AA 2015-2016
Metodi del Calcolo Scientifico
Algebra lineare numerica -
Metodi diretti per matrici sparse

743464 - Banfi Alessandro
735722 - Curatitoli Mattia
742752 - Ghelfi Camillo

Progetto 1

Lo scopo di questo progetto é studiare l'implementazione di algoritmi di risoluzione diretta di sistemi lineari per matrici sparse in ambienti di programmazione standard.
I linguaggi da noi scelti sono: MATLAB, Python e C.

MATLAB
Come mostrato nel codice allegato in MATLAB é molto semplice soddisfare le richieste grazie alle funzionalità proposte da MATLAB stesso.
Il programma scritto permette di indicare la cartella da analizzare, dalla quale vengono prese le matrici (in formato .mat) e per ognuna (A):

- viene calcolato il vettore colonna unitario (xe) con numero di righe pari a quelle della matrice sparsa considerata
- viene calcolato il vettore dei termini noti b = A*xe
- viene risolto il sistema lineare Ax = b
- vengono calcolati il tempo di risoluzione, errore relativo e memoria allocata

Per la risoluzione del sistema lineare si utilizza il comando "\" o "mldivide" già presente in MATLAB che utilizza la libreria UMFPACK.
Per il calcolo della memoria utilizzata abbiamo utilizzato il "profiler" messo a disposizione da MATLAB che permette di tenere traccia del tempo di esecuzione ma soprattutto della memoria allocata da ogni funzione chiamata nel programma eseguito.
"profile viewer" nella riga 52 permette di mostrare la schermata del profiler con i dettagli di ogni funzione utilizzata.
"profile.FunctionTable.TotalMemAllocated" invece ritorna esattamente la memoria allocata dalla funzione profilata.

Python

La seconda scelta è ricaduta su Python, un linguaggio di programmazione ad alto livello, interpretato e molto diffuso, comodo per le numerose librerie disponibili.

1.1 - os

Libreria che consente di interfacciarsi al sistema operativo utilizzato, in modo da leggere e scrivere file, e manipolare i loro path.

1.2 - datetime

Libreria che permette di manipolare la data, utilizzata per calcolare i tempi di calcolo.

1.3 - scipy

Libreria open-source nata nel 2001 dalla cooperazione tra Travis Oliphant, Eric Jones e Pearu Peterson, utilizzata per l'elaborazione e analisi di dati in ambito matematico e molto aggiornata (SciPy 0.17.1 released 12-05-2016), scaricabile e completa di documentazione al sito www.scipy.org .
Negli anni si è espansa comprendendo moduli come NumPy (utilizzata in generale per la computazione scientifica in Python) e sottomoduli per l'elaborazione di matrici, algebra lineare, trasformata FFT e image processing.
scipy.io permette la manipolazione in lettura e scrittura di file anche in formato matriciale Matrix Market (.mtx).
scipy.sparse invece è il modulo dedicato alla manipolazione di matrici sparse e algebra lineare; in particolare il modulo *csc_matrix* è utilizzato per effettuare operazione algebriche di addizione, sottrazione, moltiplicazione e divisione tra matrici csc, ovvero Compressed Sparse Column (analogamente al modulo *csr_matrix* per le Compressed Sparse Row);
il modulo *sparse.linalg* per l'algebra lineare comprende *spsolve(A,b,use_umfpack=bool)* per la risoluzione di sistemi lineari tra matrici in forma Ax=b che prende in input la matrice quadrata A, la matrice o vettore b e permette l'utilizzo di UMFPACK.
UMFPACK utilizza BLAS (Basic Linear Algebra Subprograms, la specifica di alcune operazioni fondamentali tra matrici e vettori) fornita nel package *scikit-umfpack (0.2.1)* (released: 12-10-2015) disponibile all'indirizzo https://pypi.python.org/pypi/scikit-umfpack e installabile in ambiente Unix semplicemente eseguendo da terminale `python setup.py install` nella cartella scaricata e decompressa.

1.4 - memory_profiler e psutil

memory_profiler (0.41) è una libreria per il monitoraggio dell'occupazione di memoria dei processi Python in grado di analizzare anche riga per riga il codice. É un modulo in puro python scaricabile all' indirizzo https://pypi.python.org/pypi/memory_profiler e installabile in ambiente Unix eseguendo da terminale `python setup.py install` nella cartella scaricata e decompressa (released: 07-01-2016).
Come opzione si può installare *psutil (4.2.0)* (released: 15-05-2016), un modulo cross-platform per l'analisi dei processi e il loro utilizzo delle risorse di sistema (CPU, memory, network) in Python, ben documentato all'indirizzo http://pythonhosted.org/psutil/ e scaricabile all'inidirizzo https://pypi.python.org/pypi/psutil .

ANSI C

1.1 - INTRODUZIONE GENERALE

SuperLU (Supernodal LU) è una libreria generica per la risoluzione diretta di sistemi lineari di matrici sparse aventi grandi dimensioni, anche non simmetriche, concepita per sistemi di calcolo ad alte prestazioni. La libreria è scritta in ANSI C, ed è utilizzabile sia da programmi scritti in C sia scritti in linguaggio FORTRAN. La libreria compie la fattorizzazione LU della matrice A dei coefficienti, con pivoting parziale e in seguito applica l'algoritmo di Gauss per la risoluzione del sistema. Le colonne della matrice possono essere preordinate prima della fattorizzazione, sia utilizzando routine fornite dall'utente sia quelle fornite con la libreria stessa. Il progetto è stato fondato dagli enti statunitensi DOE, NSF e DARPA e sviluppato all'interno dell'università di Berkeley. Lo sviluppo è attualmente attivo (ultimo rilascio 22/05/2016) e la libreria risulta essere ben documentata da un guida liberamente fruibile in formato PDF.
Esistono tre varianti della libreria:

- *SuperLU sequenziale* (quella provata). Particolarmente indicata per macchine di calcolo sequenziali.
- *SuperLU _MT*. Utilizzabile su macchine parallele a memoria condivisa.
- *SuperLU_DIST*. Utilizzabile in sistemi paralleli distribuiti.

1.2 - BLAS

La libreria SuperLU (come molti altri software matematici, tra cui MATLAB) dipende da un'implementazione dell'interfaccia BLAS per l'esecuzione di operazioni fondamentali tra matrici e vettori, quali ad esempio moltiplicazione matrice/matrice e moltiplicazione vettore/matrice. *BLAS (Basic Linear Algebra Subprograms)* è un'interfaccia di programmazione (API), utilizzabile con i linguaggi C e FORTRAN, che fornisce un insieme di routine per l'esecuzione di funzioni basilari su matrici e vettori. L'interfaccia si divide in tre livelli distinti:

1. Operazioni con scalari, vettori e vettori-vettori.
2. Operazioni matrici-vettori.
3. Operazioni matrici-matrici.

BLAS è stata pensata per garantire la massima portabilità tra sistemi operativi e architetture delle applicazioni matematiche che ne fanno uso.
Esistono differenti implementazioni di BLAS, alcune freeware ed altre commerciali, tra cui:

- NetLib CBALS (implementazione di riferimento per il linguaggio C)
- OpenBLAS
- ATLAS
- Intel MKL

SuperLU è fornito con una propria implementazione delle routine BLAS, che tuttavia, gli autori stessi della libreria, sconsigliano di utilizzare in ambiente di produzione, a causa delle basse performance. Nella prova è stata utilizzata tale libreria.

1.3 - TEST di SuperLU

Per eseguire il test della libreria sono stati svolti i seguenti passaggi:

1. Download dei sorgenti della versione sequenziale di SuperLU dal sito web del progetto (http://crd-legacy.lbl.gov/~xiaoye/SuperLU/).
2. Download di Intel MKL dal sito web di Intel (previa registrazione).
3. Compilazione della libreria SuperLU e link con le librerie statiche di Intel MKL.
4. Scrittura di una libreria wrapper ANSI C utilizzata per agevolare l'esecuzione del test (`libslu.h` e `libslu.c`).
5. Compilazione della libreria wrapper e link con SuperLU.
6. Esecuzione del test.

I sorgenti sono stati compilati e testati sia in ambiente Windows (compilatore Microsoft C/C++ e IDE Visual Studio 2015), sia in ambiente Mac OSX (compilatore gcc e Eclispe CDT/XCode).

