UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Risolutore di sistemi lineari tramite l'algoritmo di Gauss-Jordan scritto in linguaggio C

Linear system solver using Gauss-Jordan's algorithm written in the C language

Relatore: Prof. Antonio Giunta

Laureando: Alessandro Pisent

ANNO ACCADEMICO 2021 – 2022

Data di laurea 21/09/2022

Ai miei nonni,
che anche se non ci sono più
sono sempre con me

Indice

1. INTRODUZIONE **5**

2. LE STRUTTURE DATI UTILIZZATE **6**

La matrice dei coefficienti 6
La matrice delle relazioni algebriche 6
Array delle equazioni linearmente dipendenti 6
Tipo strutturato 7

3. L'ALGORITMO DI GAUSS-JORDAN **8**

Possibili soluzioni 9
Analisi asintotica 10
Nel programma 10
L'algoritmo per trovare le relazioni algebriche tra le equazioni 11

4. STRUTTURA E ORGANIZZAZIONE DEL PROGRAMMA **12**

Descrizione del file di input 12
Descrizione del file di Output 12

5. FASI DEL PROGRAMMA **14**

1. Avvio e memorizzazione 14
2. Risoluzione del sistema 14
3. Stampa del risultato 14

6. OSSERVAZIONI FINALI **15**

BIBLIOGRAFIA **16**

SITOGRAFIA **16**

APPENDICE **17**

File: GaussJordan.c 17
File: GaussJordan.h 26
File: GaussJordan_Tester.c 27

1. Introduzione

In questa tesi descriverò il programma da me sviluppato che risolve sistemi di equazioni lineari; la loro risoluzione è fondamentale per molte applicazioni, per esempio per la programmazione lineare, per la fisica o per l'automatica.

Un sistema di equazioni lineari a coefficienti reali è un insieme S di equazioni del tipo:

$$S: \begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ..... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases}.$$

con $a_{ij}, b_h \in \mathbb{R}$.

Gli elementi $a_{i\,j}$ sono detti *coefficienti del sistema*, mentre $b_1,\ldots,b_m$ sono i *termini noti*. Le $x_1,\ldots,x_n$ sono le *incognite del sistema*. Risolvere il sistema significa trovare tutti i valori delle incognite che soddisfino contemporaneamente tutte le equazioni di S.

Esistono solo 3 tipi di sistemi:

1. Sistema impossibile, privo di soluzioni
2. Sistema con una soluzione unica
3. Sistema con infinte soluzioni

Inoltre esiste la situazione tale per cui delle equazioni del sistema sono linearmente dipendenti tra di loro. Potrebbe essere, ad esempio, che la seconda equazione abbia gli stessi coefficienti della prima equazione, e stesso termine noto.

Il programma da me sviluppato risolve i sistemi lineari, cioè identifica il tipo di sistema e scrive le sue soluzioni. Se un sistema è impossibile, lo identifica; se un sistema ha una soluzione unica, la scrive; se è un sistema con infinite soluzioni, trova l'insieme delle soluzioni con le incognite libere e infine, trova la dipendenza lineare tra le equazioni.

Alcuni software, come ad esempio Matlab, utilizzano fattorizzazioni di tipo LU e QR per risolvere sistemi lineari. Questi algoritmi hanno dalla loro parte che permettono di risolvere più rapidamente lo stesso sistema una seconda volta con termini noti differenti. Tuttavia LU e QR sono asintoticamente comparabili rispetto all'algoritmo scelto e Gauss-Jordan permette di trovare le dipendenze lineari tra le equazioni.

