) \sim 2NM$$

2.3 JTransforms

JTransforms è la prima, open source, libreria multithreaded FFT scritta in puro Java. Attualmente, sono disponibili quattro tipi di trasformazioni: Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Discrete Sine Transform (DST) e Discrete Hartley Transform (DHT).

Caratteristiche:

- L'implementazione più veloce di DFT, DCT, DST e DHT in puro Java.
- Trasformazioni a 1, 2 e 3 dimensioni
- Dimensione arbitraria dei dati.
- Singola e doppia precisione.
- Varianti Uni-dimensionale e multi-dimensionale di trasformazioni 2D e 3D.
- Multithreading automatico - i thread sono utilizzati automaticamente se il numero di CPU > 1.
- FFT ottimizzate per i dati di input reali (40% più veloce che con i dati complessi).

Limitazioni:

- Il numero di thread deve essere una potenza di due.
- Sono disponibili solo in-place trasformazioni.
- Solo DCT-II-III e DCT sono disponibili.
- Solo DST-II-III e DST sono disponibili.
- Solo DHT reale è disponibile.

2.4 Programma discrete-cosine-transform

Per il progetto ho deciso di utilizzare JTransforms come implementazione FAST.
Qui è listata la classe che gestisce la mia implementazione di dct sia unidemensionale che bidimensionale. Nel caso bidimensionale ho implementato sia quella diretta a due dimensioni, che quella fatta prima su righe e poi su colonne. I benchmark sono stati eseguiti solo sulla seconda implementazione.

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;
 }

 public static double[][] filter(double[][] z, double threshold)
         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;

  int i=((int) Math.rint(n*threshold));
  int j=((int) Math.rint(m*threshold));

  System.out.println(i+" "+n+" <-> "+j+" "+m);

  for(; i<n; i++) {
        for(; j<m; j++) {
                z[i][j] = 0.0;
        }
  }

  return z;
 }

 /*
  * 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));

  result = filter(result, 1.0);

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

  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));

 result = filter(result, 0.25);

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

```
 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));

 result = filter(result, 0.25);
```