1.4 - FORMATO DELLE MATRICI

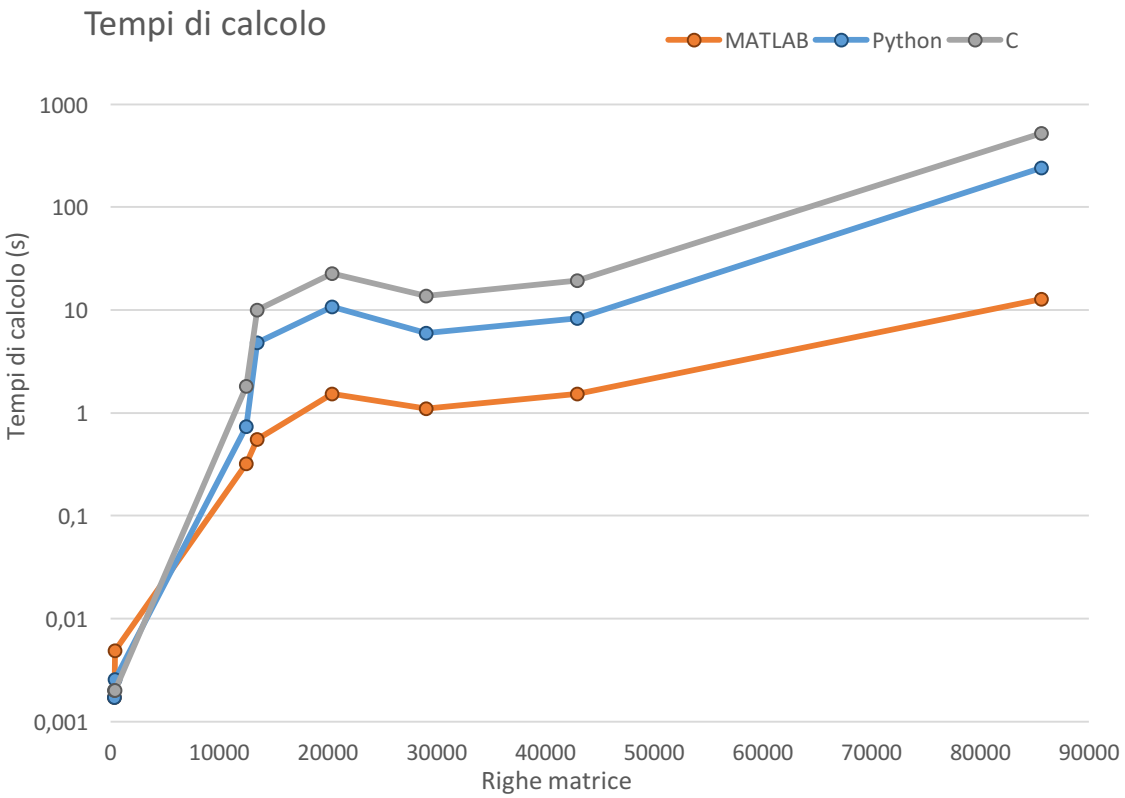
La libreria SuperLU utilizza il formato binario *CC (Compressed Columns)* per la rappresentazione delle matrici. Utilizzando tale formato è possibile rappresentare una matrice mediante tre array: uno per memorizzare i coefficienti diversi da zero della matrice, uno per memorizzare i rispettivi indici di riga e uno per memorizzare gli indici dei coefficienti situati all'inizio di ogni colonna. È stata scritta una routine di supporto (`libslu_read_mm_file`) per leggere un file in formato Matrix Market e popolare i tre array secondo la convezione CC.

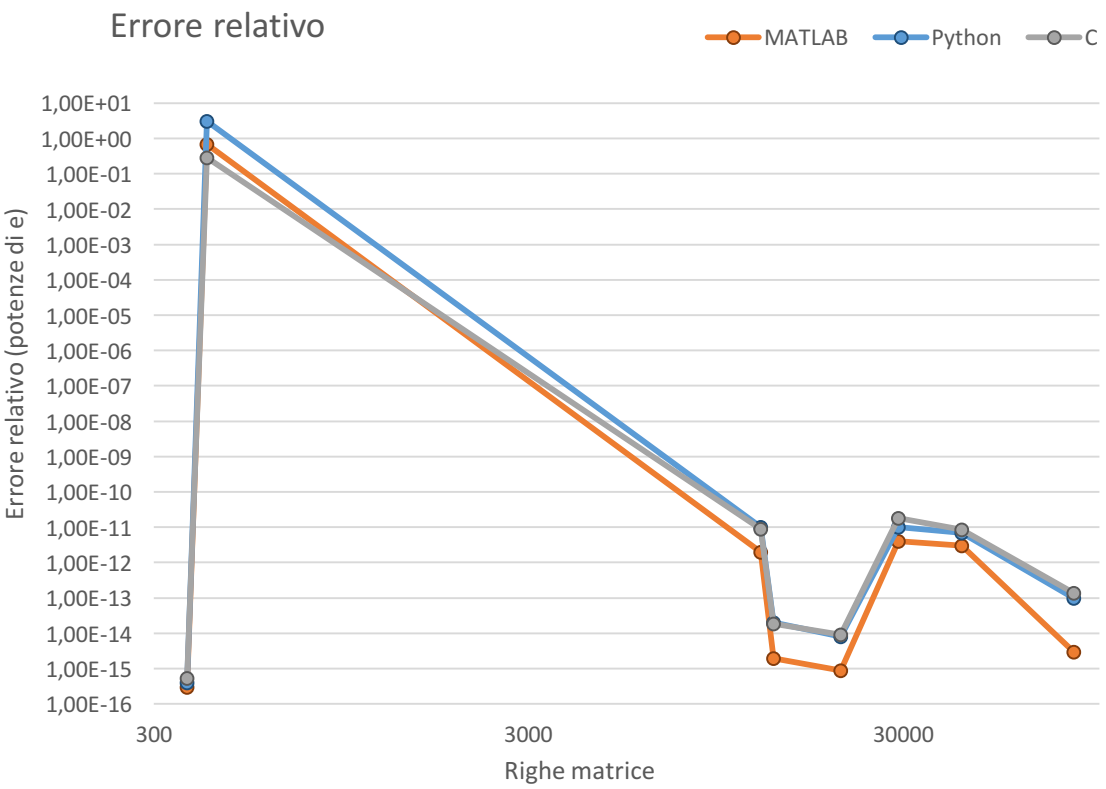
1.5 - CODICE SORGENTE

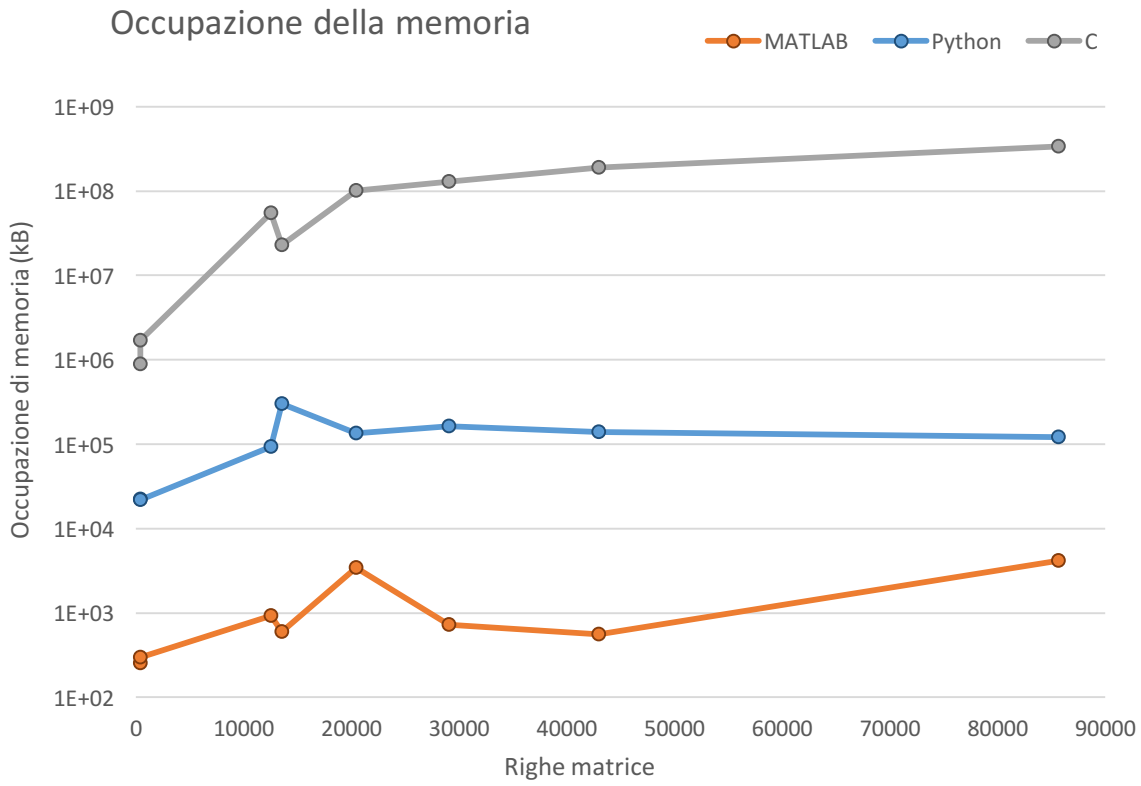
La funzione principale presente nel file libslu.c è `libslu_solvesparse`, essa si occupa dell'effettiva risoluzione del sistema lineare a matrice sparsa, utilizzando la libreria SuperLU. Alla riga 142 viene aperto il file in formato Matrix Market in cui è specificata la matrice dei coefficienti e successivamente (riga 154), mediante la funzione `libslu_read_mm_file` sono ricavati i tre array che costituiscono la rappresentazione CC (Compressed Columns) della suddetta matrice (`nzVal`, `rowind`, `colptr`). Mediante la lettura del file è anche ricavato il numero di elementi diversi da zero (`nonZero`) e il numero di righe e colonne (`m, n`). Alla riga 169 viene costruita la rappresentazione interna della libreria SuperLU della matrice dei coefficienti (struttura `SuperMatrix`). Dalla riga 173 alla riga 198 avviene la costruzione del vettore dei termini noti (`b`), ottenuta moltiplicando il vettore unitario `xe` con la matrice A. Per la moltiplicazione di un vettore con una matrice in formato CC, è stata scritta la funzione di utilità `libslu_matrix_vector` (definita alla riga 303). Alla riga 223 sono impostate le opzioni del solver SuperLU. L'unico valore impostato, differente rispetto a quelli di default, è la tipologia di permutazione da applicare alle colone di A (COLAMD). Alla riga 240 viene invocato il solver SuperLU, in particolare, tramite la funzione `dgssv`, viene utilizzato il solutore a coefficienti a virgola mobile in doppia precisione (tipo double). La soluzione del sistema è posta dal solutore all'interno del vettore dei termini noti (`b`). Dalla riga 292 è liberata la memoria in precedenza allocata.