2. Le strutture dati utilizzate

Le strutture di dati fondamentali per il programma sono:

1. La matrice dei coefficienti
2. La matrice delle relazioni algebriche tra equazioni
3. Un array con tutte le equazioni linearmente dipendenti

La matrice dei coefficienti

Dato un sistema di n equazioni e m incognite, la matrice dei coefficienti è una matrice *(n+1)* x (*m+1*), cioè di *n+1* righe e *m+1* colonne. Le righe della matrice sono *n+1* perché la prima riga, o riga 0, viene lasciata libera per future applicazioni, cioè per la risoluzione di problemi della programmazione lineare. Le colonne della matrice sono *m+1* perché nella prima colonna ci sono i termini noti del sistema.

$$\text{MCoeff} = \left[\begin{array}{c|cccc} & & \dots & & \\ b_1 & a_{11} & a_{12} & \dots & a_{1m} \\ b_2 & a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ b_n & a_{n1} & a_{n2} & \dots & a_{nm} \end{array}\right]$$

La matrice delle relazioni algebriche

Dato un sistema di n equazioni, la matrice delle relazioni algebriche sarà una matrice quadrata *n* x *n*. Ogni elemento $a_{i\,j}$ è espressione di quante volte è stata aggiunta o sottratta la j-esima equazione dalla i-esima equazione, e gli elementi sulla diagonale $a_{i\,i}$ rappresentano per quanto è stata divisa la i-esima equazione.

$$\text{MRAlg} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & r_{nm} \end{bmatrix} \qquad \text{inizializzazione MRAlg} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

La matrice viene inizializzata in questo modo poiché inizialmente non è stata sottratta nessuna equazione dalle altre, e quindi ci sono 0 dappertutto, a parte sulle diagonali, dove c'è 1 poiché le equazioni non sono state ancora divise per nessun scalare.

Array delle equazioni linearmente dipendenti

Dato un sistema di n equazioni, l'array delle equazioni linearmente dipendenti contiene gli indici delle equazioni linearmente dipendenti del sistema.

$$aEDip = \begin{bmatrix} i_1 & i_2 & \dots & i_k \end{bmatrix}$$

Dato k , numero di equazioni linearmente dipendenti, k sono anche il numero di elementi nel array.

Tipo strutturato

Il tipo strutturato da me sviluppato, come in appendice, è:

```
typedef struct {
    double** MCoef;     /*Matrice con allocazione dinamica memoria*/
    int nEq; /*n*/          /*# di equazioni, # Righe*/
    int nIn; /*m*/          /*# di Incognite, # Colonne*/
    int* aEDip;           /*array con gli indici delle equazioni dipendenti */
    int nEDip;            /*# di elementi in $dipRow*/
    double** MRAlg;     /*matrice con relazioni algebriche tra righe*/
    double error;       /*precisione di errore*/
} Matrix;
```

Quindi contiene tutte le strutture dati precedentemente elencate e le racchiude.
Possiede anche le capienze delle matrici e degli array, oltre che un riferimento alla precisione con la quale un numero verrà considerato zero.

3. L'algoritmo di Gauss-Jordan

L'algoritmo di Gauss-Jordan permette, tramite l'utilizzo di *operazioni elementar*i sulla matrice dei coefficienti, di ottenere la matrice identità, e di quindi risolvere il sistema.

$$\mathrm{MCoeff}_{risolta} = \left[\begin{array}{c|cccc} & & \dots & & \\ b'_1 & 1 & 0 & \dots & 0 \\ b'_2 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ b'_n & 0 & 0 & \dots & 1 \end{array}\right]$$

Le operazioni elementari per una matrice sono : lo scambio di due equazioni, la moltiplicazione di entrambi i membri di una equazione per uno scalare non nullo, o la somma di un multiplo di un'equazione ad un'altra.

I passi fondamentali del algoritmo sono 2:

1. La normalizzazione sull'elemento di pivot i-esimo
2. L'azzeramento della colonna i-esima

Precisamente, iniziamo con la matrice dei coefficienti letta dal file di input:

$$\mathrm{MCoeff} = \left[\begin{array}{c|cccc} & & \dots & & \\ b_1 & a_{11} & a_{12} & \dots & a_{1m} \\ b_2 & a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ b_n & a_{n1} & a_{n2} & \dots & a_{nm} \end{array}\right]$$

Se l'elemento a_{11} è diverso da zero, dividiamo tutta la prima riga per l'elemento a_{11} , detto elemento di pivot, fino ad ottenere:

$$\mathrm{MCoeff}_1 = \left[\begin{array}{c|cccc} & & \dots & & \\ b^1_1 & 1 & a^1_{12} & \dots & a^1_{1m} \\ b_2 & a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ b_n & a_{n1} & a_{n2} & \dots & a_{nm} \end{array}\right]$$