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

  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();
 }
}
```

2.5 Risultati e conclusioni

Nella tabella riportata qui sotto ho messo i risultati dei benchmark sia della mia implementazione che di Jtransforms. Notiamo la grande differenza nei tempi di computazione dovuta, sia al fatto che Jtransforms utilizza multithread, sia anche per l'implementazione FAST.

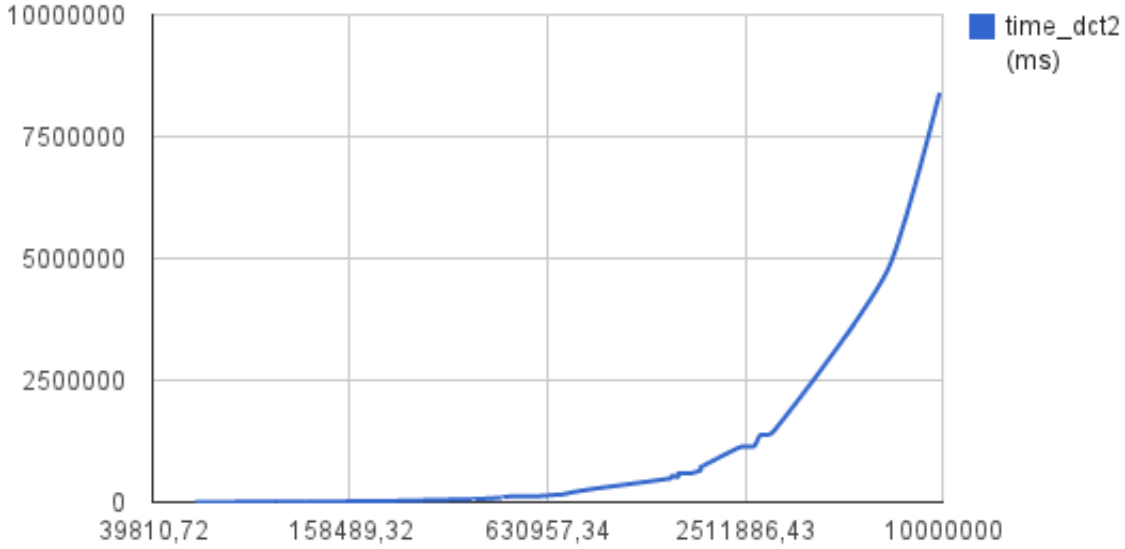


Figura 1: Grafico dct2 implementata

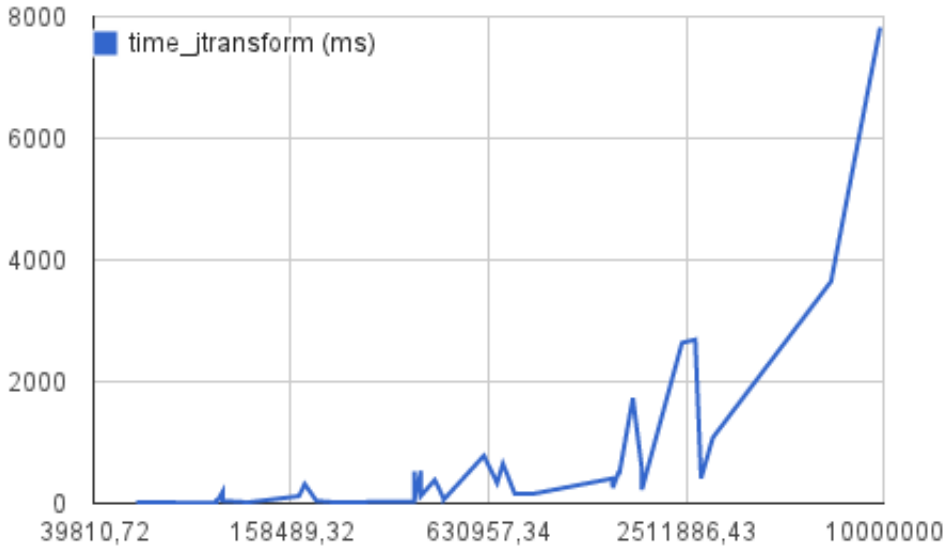


Figura 2: Grafico dct2 FAST

Name	Time Dct2 (ms)	Time Jt. (ms)	width (px)	height (px)
scaled/scaled/scaled/artificial.bmp	7942	193	384	256
scaled/scaled/scaled/big_building.bmp	127478	783	902	677
scaled/scaled/scaled/big_tree.bmp	81907	383	761	569
scaled/scaled/scaled/bridge.bmp	19125	316	344	507
scaled/scaled/scaled/cathedral.bmp	7479	79	250	376
scaled/scaled/scaled/deer.bmp	18021	118	506	331
scaled/scaled/scaled/fireworks.bmp	10058	19	392	294
scaled/scaled/scaled/flower_foveon.bmp	3284	18	284	189
scaled/scaled/scaled/hdr.bmp	7875	40	384	256
scaled/scaled/scaled/leaves_iso_200.bmp	7438	10	376	250
scaled/scaled/scaled/leaves_iso_1600.bmp	7551	10	376	250
scaled/scaled/scaled/nightshot_iso_100.bmp	10080	33	392	294
scaled/scaled/scaled/nightshot_iso_1600.bmp	10245	10	392	294
scaled/scaled/scaled/spider_web.bmp	22486	38	532	356
scaled/scaled/scaled/zone_plate.bmp	7588	8	375	250
scaled/scaled/artificial.bmp	66245	534	768	512
scaled/scaled/big_building.bmp	1135163	2639	1804	1354
scaled/scaled/big_tree.bmp	597656	1731	1522	1138
scaled/scaled/bridge.bmp	157433	650	688	1014
scaled/scaled/cathedral.bmp	60869	118	500	752
scaled/scaled/deer.bmp	151674	335	1012	662
scaled/scaled/fireworks.bmp	82743	82	784	588
scaled/scaled/flower_foveon.bmp	25788	17	568	378
scaled/scaled/hdr.bmp	63813	125	768	512
scaled/scaled/leaves_iso_200.bmp	60658	527	752	500