Analisi e confronto

Per fare un paragone il più possibile omogeneo i programmi sono stati testati sulla stessa macchina, con processore Intel Core i7, 2.6 GHz e 8GB di RAM su sistema operativo OSX El Capitan, basato su Unix.




Errore relativo
MATLAB
Python
C
Errore relativo (potenze di e)
1,00E+01
1,00E+00
1,00E-01
1,00E-02
1,00E-03
1,00E-04
1,00E-05
1,00E-06
1,00E-07
1,00E-08
1,00E-09
1,00E-10
1,00E-11
1,00E-12
1,00E-13
1,00E-14
1,00E-15
1,00E-16
300
3000
30000
Righe matrice


Occupazione della memoria
MATLAB
Python
C
Occupazione di memoria (kB)
1E+09
1E+08
1E+07
1E+06
1E+05
1E+04
1E+03
1E+02
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
Righe matrice

MATLAB

```
% ***************************************
%   Metodi del Calcolo Scientifico 2016
% ---------------------------------------
%   743464 Banfi Alessandro
%   735722 Curatitoli Mattia
%   742752 Ghelfi Camillo
% ***************************************

clear all
clc

% ottengo la lista dei file .mat nella cartella
folderName = input('type folder name: ','s')
matfolder = what(folderName);
addpath(matfolder.path)
matlist = matfolder.mat;

for i = 1:(length(matlist))

    disp(['====================================']);

    % carico il file di dati ed ottengo il referimento alla matrice dei
    % coefficienti A
    mat = load(char(matlist(i)));
    A = mat.Problem.A;

    disp(['Name matrix: ', mat.Problem.name]);
    disp(['Rows: ', num2str(size(A,2))]);
    disp(['Columns: ', num2str(size(A,1))]);
    disp(['Nonzero: ', num2str(nnz(A))]);
    disp(['-  -  -  -  -  -  -  -  -  -']);

    % abilito il profiler all'analisi della memoria
    profile clear
    profile -memory -history on

    % ottengo il vettore colonna unitario con numero righe di A
    xe = ones(length(A),1);
    % ottengo il vettore dei termini noti b = A * xe
    b = A * xe;

    tic
    % risoluzione del sistema lineare (con UMFPACK)
    x = A\b;
    t = toc;
    disp(['Solving time: ',num2str(t),' s']);

    % calcolo errore relativo
    relErr = norm(x-xe)/norm(xe);
    disp(['Relative error: ',num2str(relErr)]);

    % genero il report dell'occupazione della memoria
    %profile viewer
    p = profile('info');
    mem = (p.FunctionTable(4).TotalMemAllocated)/1000;
    disp(['Allocated memory: ',num2str(mem),' kB']);

    % memorizzo i dati ottenuti in una lista
     listMatrix(i).name_matrix = mat.Problem.name;
     listMatrix(i).rows = size(A,2);
     listMatrix(i).solv_time = t;
     listMatrix(i).relative_error = relErr;
     listMatrix(i).alloc_memory = double(mem);

end

sortrows(struct2table(listMatrix),2)
```

Python

```
1.  # ***************************************
2.  #   Metodi del Calcolo Scientifico 2016
3.  # ---------------------------------------
4.  #   743464 Banfi Alessandro
5.  #   735722 Curatitoli Mattia
6.  #   742752 Ghelfi Camillo
7.  # ***************************************
8.
9.  # ************************************************
10. # Per una corretta esecuzione installare i pacchetti:
11. # - scikit-umfpack (0.2.1)
12. # - memory_profiler (0.41)
13. # - psutil (4.2.0)
14. # ************************************************
15.
16. ### package per interfacciarsi col sistema operativo
17. import os as os
18. ### package scipy con moduli per calcoli matematici
19. import scipy as scipy
20. ### modulo input/output files
21. import scipy.io as scipyio
22. ### modulo matrici Compressed Sparse Column
23. from scipy.sparse import csc_matrix
24. ### modulo per algebra lineare
25. from scipy.sparse import linalg
26. ### modulo per calcolo della norma in matrici sparse
27. from scipy.linalg import norm
28. ### modulo per il timestamp
29. from datetime import datetime
30. ### modulo profiler memoria
31. from memory_profiler import profile
32. import psutil
33.
34. ### file di log della memoria rilevata dal profiler
35. # mem_prof = open('memory_profiler.log','w+')
36. # @profile(stream=mem_prof)
37. @profile
38. def xSolver(A,b,bool):
39.     x = scipy.sparse.linalg.spsolve(A,b,use_umfpack=bool)
40.     return x
41.
42.
43. def matrixSolver(folder, eachmtx):
44.     print '====================================='
45.     ### ottengo il percorso di ogni file .mtx
46.     pathmtx = folder + '/' + eachmtx
47.     ### leggo le info del file .mtx con scipyio.mminfo()
48.     mminfo = scipyio.mminfo(pathmtx)
49.     ### scipyio.mminfo() = [rows, cols, entries, format, field, symmetry]
50.     ### -> mminfo[0] = numero righe della matrice
51.     rows = mminfo[0]
52.     ### carico file .mtx e converto in matrice csc (Compressed Sparse Column)
53.     A = scipyio.mmread(pathmtx).tocsc()
54.     ### ottengo elementi diversi da zero in A
55.     nonzero = A.nnz
56.
57.     print 'Name matrix: ' + pathmtx
58.     print 'Rows: ' + str(rows)
59.     print 'Columns: ' + str(rows)
60.     print 'Nonzero: ' + str (nonzero)
61.     print '-  -  -  -  -  -  -  -  -  -'
62.
63.     ### ottengo il vettore colonna unitario xe con numero righe di A
64.     xe = scipy.ones(rows)
65.     ### ottengo il vettore dei termini noti b = A * xe
66.     b = A * xe
67.
68.     ### inizializzo il vettore soluzione con il numero righe di A
```

```
69.     x = scipy.empty(rows)
70.     timerOn = datetime.now()
71.     ### risoluzione del sistema lineare senza UMFPACK
72.     x = xSolver(A,b,False)
73.     t = datetime.now() - timerOn
74.     print 'Solving time without UMFPACK: ' + str(t) + ' s'
75.
76.     ### reinizializzo il vettore soluzione con il numero righe di A
77.     x = scipy.empty(rows)
78.     timerOn = datetime.now()
79.     ### risoluzione del sistema lineare con UMFPACK
80.     x = xSolver(A,b,True)
81.     t = (datetime.now() - timerOn)
82.     print 'Solving time with UMFPACK: ' + str(t) + ' s'
83.
84.     ### calcolo errore relativo
85.     relErr = norm(x-xe)/norm(xe)
86.     print 'Relative error: ' + str(relErr) + '\n'
87.
88.
89. def loopSparseMatrix():
90.     error = False
91.     while not error:
92.         folder = raw_input('type folder name: ')
93.         if os.path.isdir(folder):
94.             error = True
95.             path = os.path.abspath(folder)
96.             ### elenco tutti i file .mtx contenuti nella cartella 'folder'
97.             mtxlist = [each for each in os.listdir(path) if each.endswith('.mtx')]

98.             ### per ogni mtx nella lista eseguo la funzione matrixSolver
99.             for eachmtx in mtxlist:
100.                    matrixSolver(folder, eachmtx)
101.            else:
102.                print 'Error with folder name. retry'
103.
104.
105.    if __name__ == '__main__':
106.
107.        ### funzione per la soluzione di TUTTE le matrici
108.        loopSparseMatrix()
109.
110.        ### funzioni per la soluzione di CIASCUNA SINGOLA matrice
111.        # matrixSolver('FEMLAB', 'ns3Da.mtx')
112.        # matrixSolver('FEMLAB', 'poisson2D.mtx')
113.        # matrixSolver('FEMLAB', 'poisson3Da.mtx')
114.        # matrixSolver('FEMLAB', 'poisson3Db.mtx')
115.        # matrixSolver('FEMLAB', 'problem1.mtx')
116.        # matrixSolver('FEMLAB', 'sme3Da.mtx')
117.        # matrixSolver('FEMLAB', 'sme3Db.mtx')
118.        # matrixSolver('FEMLAB', 'sme3Dc.mtx')
```