Ora togliamo a_{i1} volte la prima riga così ottenuta, alla riga i-esima della matrice, questo per tutte le n-1 equazioni rimanenti, fino ad ottenere:

$$\mathrm{MCoeff}_1 = \left[\begin{array}{c|cccc} & & \dots & & \\ b^1_1 & 1 & a^1_{12} & \dots & a^1_{1m} \\ b^1_2 & 0 & a^1_{22} & \dots & a^1_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ b^1_n & 0 & a^1_{n2} & \dots & a^1_{nm} \end{array}\right]$$

Dove gli apici stanno ad indicare il fatto che a_{22} è diverso da a^1_{22} .

Quindi ora basta ripetere n-volte queste due operazioni per arrivare al risultato desiderato. Inoltre se ci saranno una equazione linearmente dipendenti dalle altre allora la sua riga diventerà composta da

soli zeri. Nel mentre che azzeriamo la colonna i-esima quindi ci possiamo rendere conto delle colonne linearmente dipendenti

Possibili soluzioni

Il primo caso che vado ad analizzare è quello in cui ci siano delle equazioni con condizioni incompatibili alle altre. Per esempio ogni sistema con una equazione con tutti i coefficienti uguali ad un'altra, ma con termini noti diversi. Tali sistemi con l'algoritmo di Gauss-Jordan generano un'equazione con uno 0 = k, con k che appartiene ai numeri reali, e quindi impossibile.

Il secondo caso che vado ad analizzare è quello in cui il risultato sia unico. Tali sistemi hanno n equazioni linearmente indipendenti e n incognite, e la matrice dei coefficienti risultante alla fine dell'algoritmo è del tipo:

$$\left[\begin{array}{c|c} B & I_n \end{array}\right]$$

Dove:

- B: sono i termini noti, cioè è un vettore colonna di n elementi
- I_n : è una matrice identità *n* x *n*, a meno di qualche scambio tra le righe

Per esempio, alla fine dell'algoritmo, potremmo trovare una matrice del tipo:

$$\left[\begin{array}{c|cc} 1 & 0 & 1 \\ 3 & 1 & 0 \end{array}\right]$$

Che significa che :

$$\begin{cases} x_1 = 3 \\ x_2 = 1 \end{cases}$$

Il terzo ed ultimo caso è se la matrice dovesse avere meno equazioni indipendenti che incognite; allora si crea una matrice identità con a destra dei termini che saranno i coefficienti delle incognite libere. Quindi la matrice risultante sarà del tipo

$$\left[\begin{array}{c|cc} B & I_n & C \end{array}\right]$$

dove:

- "B" e "I_n" sono come sopra, cioè per il caso di sistema con soluzione unica
- "C" invece è una matrice di coefficienti reali, di dimensione *n* x (*n*-*m*), dati n come numero di equazioni e m numero di incognite

Ad esempio, alla fine dell'algoritmo potremmo trovare una matrice del tipo

$$\left[\begin{array}{c|ccc} 3 & 1 & 0 & 5 \\ 4 & 0 & 1 & 2 \end{array}\right]$$

che ha come insieme delle soluzioni

$$\begin{cases} x_1 = 3 - 5 \cdot x_3 \\ x_2 = 4 - 2 \cdot x_3 \end{cases}$$

e che quindi ha le prime due variabili dipendenti dal valore della terza variabile

Analisi asintotica

Assumo le seguenti operazioni come O(1): la singola operazione elementare di divisione di un elemento della matrice per uno scalare, la sottrazione di un elemento della matrice da un altro moltiplicato per uno scalare.

Fatte queste prime assunzioni si conclude che il primo passaggio, ossia la normalizzazione del elemento di pivot sia O(m+1), dato m come numero di incognite, perché dobbiamo dividere per l'elemento di pivot gli m+1 coefficienti della matrice. Il secondo passaggio sono sempre m+1 operazioni ripetute per le n-1 equazioni rimanenti. Complessivamente questi due passaggi hanno peso computazionale O(n(m+1)). Questi due passaggi però vanno ripetuti n volte, per risolvere le n equazioni. In conclusione l'algoritmo ha un peso computazionale $O(n^2(m+1))$, dati n come numero di equazioni e m numero di incognite.