scaled/scaled/leaves_iso_1600.bmp	61871	68	752	500
scaled/scaled/nightshot_iso_100.bmp	82330	111	784	588
scaled/scaled/nightshot_iso_1600.bmp	111011	59	784	588
scaled/scaled/spider_web.bmp	207638	161	1064	712
scaled/scaled/zone_plate.bmp	60285	29	750	500
scaled/artificial.bmp	518198	535	1536	1024
scaled/big_building.bmp	8364020	7823	3608	2708
scaled/big_tree.bmp	4872149	3647	3044	2276
scaled/bridge.bmp	1369571	411	1376	2028
scaled/cathedral.bmp	540372	267	1000	1504
scaled/deer.bmp	1151979	2690	2024	1324
scaled/fireworks.bmp	658911	568	1568	1176
scaled/flower_foveon.bmp	267050	155	1136	756
scaled/hdr.bmp	585721	469	1536	1024
scaled/leaves_iso_200.bmp	525412	335	1504	1000
scaled/leaves_iso_1600.bmp	540518	256	1504	1000
scaled/nightshot_iso_100.bmp	727666	227	1568	1176
scaled/nightshot_iso_1600.bmp	720260	225	1568	1176
scaled/spider_web.bmp	1416648	1079	2128	1424
scaled/zone_plate.bmp	491263	408	1500	1000

2.6 Filtro delle frequenze

La DCT è perfettamente reversibile e non perdiamo definizione dell'immagine.
Nel caso in cui vogliamo tagliare un certo numero di frequenze definiamo DCT(M%), dove che sia per altezza che larghezza prendiamo $\frac{M}{100} \cdot N$ valori, dove N sono i valori totali.
Nelle figure sottostanti vediamo nelle prime quattro, il caso in cui tagliamo le frequenze con la mia implementazione e nelle quattro foto successive il filtraggio con l'implementazione FAST (Jtransforms).
Come vediamo JTransforms non è infallibile, in alcuni casi, se si tagliano certe frequenze, non riesce più a ricostruire l'immagine, questo è causato dal fatto che sono più thread che cercano di collaborare, si ricade su un problema di threads ed è un baco noto.

(a) Immagine originale

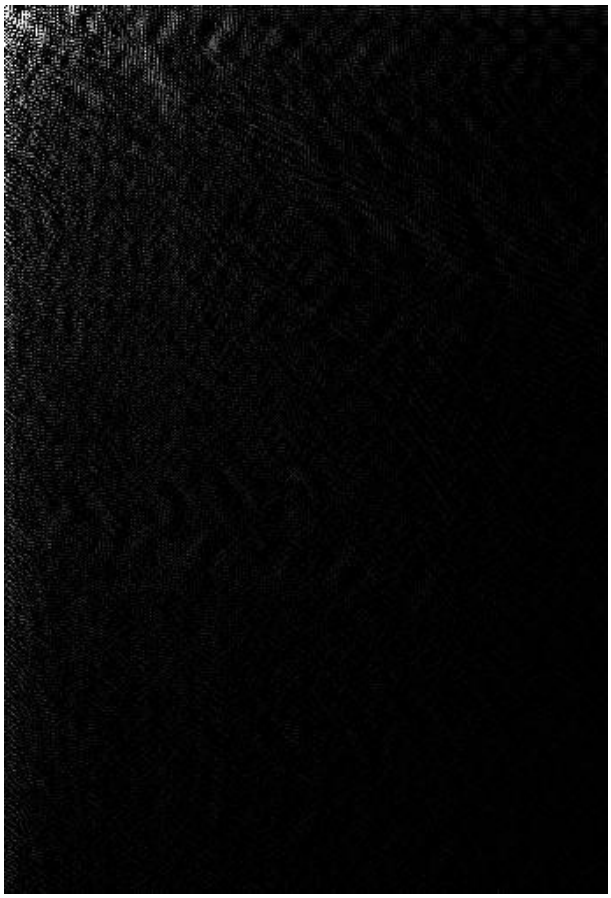
(b) DCT(50%)

(c) DCT(10%)

(d) DCT(1%)

(e) Immagine originale

(f) FAST DCT(50%)



(g) FAST DCT(10%)

(h) FAST DCT(1%)