ANSI C

libslu_main.c

```c
#include <stdio.h>
#include <string.h>
#include <stdio.h>
#include "libslu.h"
#include <dirent.h>
#include <unistd.h>
#define DEBUG 1
#define MAX_DIRPATH_LEN     200
#define MAX_FILENAME_LEN     30

int main(int argc, char* argv[])
{
    LIBSLU_SOLUTION* solution;
    int errCode = 0;
    char filePath[MAX_DIRPATH_LEN + MAX_FILENAME_LEN + 1];
    char dirPath[MAX_DIRPATH_LEN + 1];
    DIR* dir;
    struct dirent *nextFile;

    /* Valida numero di parametri in ingresso */
    if (argc != 2)
    {
        printf("\nUsage: %s <directory path>\r\n", argv[0]);
        return -1;
    }
    /* Verifica lunghezza path directory */
    if (strlen(argv[1]) > MAX_DIRPATH_LEN)
    {
        printf("\nDirectory path too long. Max allowed length is %d\r\n", MAX_DIRP
ATH_LEN);
    }
    /* Copia path directory e concatena "\*.mtx" */
    strcpy(dirPath, argv[1]);

    /* Cerca il primo file nella directory */
    dir = opendir(dirPath);

    if (dir == NULL)
    {
        printf("\nError in opendir\r\n");
        return -1;
    }

    printf("Start libslu_main\r\n");

    while ((nextFile = readdir(dir)) != NULL)
    {
        if (nextFile->d_type == DT_REG)
        {
            /* Ottiene il path assoluto del file */
            strcpy(filePath, dirPath);
            strcat(filePath, "/");
            strcat(filePath, nextFile->d_name);

            if (strstr(filePath, ".mtx") == NULL)
            {
                continue;
            }

            /* Risolve la matrice associata al file .mtx */
            solution = libslu_init(filePath, &errCode, DEBUG);
            if (errCode != 0)
            {
                return -1;
```

```
64.             }
65.
66.             errCode = libslu_solvesparse(solution, DEBUG);
67.             if (errCode != 0)
68.             {
69.                 return -1;
70.             }
71.
72.             libslu_print_solution(solution, DEBUG);
73.             libslu_free(solution, DEBUG);
74.         }
75.     }
76.
77.     closedir(dir);
78.
79.     return 0;
80.
81. }
```

libslu.h

```
1.  #ifndef LIBSLU_H
2.  #define LIBSLU_H
3.
4.  typedef unsigned int BOOL;
5.
6.  typedef struct
7.  {
8.      char* matrixFile;
9.      unsigned long rows;
10.     unsigned long cols;
11.     unsigned long nonZero;
12.     double* solution;
13.     unsigned int solutionTime;
14.     double relError;
15.     unsigned long memoryUsage;
16.
17. } LIBSLU_SOLUTION;
18.
19. typedef int LIBSLU_RET_CODE;
20.
21. #define LIBSLU_NO_ERR                               0
22. #define LIBSLU_ERR_PARAM                            1
23. #define LIBSLU_ERR_MALLOC                           2
24. #define LIBSLU_ERR_FOPEN                            3
25. #define LIBSLU_ERR_MM_HEADER                        4
26. #define LIBSLU_ERR_MM_BODY                          5
27. #define LIBSLU_ERR_MM_FILE                          6
28.
29. LIBSLU_SOLUTION* libslu_init(char* matrixFile, LIBSLU_RET_CODE* error, BOOL debug)
    ;
30. LIBSLU_RET_CODE  libslu_solvesparse(LIBSLU_SOLUTION* solutionHdlr, BOOL debug);
31. LIBSLU_RET_CODE  libslu_print_solution(LIBSLU_SOLUTION* solutionHdlr, BOOL debug);

32. LIBSLU_RET_CODE  libslu_free(LIBSLU_SOLUTION* solutionHdlr, BOOL debug);
33.
34. #endif /* LIBSLU_H */
```

libslu.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <slu_ddefs.h>

#ifdef WIN32
    #include <windows.h>
#elif __APPLE__
    #include <mach/mach.h>
    #include <mach/mach_time.h>
    #include <CoreServices/CoreServices.h>
#endif

#include "libslu.h"

/* #define LIBSLU_FILE_TEST */

double* libslu_matrix_vector(int m, int n, int ncc, int icc[], int ccc[], double acc[], double x[]);
LIBSLU_RET_CODE libslu_read_mm_file(FILE* fp, int* n, int* m, int* nonZero, double ** nzVal, int** rowind, int** colptr);
void libslu_print_vector_file(char* filePath, double* solution, int solutionLen);


LIBSLU_SOLUTION* libslu_init(char* matrixFile, LIBSLU_RET_CODE* error, BOOL debug)

{
    LIBSLU_SOLUTION* libsluHdlr = NULL;

    /*Verifica validitÃ  parametri */
    if (error == NULL)
    {
        if (debug == TRUE)
        {
            printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "NULL parameter - error");
        }
        return NULL;
    }

    if (matrixFile == NULL)
    {
        *error = LIBSLU_ERR_PARAM;
        if (debug == TRUE)
        {
            printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "NULL parameter - matrixFile");
        }
        return NULL;
    }

    /* Alloca la memoria necessaria per contenere una struttura LIBSLU_SOLUTION */

    libsluHdlr = (LIBSLU_SOLUTION*) malloc(sizeof(LIBSLU_SOLUTION));
    if (libsluHdlr == NULL)
    {
        *error = LIBSLU_ERR_MALLOC;
        /* Errore malloc */
        if (debug == TRUE)
        {
            printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_MALLOC, "malloc error");
        }
        return NULL;
    }

    /* Inizializza la memoria */
```

```
61.     memset(libsluHdlr, 0, sizeof(LIBSLU_SOLUTION));
62.     /* Copia stringa nome file matrice */
63.     libsluHdlr->matrixFile = (char*) malloc(strlen(matrixFile) + 1);
64.     strcpy(libsluHdlr->matrixFile, matrixFile);
65.     if(libsluHdlr->matrixFile == NULL)
66.     {
67.         *error = LIBSLU_ERR_MALLOC;
68.         /* Errore malloc */
69.         if (debug == TRUE)
70.         {
71.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_MALLOC, "malloc error"
    );
72.         }
73.         goto error_free;
74.     }
75.
76.     return libsluHdlr;
77.
78.     error_free:
79.
80.         if(libsluHdlr != NULL)
81.         {
82.             free(libsluHdlr);
83.         }
84.
85.         return NULL;
86. }
87.
88.
89. LIBSLU_RET_CODE  libslu_solvesparse(LIBSLU_SOLUTION* solutionHdlr, BOOL debug)
90. {
91.     FILE* fp;                    /* File pointer a file matrice (Formato HB) */
92.     SuperMatrix A, L, U, B;      /* Matrici e vettori in formato superLU     */
93.     int m,n;                     /* m : numero righe, n : numero colonne     */
94.     int nonZero;                 /* Numero di elementi diversi da 0 in A     */
95.     double* nzVal;               /* Puntatore a valori diversi da zero in A  */
96.     int* rowind;                 /* Indice righe elementi in nzVal           */
97.     int* colptr;                 /* Indice elementi inizio colonne in nzVal  */
98.     double* xe =  NULL;          /* Vettore unitario di lunghezza m          */
99.     double* b = NULL;            /* Vettore dei termini noti                 */
100.    int* permRow = NULL;         /* Vettore permutazioni righe               */
101.    int* permCol = NULL;         /* Vettore permutazioni colonne             */
102.    superlu_options_t options;   /* Opzioni solver SuperLU                   */
103.    SuperLUStat_t stat;          /* Statistiche solver SuperLU               */
104.    int info;
105.    int ret;
106.    int i = 0;
107.
108. #ifdef WIN32
109.     LARGE_INTEGER StartingTime;   /* TIMESTAMP prima di invocare il solver  */
110.     LARGE_INTEGER EndingTime;     /* TIMESTAMP dopo aver invocato il solver */
111.     LARGE_INTEGER Frequency;      /* Tick per secondo CPU                   */
112.     LARGE_INTEGER ElapsedMicroseconds;  /* Tempo impiegato in microsecondi  */
113. #elif __APPLE__
114.     unsigned long StartingTime;   /* TIMESTAMP prima di invocare il solver  */
115.     unsigned long EndingTime;     /* TIMESTAMP dopo aver invocato il solver */
116.     unsigned long Elapsed;        /* Differenza tra inizio e fine           */
117.     unsigned long ElapsedNano;    /* Tempo impiegato in nanosecondi         */
118.     static mach_timebase_info_data_t    sTimebaseInfo;
119. #endif
120.
121.
122.     /* Verifica validità  parametri */
123.     if (solutionHdlr == NULL)
124.     {
125.         if (debug == TRUE)
126.         {
127.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "NULL paramet
    er - solutionHdlr");
128.         }
129.         return LIBSLU_ERR_PARAM;
```