Nel programma

L'algoritmo appena descritto e analizzato nel programma viene sviluppato in questi passaggi chiave:

1. L'invocazione alla funzione solveTheMatrix ..
2. che a sua volta invoca zerosCol ..
3. che a sua volta invoca zerosRow

Ognuna di queste funzioni esegue un ciclo.

solveTheMatrix. ha il ciclo principale del programma. Il ciclo principale è :

```
while( i <( n-(M->nEDip) ) )
```

"i" è il numero di equazioni risolte; "n" è il numero di equazioni nel sistema; "nEDip" è il numero di equazioni linearmente dipendenti dalle altre, ed è aggiornato ad ogni ciclo.

Se l'elemento di pivot non è zero, allora avvio il ciclo contenuto dentro la funzione "zerosCol". che si occupa di :

1. Normalizzare l'elemento r c (indici di riga e di colonna in solveMatrix)
2. Invocare zerosRow per ogni riga diversa da r

Infine la funzione zerosRow, si occupa di azzerare l'elemento richiesto dall'algoritmo e modificare il resto della riga.

L’algoritmo per trovare le relazioni algebriche tra le equazioni

Sulla matrice delle relazioni algebriche eseguiremo passi del tutto speculari a quelli compiuti sulla matrice dei coefficienti.

Come descritto sulle strutture dati utilizzate, la matrice delle relazioni algebriche inizia con una matrice identità n x n, dato n numero di equazioni.

Nel primo passaggio dell’algoritmo, ossia la normalizzazione, l’elemento sulla diagonale i-esima, viene diviso per lo stesso elemento per il quale l’equazione i-esima viene divisa. Quindi, se per esempio, la prima equazione viene divisa per l’elemento a_{11} nella matrice dei coefficienti, allora anche l’elemento r_{11} verrà diviso per a_{11}.

$$\mathrm{MRAlg} = \begin{bmatrix} r_{11} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

$$\text{con } r_{11} = \frac{1}{a_{11}}$$

Dopo, poiché nell’algoritmo di Gauss Jordan la prima equazione viene tolta a_{22} volte dalla seconda, allora memorizzo questa operazione nella matrice MRAlg. Quindi devo togliere alla seconda riga della matrice ogni elemento della prima riga moltiplicato per k, dato k come r_{11} * a_{22}. Va moltiplicato ogni elemento della prima riga perché dalla prima equazione possono essere state tolte delle altre equazioni.

Nel programma questi calcoli vengono fatti parallelamente alla risoluzione del programma. Infatti, prima di normalizzare l’elemento di pivot all’interno della funzione zerosCol, viene fatta l’operazione di divisione in MRAlg. Inoltre, prima di eseguire zerosRow, viene eseguito factMRAlg, che è la funzione che esegue tutti i calcoli per trovare le relazioni algebriche tra le equazioni

4. Struttura e organizzazione del programma

Ho scritto il programma in ANSI C, la versione standard del linguaggio di programmazione C.

Il codice è composto da 3 file, forniti in appendice.

1. "GaussJordan.h", file header che contiene la struttura dati per la risoluzione del sistema, e gli headers delle funzioni sviluppate nel file "GaussJordan.c"
2. "GaussJordan.c", file c, che contiene tutte le funzioni relative alla risoluzione del sistema
3. "GaussJordan_Tester.c", file c, che contiene il main, e le funzioni input e output

Dopo la compilazione dei due file .c e il loro link, sarà disponibile l'eseguibile "GaussJordan_Tester.c". Per eseguire il programma sarà necessario eseguire il comando

```
GaussJordan_Tester.exe fileInput.txt fileOutput.txt
```

dove :

- "fileInput.txt" è il file di testo in ingresso con tutte le informazioni relative al sistema di equazioni di seguito descritto
- "fileOutput.txt" è il nome del file che verrà creato con la soluzione del sistema lineare.

Descrizione del file di input

Il file di input dovrà contenere 4 informazioni principali, in quest'ordine:

1. Il numero di equazioni
2. Il numero di incognite
3. La precisone con la quale intendiamo eseguire i calcoli
4. Il sistema di equazioni:

Dato un sistema di equazioni scritto in questo modo:

$$S : \begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ..... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases} .$$

Il sul file di input andrà scritto in questo modo:

$$\begin{array}{ccccc} b_1 & a_{11} & a_{12} & ... & a_{1m} \\ b_2 & a_{21} & a_{22} & ... & a_{2m} \\ ... & ... & ... & ... & ... \\ b_n & a_{n1} & a_{n2} & ... & a_{nm} \end{array}$$

Descrizione del file di Output

Il file di output invece mostrerà il risultato del sistema.

Se il sistema ha una soluzione unica verrà indicato :

```
SISTEMA CON RISULTATO UNICO
```

```
x1 = …
x2 = …
…
```

Se il sistema non avrà soluzione, verrà semplicemente stampato sul file:

```
SISTEMA IMPOSSIBILE
```

Se infine il sistema avrà infinite soluzioni, un esempio di output potrebbe essere :

```
SISTEMA INDETERMINATO
x1 = 1 + 2 * x3
x2 = 4 + 4 * x3
```

Trovando quindi l'insieme delle soluzioni.
Il programma, dopo aver stampato la soluzione del sistema, stampa anche le informazioni relative alle relazioni algebriche tra le equazioni che ha trovato. Un esempio di output potrebbe essere:

```
Il sistema ha 2 equazioni linearmente dipendenti: ( Eq5 Eq6 )

Le relazione tra le equazioni linearmente indipendenti (inizio a contare da 1):

R5= 0.11 * Eqn1  -0.67 * Eqn3
R6=-2.00 * Eqn2  -9.00 * Eqn4
```

5. Fasi del programma
1. Avvio e memorizzazione

Dopo l'avvio del programma viene creata la variabile strutturata
Matrix M

che viene inizializzata con la funzione initMatrix, contenuta nel file GaussJordan.c.

Successivamente vengono salvati i coefficienti e i termini noti del sistema con la funzione readFileMatrix, che prende in input il nome del file di input e un puntatore alla variabile strutturata M.

2. Risoluzione del sistema

Come visto nella descrizione dell'algoritmo, viene invocata la funzione solveMatrix, la quale chiamerà tutte le altre funzioni per la risoluzione del sistema

3. Stampa del risultato

Quando il sistema viene risolto dalle funzioni in GaussJordan.c, la matrice M viene passata alla funzione printFileMatrix, la quale si occupa di stampare il risultato del sistema, scegliendo anche le funzioni adeguate per il tipo di soluzione.
Se il sistema non dovesse avere una soluzione, stamperebbe questa informazione sul file.
Se il sistema dovesse avere una soluzione unica, allora chiamerebbe fprintSolUnic.
Se il sistema è indeterminato, chiama la funzione fprintIndet.
Infine se il programma trova delle equazioni linearmente dipendenti dalle altre, allora invoca la funzione fprintRel, che si occupa di stampare le relazioni tra le equazioni.

6. Osservazioni finali

Dalla descrizione dell'algoritmo di Gauss Jordan, va inoltre detto che può essere utilizzato anche per trovare l'inversa della matrice dei coefficienti, cosa che il mio programma non implementa per cercare di risparmiare operazioni da fare, ma che in un futuro questa funzione potrebbe essere implementata.

Un'altra osservazione da fare è che anche la prima riga della matrice MCoeff viene riservata ma non utilizzata e quindi potrebbe essere uno spreco di spazio; questo spreco però è trascurabile, perché comunque non viene riservata troppa memoria, in più rende più facile l'implementazione futura della risoluzione di problemi di programmazione lineare.

Oltre a queste osservazioni c'è da dire che, per risparmiare spazio in memoria e velocizzare le operazioni, i coefficienti delle matrici vengono salvati e modificati come double, questa scelta va a discapito della precisione del risultato. Molto tempo è stato dedicato anche al controllo dei risultati, e testando sistemi anche molto grandi, come per esempio 3000 equazioni x 3000 incognite e il programma riesce a risolvere il sistema con una precisone e velocità accettabili.