```
130.     }
131.
132.     if (solutionHdlr->matrixFile == NULL)
133.     {
134.         if (debug == TRUE)
135.         {
136.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "NULL paramet
    er - solutionHdlr->matrixFile");
137.         }
138.         return LIBSLU_ERR_PARAM;
139.     }
140.
141.     /* Apre file matrice */
142.     fp = fopen(solutionHdlr->matrixFile, "rb");
143.     if (fp == NULL)
144.     {
145.         /* Errore apertura file */
146.         if (debug == TRUE)
147.         {
148.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "fopen error"
    );
149.         }
150.         return LIBSLU_ERR_FOPEN;
151.     }
152.
153.     /* Converte file matrice da formato MM a formato interno CC */
154.     if (libslu_read_mm_file(fp,&n,&m,&nonZero,&nzVal,&rowind,&colptr) != 0)
155.     {
156.         fclose(fp);
157.         /* Errore lettura file file */
158.         if (debug == TRUE)
159.         {
160.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_MM_FILE, "MM file re
    ad error");
161.         }
162.         return LIBSLU_ERR_MM_FILE;
163.     }
164.
165.     /* Close file */
166.     fclose(fp);
167.
168.     /* Popola una struttura SuperMatrix utilizzando A (rappresentazione interna
    SuperLU) */
169.     dCreate_CompCol_Matrix(&A, m, n, nonZero, nzVal, rowind, colptr, SLU_NC, SLU
    _D, SLU_GE);
170.
171.     /* Costruisce vettore termini noti b = A * xe. xe è il vettore unitario di d
    imensione pari a m */
172.     /* Alloca xe */
173.     xe = (double*) malloc(sizeof(double) * m);
174.     if (xe == NULL)
175.     {
176.         ret = LIBSLU_ERR_MALLOC;
177.         if (debug == TRUE)
178.         {
179.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_MALLOC, "malloc erro
    r");
180.         }
181.         goto error_free;
182.     }
183.     /* Inizializza xe */
184.     for (i = 0; i < m; i++)
185.     {
186.         xe[i] = 1.0;
187.     }
188.     /* Ricava b */
189.     b = libslu_matrix_vector(m, n, nonZero, rowind, colptr, nzVal, xe);
190.
191. #ifdef LIBSLU_FILE_TEST
192.     /* Scrive vettore termini noti in file di test */
193.     libslu_print_vector_file("/Users/Username/Documents/testB.txt", b, m);
```

```
194.     printf("testB ");
195. #endif
196. 
197.     /* Converte B in formato interno SuperLU */
198.     dCreate_Dense_Matrix(&B, m, 1, b, m, SLU_DN, SLU_D, SLU_GE);
199. 
200.     /* Inizializza vettori permutazioni */
201.     permRow = (int*) malloc(sizeof(int) * m);
202.     if (permRow == NULL)
203.     {
204.         ret = LIBSLU_ERR_MALLOC;
205.         if (debug == TRUE)
206.         {
207.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_MALLOC, "malloc erro
    r");
208.         }
209.         goto error_free;
210.     }
211.     permCol = (int*) malloc(sizeof(int) * m);
212.     if (permCol == NULL)
213.     {
214.         ret = LIBSLU_ERR_MALLOC;
215.         if (debug == TRUE)
216.         {
217.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_MALLOC, "malloc erro
    r");
218.         }
219.         goto error_free;
220.     }
221. 
222.     /* Opzioni di default per SuperLU */
223.     set_default_options(&options);
224.     options.ColPerm = COLAMD;
225. 
226.     /* Inizializza statistiche */
227.     StatInit(&stat);
228. 
229. #ifdef WIN32
230.     /* Acquisce un timestamp ad alta risoluzione (WIN32 API)*/
231.     QueryPerformanceCounter(&StartingTime);
232.     /* Acquisisce frequenza CPU in tick per secondo         */
233.     QueryPerformanceFrequency(&Frequency);
234. #elif __APPLE__
235.     /* Acquisce un timestamp in mach_absolute_time */
236.     StartingTime = mach_absolute_time();
237. #endif
238. 
239.     /* ***** Risolve il sistema lineare ****** */
240.     dgssv(&options, &A, permCol, permRow, &L, &U, &B, &stat, &info);
241. 
242. #ifdef WIN32
243.     /* Acquisce un timestamp ad alta risoluzione (WIN32 API)      */
244.     QueryPerformanceCounter(&EndingTime);
245.     /* Calcola il numero di microsecondi impiegati per il calcolo */
246.     ElapsedMicroseconds.QuadPart = EndingTime.QuadPart -
     StartingTime.QuadPart;
247.     ElapsedMicroseconds.QuadPart *= 1000;
248.     ElapsedMicroseconds.QuadPart /= Frequency.QuadPart;
249.     /* Converte il millisecondi                                   */
250.     solutionHdlr-
    >solutionTime = (unsigned long) (ElapsedMicroseconds.QuadPart);
251. #elif __APPLE__
252.     /* Acquisce un timestamp in mach_absolute_time */
253.     EndingTime = mach_absolute_time();
254.     /* Calcola la differenza e la converte in millisecondi */
255.     Elapsed = EndingTime - StartingTime;
256.     (void) mach_timebase_info(&sTimebaseInfo);
257.     ElapsedNano = Elapsed * sTimebaseInfo.numer / sTimebaseInfo.denom;
258.     ElapsedNano = ElapsedNano / 1000000;
259.     solutionHdlr->solutionTime =  ElapsedNano;
260. #endif
```

```
261.
262. #ifdef LIBSLU_FILE_TEST
263.     /* Scrive vettore soluzione in file di test */
264.     libslu_print_vector_file("/Users/Username/Documents/testSolution.txt", b, m)
     ;
265.     printf("testSolution ");
266. #endif
267.
268.     /* Copia risultati in struttura LIBSLU_SOLUTION */
269.     solutionHdlr->rows = m;
270.     solutionHdlr->cols = n;
271.     solutionHdlr->nonZero = nonZero;
272.     solutionHdlr->solution = b;
273.     libslu_relative_error(b, xe, m, &(solutionHdlr->relError));
274.
275.     ret = LIBSLU_NO_ERR;
276.
277.     error_free:
278.
279.         if (permRow != NULL)
280.         {
281.             free(permRow);
282.         }
283.         if (permCol != NULL)
284.         {
285.             free(permCol);
286.         }
287.         if (xe != NULL)
288.         {
289.             free(xe);
290.         }
291.
292.         Destroy_CompCol_Matrix(&A);
293.         Destroy_SuperMatrix_Store(&B);
294.         Destroy_SuperNode_Matrix(&L);
295.         Destroy_CompCol_Matrix(&U);
296.         StatFree(&stat);
297.
298.         return ret;
299.
300. }
301.
302. /* Moltiplica una matrice per un vettore. Ritorna un puntatore al risultato, per
      il quale alloca la memoria necessaria */
303. double* libslu_matrix_vector(int m, int n, int ncc, int icc[], int ccc[], double
      acc[], double x[])
304. {
305.     double *b;
306.     int i;
307.     int j;
308.     int k;
309.
310.     b = (double *)malloc(m * sizeof(double));
311.
312.     for (i = 0; i < m; i++)
313.     {
314.         b[i] = 0.0;
315.     }
316.
317.     for (j = 0; j < n; j++)
318.     {
319.         for (k = ccc[j]; k < ccc[j + 1]; k++)
320.         {
321.             i = icc[k];
322.             b[i] = b[i] + acc[k] * x[j];
323.         }
324.     }
325.     return b;
326. }
327.
```

```
328. LIBSLU_RET_CODE  libslu_print_solution(LIBSLU_SOLUTION* solutionHdlr, BOOL debug
     )
329. {
330.     /* Verifica validità parametri */
331.     if (solutionHdlr == NULL)
332.     {
333.         if (debug == TRUE)
334.         {
335.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "NULL paramet
     er - solutionHdlr");
336.         }
337.         return LIBSLU_ERR_PARAM;
338.     }
339.
340.     printf("MATRIX: %s\r\n", solutionHdlr->matrixFile);
341.     printf("rows: %u\r\n", solutionHdlr->rows);
342.     printf("columns: %u\r\n", solutionHdlr->cols);
343.     printf("Non zero elements: %u\r\n", solutionHdlr->nonZero);
344.     printf("Solution time: %u ms\r\n", solutionHdlr->solutionTime);
345.     printf("Relative error: %.30lf\r\n", solutionHdlr->relError);
346.     printf("Memory usage: %u\r\n", solutionHdlr->memoryUsage);
347.     printf("\r\n");
348.
349.     return LIBSLU_NO_ERR;
350. }
351.
352. LIBSLU_RET_CODE  libslu_free(LIBSLU_SOLUTION* solutionHdlr, BOOL debug)
353. {
354.     /* Verifica validità parametri */
355.     if (solutionHdlr == NULL)
356.     {
357.         if (debug == TRUE)
358.         {
359.             printf("LIBSLU ERROR : %d (%s)\r\n", LIBSLU_ERR_PARAM, "NULL paramet
     er - solutionHdlr");
360.         }
361.         return LIBSLU_ERR_PARAM;
362.     }
363.
364.     if (solutionHdlr->matrixFile != NULL)
365.     {
366.         free(solutionHdlr->matrixFile);
367.     }
368.
369.     return LIBSLU_NO_ERR;
370.
371. }
372.
373. LIBSLU_RET_CODE libslu_read_mm_file(FILE* fp, int* n, int* m, int* nonZero, doub
     le** nzVal, int** rowind, int** colptr)
374. {
375. #define MAX_LINE_LEN    512
376.
377.     char lineBuf[MAX_LINE_LEN];   /* Buffer in cui viene copiata la riga corrent
     e del file */
378.     int curCol = 0;       /* Indice colonna corrente durante scansione file */
379.     int curItem = 0;      /* Indice elemento corrente                       */
380.     double* nzVal_int = NULL;
381.     int* rowind_int = NULL;
382.     int* colptr_int = NULL;
383.     int ret;
384.
385.     memset(lineBuf, '\0', sizeof(lineBuf));
386.
387.     /* Scarta i commenti iniziali (righe che iniziano con '%') */
388.     while (fgets(lineBuf, MAX_LINE_LEN, fp) != NULL)
389.     {
390.         if (lineBuf[0] == '%')
391.         {
392.             continue;
393.         }
```