Bibliografia

- Bottacin, Francesco. Algebra lineare e geometria. Esculapio, 2011.

Sitografia

- https://www.mathworks.com/help/matlab/ref/linsolve.html

Appendice

File: GaussJordan.c

```
/*  Autore   : Alessandro Pisent
    Matricola: 1162916
    File     : GaussJordan.c

    Descrizione file:
    File che contiene i principali metodi per la risoluzione del sistema
lineare
*/
#include <stdlib.h>         /*malloc, free*/
#include <assert.h>         /*assert*/
#include <stdio.h>          /*printf*/
#include <math.h>           /*fabs*/
#include <string.h>         /*strcat*/
#include "GaussJordan.h"    /*definizione delle funzioni nel file*/

/*inizializzazione della matrice e allocazione della memoria
    IP n, numero di righe
    IP m, numero di colonne
    IOP A, matrice inizializzata */
void initMatrix(int n, int m, Matrix *M){
    int i;
    int initNDipRow = n-1; /*al massimo le righe indipendenti saranno il #
di nEq-1*/
    /*Assegnazioni delle capacita' di righe e colonne*/
    M->nEq = n;   /*Equazioni / Righe*/
    M->nIn = m;   /*Variabili / Colonne*/
    /*Creazione del array di riga*/
    M->MCoef = (double**) malloc( sizeof(double*) *(M->nIn + 1) );
    assert(M->MCoef != NULL);
    /*Creazione di un array per ogni riga*/
    for(i = 0; i<(n+1) ; i++){
        (M->MCoef)[i] = (double*) calloc( m+1 ,sizeof(double));
        assert((M->MCoef)[i] != NULL);
    }/*for*/
    /*Creazione del array con le righe dipendenti*/
    M->nEDip = 0; /*Array inizialmente vuoto*/
    M->aEDip = (int*) malloc(sizeof(int) * initNDipRow);
    assert(M->aEDip != NULL);
    /*creazione della matrice per le relazioni tra le righe*/
    M->MRAlg = (double**) malloc( sizeof(double*) * n );
    assert(M->MRAlg != NULL);
    /*Creazione di un array per ogni riga*/
    for(i = 0; i< n ; i++){
        (M->MRAlg)[i] = (double*) calloc( n , sizeof(double));
        assert((M->MRAlg)[i] != NULL);
    }/*for*/
    /*inizializzazzione della matrice relazioni*/
    oneMatrixRAlg(M);
}/*initMatrix*/

/*   Funzione che crea una matrice di 1 di dimensione $nEq x $nEq nella
matrice MRAlg
    IOP Matrix M, strutturato con dentro le equazioni */
void oneMatrixRAlg(Matrix *M){
```

```
    int i;
    for(i=0;i<M->nEq;i++)
        (M->MRAlg)[i][i]=1;
}/*oneMatrixRAlg*/

/*  Funzione che libera la memoria assegnata alla matrice
    IOP M, matrice da liberare
*/
void freeMatrix(Matrix *M){
    int i;          /*per il for*/
    int n = M->nEq; /*per chiarezza di lettura*/
    /*libera le righe coefficienti*/
    for(i = 0; i < (n+1); i++)
        free((M->MCoef)[(i+1)%(n+1)]);
    /*libera righe relazioni*/
    for ( i = 0; i < n; i++)
        free((M->MRAlg)[i]);
    free(M->MCoef);
    free(M->MRAlg);
    /*libera la matrice con righe dipendenti*/
    free(M->aEDip);
}/*freeMatrix*/

/*  Funzione che stampa la matrice a schermo
    IP M, matrice da stampare
    OV matrice stampata */
void printFMatrix(const Matrix *M){
    int i,j;
    /*forntespizio*/
    printf("\nSTAMPO A: b|A\n");
    /*scandisco tutte le righe*/
    for(i=0;i<M->nEq;i++){
        /*scandisco tutte le colonne*/
        for(j = 0; j<(M->nIn)+1;j++)
            printf("%5.2f ",(M->MCoef)[i+1][j]);
        printf("\n");   /*fine riga*/
    }/*for*/