```
394.         else
395.         {
396.             break;
397.         }
398.     }
399.
400.     /* Legge prima riga */
401.     if (sscanf(lineBuf, "%d %d %d", n, m, nonZero) != 3)
402.     {
403.         /* Errore intestazione file */
404.         return LIBSLU_ERR_MM_HEADER;
405.     }
406.
407.     /* Alloca i buffer per la rappresentazione CC */
408.     nzVal_int = (double*) malloc((*nonZero) * sizeof(double));
409.     rowind_int = (int*) malloc((*nonZero) * sizeof(int));
410.     colptr_int = (int*) malloc(((*n) + 1) * sizeof(int));
411.     if (nzVal_int == NULL || rowind_int == NULL || colptr_int == NULL)
412.     {
413.         ret = LIBSLU_ERR_MALLOC;
414.         goto error_free;
415.     }
416.
417.     /* Legge le righe della matrice */
418.     while (fgets(lineBuf, MAX_LINE_LEN, fp) != NULL)
419.     {
420.         int col;
421.         int row;
422.         double value;
423.
424.         if (sscanf(lineBuf, "%d %d %lf", &row, &col, &value) != 3)
425.         {
426.             /* Errore */
427.             ret = LIBSLU_ERR_MM_BODY;
428.             goto error_free;
429.         }
430.
431.         nzVal_int[curItem] = value;
432.         rowind_int[curItem] = row - 1;
433.
434.         if (col != curCol)
435.         {
436.             colptr_int[col - 1] = curItem;
437.             curCol = col;
438.         }
439.         curItem++;
440.     }
441.     colptr_int[(*n)] = curItem;
442.
443.     *nzVal = nzVal_int;
444.     *rowind = rowind_int;
445.     *colptr = colptr_int;
446.
447.     return LIBSLU_NO_ERR;
448.
449. error_free:
450.
451.     if (nzVal_int != NULL)
452.     {
453.         free(nzVal_int);
454.     }
455.     if (rowind_int != NULL)
456.     {
457.         free(rowind_int);
458.     }
459.     if (colptr_int != NULL)
460.     {
461.         free(colptr_int);
462.     }
463.
464.     return ret;
```

```
}

LIBSLU_RET_CODE libslu_relative_error(double x[], double xe[], int len, double*
relError)
{
    int i;
    double norm1 = 0;
    double norm2 = 0;

    for (i = 0; i < len; i++)
    {
        norm1 = norm1 + pow((x[i] - xe[i]), 2);
    }
    norm1 = sqrt(norm1);

    for (i = 0; i < len; i++)
    {
        norm2 = norm2 + pow(xe[i], 2);
    }
    norm2 = sqrt(norm2);

    *relError = norm1 / norm2;
}

void libslu_print_vector_file(char* filePath, double* solution, int solutionLen)

{
    int i;
    FILE *fp;

    fp = fopen(filePath, "wb");
    if (fp == NULL)
    {
        return;
    }

    for(i = 0; i < solutionLen; i++)
    {
        fprintf(fp, "%.30lf\r\n", solution[i]);
    }
    fclose(fp);
}
```

Progetto 2

Lo scopo del progetto è studiare una estensione della compressione jpeg introducendo, rispetto allo standard, la possibilità di utilizzare blocchetti di 8N × 8N pixel con N scelto dall'utente invece che semplicemente 8 × 8.

Il linguaggio scelto per l'implementazione è Python versione 2.7.
Risultato del progetto è la creazione di una libreria di funzioni che implementa il metodo di compressione sopra anticipato e di un'interfaccia grafica che consente all'utente di applicare lo stesso su di un'immagine a sua scelta e visualizzarne il risultato, al variare di una serie di parametri di input per l'algoritmo alla base della compressione.

Segue una descrizione delle librerie importate e del loro utilizzo nell'implementazione del progetto.

2.1 - Numpy

Numpy è una libreria per il calcolo matriciale nata nel 2005 con lo scopo di aprire all'utilizzo di Python quale linguaggio di programmazione per la comunità scientifica. Come MATLAB implementa le interfacce BLAS e LAPACK.
La libreria consiste di una serie di funzioni che costituiscono l'interfaccia verso l'implementazione a basso livello in linguaggio C. E' messa a disposizione una API per l'interfacciamento con quest'ultimo.
La struttura dati 'ndarray' implementa un array n-dimensionale, equipaggiandolo di metodi per definizione e accesso efficiente ad elementi di matrici e vettori. Differentemente dalle liste Python, da cui riprende i classici metodi di accesso con indici ed intervalli di slicing, la struttura 'ndarray' è tipizzata e omogenea (non è possibile inserire in essa due elementi di tipo diverso). Salvo l'utilizzo del metodo 'copy', ogni operazione di slicing non comporta la copia della struttura dati. Da ciò segue come le operazioni effettuate su una qualsiasi sottomatrice intervengono alla modifica della matrice sorgente.
Interessante ottimizzazione presente nella libreria è il broadcasting. Il broadcasting è l'operazione di estensione degli elementi di uno scalare, un vettore o una matrice (questi ultimi aventi almeno una dimensione eguale a 1) al momento dell'esecuzione di un'operazione binaria con una matrice le cui dimensioni sono maggiori o eguali a 1. Operazioni quali la moltiplicazione di una matrice a tre dimensioni per un'altra matrice a due dimensioni, compatibile con ognuna delle tre sottomatrici costituenti la prima, non necessita della scrittura di 3 operazioni di moltiplicazione tra matrici: il tutto è gestito in maniera trasparente dalla libreria che, senza ulteriori allocazioni di memoria procede ad eseguire l'operazione per mezzo di una procedura di moltiplicazione efficiente implementata in linguaggio C.

2.2 - Scipy

Come descritto in precedenza per il Progetto 1, Scipy è una libreria general purpose per l'analisi numerica.
I moduli utilizzati per l'implementazione dell'algoritmo di compressione sono stati misc e fftpack, rispettivamente utili per la lettura di un'immagine da un file e per il calcolo di DCT e DCT inversa.

2.3 - Matplotlib

Matplotlib è una libreria che fornisce un ambiente di plotting simile a quello di MATLAB.
Fa uso di Numpy ed è compresa all'interno di Scipy. Fornisce un'interfaccia grafica con una toolbar di comando simile a quella di MATLAB che permette di intervenire sul grafico navigandolo, operare uno zoom, esportare quanto visualizzato. Non si limita alla rappresentazione di funzioni matematiche consentendo la rappresentazione di qualsiasi immagine 2D comprese le immagini bitmap.
Oltre all'uso interattivo simile a quanto offerto da MATLAB, la libreria è pensata per permettere l'inserimento dei grafici quali frame di applicazioni munite di interfaccia grafica. E' pertanto possibile generare, a partire da un grafico creato, i relativi widget da utilizzare con le librerie grafiche GTK+, WxWidgets, Tkinter e Qt.
Il suo utilizzo ha permesso l'embedding, nella GUI creata, di un'interfaccia di plotting che consenta visualizzazione e navigazione contemporanee di immagine sorgente ed immagine compressa.

2.4 - Tkinter

Tkinter è la libreria grafica scelta dalla comunità di mantainers del linguaggio Python per la creazione dell'IDE presente in ogni installazione standard.
Nasce dalla libreria grafica Tk del linguaggio Tcl con il quale è distribuita.
Tkinter è un'interfaccia che permette un utilizzo trasparente della libreria Tk, mediante un interprete Tcl inserito nel linguaggio Python.
Il modulo ttk della libreria permette alle applicazioni che ne fanno uso di acquisire il look and feel del sistema operativo su cui sono eseguite, rendendo le stesse system indipendent, dal punto di vista dell'interfaccia utente.

2.5 - Compressione dell'immagine

La lettura dell'immagine è effettuata mediante la funzione 'imread' del modulo 'scipy.misc'. Il secondo parametro di tale funzione viene settato a True al fine di ottenere dall'immagine letta una sola matrice di valori numerici interi in luogo delle tre matrici RGB.
I valori appartenenti alla matrice L sono il risultato della seguente combinazione lineare definita nello standard ITU-R BT.601:

$$l_{i,j} = r_{i,j} \frac{299}{1000} + g_{i,j} \frac{587}{1000} + b_{i,j} \frac{114}{1000}$$

con r, g e b elementi della matrici dei tre colori di sopra.
Gli elementi contenuti in tale matrice sono valori floating point a 32 bit.
Se il numero di righe o di colonne non sono multipli della dimensione dei blocchi scelta, mediante le funzioni vstack e hstack della libreria numpy, vengono aggiunte rispettivamente una riga uguale all'ultima o una colonna uguale all'ultima alla matrice, rendendola perfettamente divisibile nei blocchi di sopra.
A seguito della sottrazione del valore numerico 128 da ogni elemento della matrice ottenuta, viene calcolata la DCT.
Tale modulo utilizza un'interfaccia di wrapping verso la libreria Fortran FFTPACK, scritta da Paul N. Swarztrauber nel 1985; tale libreria permette il calcolo di una serie di trasformate tra cui quella del coseno, mediante un algoritmo di calcolo della FFT in un tempo computazionale di $O(n \log n)$ operazioni aritmetiche.

La funzione 'dct' prende quale primo argomento un ndarray (vettore o matrice) contenente i valori su cui calcolare la DCT e una serie di parametri opzionali tra i quali tipo, asse su cui applicare la funzione e normalizzazione da applicare.
Il tipo utilizzato nell'applicazione in oggetto, configurazione di default per la funcione 'dct', è la DCT-II; il parametro norm è invece posto a 'ortho'. Tali due configurazioni permettono il calcolo della DCT ottenibile con il comando 'dct' di MATLAB, corrispondente all'espressione

$$y_k = 2\sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi(2n+1)k}{2N}\right), \qquad 0 \le k \le N$$

Gli y_k vengono poi moltiplicati per una costante così definita

$$\begin{cases} \sqrt{1/4N}\,, & se\ k = 0 \\ \sqrt{1/2N}\,, & altrimenti \end{cases}$$

La funzione 'dct' descritta esegue il calcolo dei valori suddetti relativamente ad un solo asse. Al fine della compressione è necessario applicare la DCT a righe e colonne della matrice. L'implementazione esegue la funzione 'dct' sulla trasposta della matrice in oggetto; il risultato è frutto di una trasposizione e di un'ulteriore applicazione della funzione 'dct'. In tal modo la DCT viene eseguita prima sulle colonne e poi sulle righe della matrice.
Sulla base della dimensione dei blocchi e della qualità viene calcolata la matrice di quantizzazione. A partire dalla matrice di quantizzazione standard, utilizzata nella compressione lossy della libreria libjpeg, viene costruita una nuova matrice di dimensione $8N$x $8N$. Presa Q matrice di quantizzazione d'origine, i valori della nuova matrice sono ricavati secondo la funzione

$$q'_{i,j} = q_{\left\lfloor \frac{i}{N} \right\rfloor, \left\lfloor \frac{j}{N} \right\rfloor}$$

dove con $\left\lfloor \frac{x}{y} \right\rfloor$ è rappresentata la divisione intera.
Segue la divisione, round e successiva moltiplicazione dei valori contenuti nei singoli blocchi per la matrice di quantizzazione.
Mediante la funzione 'idct' viene applicata, agli elementi della matrice, la DCT inversa, corrispondente alla formula

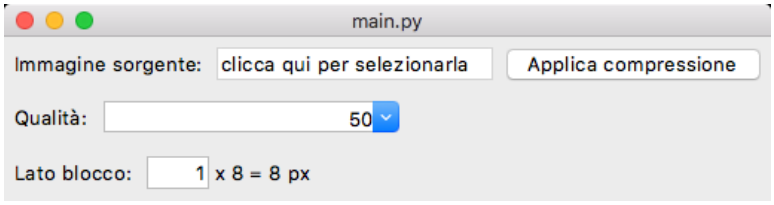
$$y_k = \frac{x_0}{\sqrt{N}} + \frac{1}{\sqrt{N}}\sum_{n=1}^{N-1} x_n \cos\left(\frac{\pi n(2k+1)}{2N}\right), \qquad 0 \le k \le N$$

Viene sommato 128 ad ogni elemento della matrice.
Differentemente da quanto avviene nell'ambiente di programmazione fornito da MATLAB, l'operazione di casting degli elementi della matrice, qualora gli stessi siano inferiori a 0 o maggiori di 255, da numeri floating point a 32 bit a interi senza segno a 8 bit è soggetta ad errori di overflow. Per questo viene applicata una funzione ad ogni elemento della matrice, che restringe l'intervallo dei valori ammessi a $[0; 255]$.

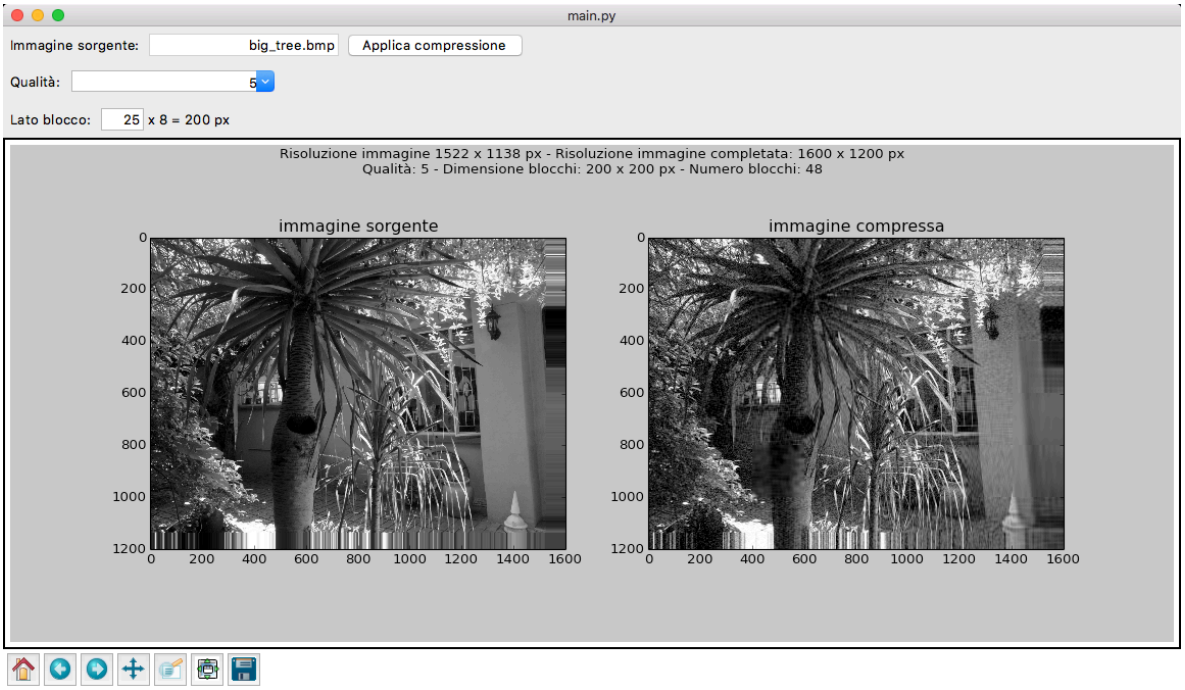
2.6 - Interfaccia grafica

Una volta avviata, l'applicazione mostra una semplice form composta dai seguenti elementi:

- un box per l'input testuale all'interno del quale compare un messaggio che indica all'utente di cliccare lo stesso per selezionare un immagine o il nome dell'immagine selezionata;
- un menù a tendina che permette di selezionare la qualità della compressione, un valore numerico compreso tra 1 e 100;
- un box per l'input testuale nel quale inserire la dimensione N che, moltiplicata per 8, darà la lunghezza del lato di ognuno dei blocchi con cui l'immagine sarà divisa e sui quali verrà applicata la compressione;
- un bottone con la scritta 'Applica compressione' che, una volta cliccato, facendo uso dei parametri precedentemente scelti dall'utente, comprimerà l'immagine sorgente e la mostrerà affiancata all'immagine risultato della compressione, in uno spazio sottostante alla form in oggetto o in sostituzione delle immagini presenti in tale spazio.



Il programma mostra il confronto tra l'immagine originale e quella ottenuta dalla compressione jpeg.



L'interfaccia è munita di una toolbar nella parte inferiore che consente navigazione, zoom ed esportazione delle immagini sorgente e compressa mostrate a seguito della procedura di compressione. L'operazione di zoom condivide gli assi delle due immagini, ciò consente all'applicazione dello zoom ad una, la ripetizione dell'operazione sull'altra immagine.

main.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# ****************************************
#   Metodi del Calcolo Scientifico 2016
# ----------------------------------------
#   743464 Banfi Alessandro
#   735722 Curatitoli Mattia
#   742752 Ghelfi Camillo
# ****************************************

from Tkinter import *
import ttk
import tkFileDialog
import tkMessageBox as mbox
import sys
import os.path
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
from matplotlib.figure import Figure
from scipy.misc import imread
import matplotlib.pyplot as plt
import pylab
import numpy as np
import dct

root = Tk()
root.title(sys.argv[0])
root.columnconfigure(0, weight=1)

subframe1 = ttk.Frame(root)
subframe1.pack(side=TOP, fill=X)

label1 = ttk.Label(subframe1, text="Immagine sorgente:")
label1.pack(side=LEFT, padx=5, pady=5)

fileCnt = StringVar()
fileCnt.set('clicca qui per selezionarla')
file = ttk.Entry(subframe1, textvariable=fileCnt, state=DISABLED)

select1Cnt = StringVar()
select1Cnt.set(50)

src_file = ''
def choose_src_img(*args):
    global src_file
    types = [('BMP', '*.bmp'), ('Tutti i file', '*.*')]
    source = tkFileDialog.askopenfilename(title='Scelta dell\'immagine sorgente', filetypes=types)
    if source != '' and os.path.isfile(source):
        file["justify"] = RIGHT
        fileCnt.set(os.path.split(source)[1])
        src_file = source

file.bind('<Button-1>', choose_src_img)
file.pack(side=LEFT, pady=5)

subframe2 = ttk.Frame(root)
subframe2.pack(side=TOP, fill=X)

label2 = ttk.Label(subframe2, text="QualitÃ :")
label2.pack(side=LEFT, padx=5, pady=5)
select1Cnt = StringVar()

qPrev = "50"
def validate_quality(*args):
```

```
67.     global qPrev
68.     try:
69.         q = int(select1Cnt.get())
70.         if (q < 1) or (q > 100):
71.             select1Cnt.set(qPrev)
72.             return False
73.         qPrev = str(q)
74.     except ValueError:
75.         select1Cnt.set(qPrev)
76.         return False
77.
78. select1Cnt.trace('w', validate_quality)
79. select1 = ttk.Combobox(subframe2, textvariable=select1Cnt)
80. select1["values"] = range(1, 101)
81. select1.set(50)
82. select1.pack(side=LEFT, pady=5)
83. select1["justify"] = RIGHT
84.
85. subframe3 = ttk.Frame(root)
86. subframe3.pack(side=TOP, fill=X)
87. label3 = ttk.Label(subframe3, text="Lato blocco:")
88. label3.pack(side=LEFT, padx=5, pady=5)
89.
90. block_size = StringVar()
91. label4textVar = StringVar()
92.
93. prev = "1"
94. def update_block_size():
95.     global prev
96.     try:
97.         bs = int(block_size.get())
98.         if not bs > 0:
99.             raise ValueError
100.                prev = str(bs)
101.                label4textVar.set('x 8 = ' + str(bs * 8) + ' px')
102.            except ValueError:
103.                block_size.set(prev)
104.                return False
105.            return True
106.
107.        edit = ttk.Entry(subframe3, textvariable=block_size,
108.            validatecommand=update_block_size, validate='focusout')
109.        edit.insert(0, "1")
110.        edit["justify"] = RIGHT
111.        edit["width"] = 4
112.        edit.pack(side=LEFT, pady=5)
113.
114.        label4 = ttk.Label(subframe3)
115.        label4['textvariable'] = label4textVar
116.        label4textVar.set('x 8 = 8 px')
117.        label4.pack(side=LEFT, pady=5)
118.
119.        subframe4 = ttk.Frame(root)
120.        subframe4.pack(side=TOP, fill=BOTH, expand=YES)
121.
122.        f = pylab.figure()
123.        ax1 = plt.subplot(1, 2, 1)
124.        ax2 = plt.subplot(1, 2, 2, sharex=ax1, sharey=ax1)
125.        canvas = FigureCanvasTkAgg(f, subframe4)
126.        printed = False
127.        canvas.get_tk_widget().update()
128.        canvas._tkcanvas.update()
129.        canvas.show()
130.        plt.close(f)
131.
132.        def apply_compression():
133.            global printed
134.            if src_file == '':
135.                mbox.showerror('Nessuna immagine selezionata',
136.                'E\' necessario selezionare un\'immagine prima di procedere alla c
    ompressione!')
```

```
137.                 return
138.             elif not os.path.isfile(src_file):
139.                 mbox.showerror('Immagine selezionata non valida',
140.                 'L\'immagine selezionata non Ã¨ un file valido per la compressione
    !')
141.                 return
142.             img = imread(src_file, True)
143.             src_res = img.shape
144.             bs = int(block_size.get()) * 8
145.             img = dct.complete_img(img, bs)
146.             f.texts = []
147.             f.suptitle((u'Risoluzione immagine %d x %d px' + \
148.                 u' - Risoluzione immagine completata: %d x %d px' + \
149.                 u"\nQualitÃ : %s - Dimensione blocchi: %d x %d px -
     Numero blocchi: %d") \
150.                 % (src_res[1], src_res[0], img.shape[1], img.shape[0],
151.                 qPrev, bs, bs,
152.                 (img.shape[1] * img.shape[0]) / (bs * bs)))
153.             ax1.clear()
154.             ax1.imshow(img, cmap='Greys_r')
155.             ax1.set_title('immagine sorgente')
156.             ax1.set_adjustable('box-forced')
157.             shk = img.copy()
158.             shk = dct.compress_img(shk, int(select1Cnt.get()), bs)
159.             ax2.clear()
160.             ax2.imshow(shk, cmap='Greys_r')
161.             ax2.set_title('immagine compressa')
162.             ax2.set_adjustable('box-forced')
163.             if not printed:
164.                 canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=YES)
165.                 canvas._tkcanvas.pack(fill=BOTH, expand=YES)
166.                 toolbar = NavigationToolbar2TkAgg(canvas, subframe4)
167.                 toolbar.pack(side=BOTTOM, fill=X)
168.                 toolbar.update()
169.                 printed = True
170.             canvas.draw()
171.
172.         button1 = ttk.Button(subframe1, text="Applica compressione", command=apply
    _compression)
173.         button1.pack(side=LEFT, padx=5, pady=5)
174.
175.         root.mainloop()
```

dtc.py

```
import os
import sys
import numpy as np
from scipy.misc import imread
from scipy.fftpack import dct, idct
import matplotlib.pyplot as plt
import pylab

Q = np.array([[16, 11, 10, 16, 24, 40, 51, 61],
              [12, 12, 14, 19, 26, 58, 60, 55],
              [14, 13, 16, 24, 40, 57, 69, 56],
              [14, 17, 22, 29, 51, 87, 80, 62],
              [18, 22, 37, 56, 68, 109, 103, 77],
              [24, 35, 55, 64, 81, 104, 113, 92],
              [49, 64, 78, 87, 103, 121, 120, 101],
              [72, 92, 95, 98, 112, 100, 103, 99]],
              dtype='float32')

def complete_img(img, bs=8):
    if img.shape[0] % bs != 0:
        delta_r = bs - (img.shape[0] % bs)
        tmp = img[-1,:]
        for _ in xrange(delta_r - 1):
            tmp = np.vstack((tmp, img[-1,:]))
        img = np.vstack((img, tmp))
    if img.shape[1] % bs != 0:
        delta_c = bs - (img.shape[1] % bs)
        col = img[:,-1]
        col = col[:,np.newaxis]
        tmp = col
        for _ in xrange(delta_c - 1):
            tmp = np.hstack((tmp, col))
        img = np.hstack((img, tmp))
    return img

def read_img(filename):
    if not(os.path.isfile(filename)):
        print "error: %s is not a regular file!" % filename
        sys.exit(1)
    return imread(filename, True)

def apply_dct(img, bs):
    for r in xrange(0, img.shape[0], bs):
        for c in xrange(0, img.shape[1], bs):
            img[r:r+bs, c:c+bs] = dct(dct(img[r:r+bs, c:c+bs].T, norm='ortho').T, norm='ortho')
    return img

def apply_idct(img, bs):
    for r in xrange(0, img.shape[0], bs):
        for c in xrange(0, img.shape[1], bs):
            img[r:r+bs, c:c+bs] = idct(idct(img[r:r+bs, c:c+bs].T, norm='ortho').T, norm='ortho')
    return img

def get_quantization_matrix(q, bs):
    if q <= 0:
        q = 1
    elif q >= 100:
        return np.ones((bs, bs), dtype='float32')
    qf = 0

    if q >= 50:
        qf = (200 - 2 * q) / 100.0
    else:
        qf = (5000.0 / q) / 100

    N = bs / 8
```

```
67.
68.     Qc = np.array(qf * Q)
69.     cmpl = lambda x, y: Qc[x/N, y/N]
70.     return np.fromfunction(cmpl, (bs, bs), dtype='int32')
71.
72. def lossless_cast(val):
73.     if val < 0:
74.         return 0
75.     elif val > 255:
76.         return 255
77.     else:
78.         return val
79.
80. def compress_img(img, q, bs):
81.     # conversione dei valori della matrice a float32
82.     img = np.float32(img)
83.     # shift dei valori di -128
84.     img = img - 128
85.     # eseguo la dct sui blocchi bs di img
86.     img = apply_dct(img, bs)
87.     # ottengo la matrice di quantizzazione dati q e bs
88.     QN = get_quantization_matrix(q, bs)
89.     # divido ogni blocco (bs, bs) per la matrice QN
90.     for r in xrange(0, img.shape[0], bs):
91.         for c in xrange(0, img.shape[1], bs):
92.             img[r:r+bs, c:c+bs] = img[r:r+bs, c:c+bs] / QN
93.     # effettuo l'arrotondamento a intero dei valori della matrice
94.     img = np.round(img)
95.     # moltiplico ogni blocco (bs, bs) per la matrice QN
96.     for r in xrange(0, img.shape[0], bs):
97.         for c in xrange(0, img.shape[1], bs):
98.             img[r:r+bs, c:c+bs] = img[r:r+bs, c:c+bs] * QN
99.     # eseguo la idct sui blocchi bs di img
100.          img = apply_idct(img, bs)
101.          # shift dei valori di +128
102.          img = img + 128
103.          # effettuo l'arrotondamento a intero dei valori della matrice
104.          img = np.round(img)
105.          # effettuo il casting dei valori prima della conversione da float32 a

106.          # uint8
107.          cast = np.vectorize(lossless_cast)
108.          img = cast(img)
109.          # converto da float32 a uint8
110.          img = np.uint8(img)
111.          return img
```