}/*printFMatrix*/

/*     Funzione   che   stampa   la   matrice   Relazioni   Algebriche   tra   le
righe/Equazioni
    a schermo
    IP M, matrice da stampare
    OV matrice stampata
*/
void printFMatrixRAlg(const Matrix *M){
    int i,j;
    /*frontespizio*/
    printf("\nSTAMPO MRAlg:\n");
    /*scandisco tutte le righe*/
    for(i=0;i<M->nEq;i++){
        /*scandisco tutte le colonne*/
        for(j = 0; j<M->nEq;j++)
            printf("%5.2f ",(M->MRAlg)[i][j]);
        printf("\n");   /*fine riga*/
    }/*for*/
}/*printFMatrixRAlg*/
```

```
/*  Funzione per normalizzare l'elemento sulla "diagonale" di una riga,
    dividendola per se stessa
    IP r, riga della matrice su cui operare
    IP c, colonna della matrice su cui operare
    IOP M, matrice da modificare
    OR { 0 nessun errore
         -1 errore di normalizzazzione     }*/
int diagNorm(int r, int c, Matrix *M){
    int i;
    /*prendo il valore sulla diagonale*/
    double t = (M->MCoef)[r][c];
    /*nomalizziamo i valori sulla riga*/
    for(i=0;i<(M->nIn+1);i++)
        (M->MCoef)[r][i]=((M->MCoef)[r][i])/t;
    /*se sono riuscito a normalizzare l'elemento: nessun errore*/
    if(isZero((M->MCoef)[r][c]-1,M->error))
        return 0;
    /*se non ho normalizzato l'elemento:errore*/
    else
        return -1;
}/*diagNorm*/

/*  Funzione che azzera l'elemento M[$r][$c] e modifica di conseguenza la
    riga $r. (prende come elemento di pivot M[$rowC][$c])
    Qui c'e' il controllo se la riga e' linearmente dipendente, nel caso,
    non prova ad modificarla tutta

    IP r, riga da modificare
    IP c, colonna dove ci troviamo (Da azzerare)
    IP rowC, riga che sto prendendo in considerazione
    IOP M, Matrice da modificare */
void zerosRow(int r, int c, int rowC, Matrix *M){
    int i;
    double el;
    int z=0;
    /*Prendo il valore del elemento M[$r][$c] che devo azzerare*/
    double t = (M->MCoef)[r][c];
    /*Azzero il valore della riga $r e modifico gli altri di conseguenza*/
    for(i=0;i<(M->nIn+1);i++){
        el = (M->MCoef)[r][i] - t * (M->MCoef)[rowC][i];
        (M->MCoef)[r][i] = el;
        /*conto gli zeri che trovo*/
        if(isZero(el,M->error))
            z++;
    }/*for*/
    /*se ho una riga di zeri allora e' una riga lin dip*/
    if(z>=(M->nIn))
        addR(r,M);
}/*zerosRow*/

/*  Funzione che scrive le relazioni tra equazioni nella MRAlg
    IP r, riga in cui siamo
    IP c, colonna in cui siamo (da azzerar)
    IP rowC, riga in cui la colonna $c e' unitaria
    IOP M, struttura con tutte le informazioni necessarie
*/
void factMRAlg(int r,int c, int rowC, Matrix *M){
    int i;                          /*intero per il for*/
```

```
    double el,                      /*# di volte che l'eqn $rowC viene tolta
da $r*/
            t  =  (M->MCoef)[r][c];      /*#  di  volte  che  l'eqn  $rowC
normalizzata viene tolta da $r*/
    /*salvo e memorizzo*/
    el = t*((M->MRAlg)[rowC-1][rowC-1]);
    (M->MRAlg)[r-1][rowC-1] = el;
    /*copia delle operazioni fatte sulla eqn $r */
    for(i=0;i<M->nEq;i++){
        /*non deve modificare i dati sulla diagonale*/
        if( (i!=(r-1)) && (i!=(rowC-1)) )
            (M->MRAlg)[r-1][i]  =  (M->MRAlg)[r-1][i]  -  el*((M->MRAlg)[rowC-
1][i]);
    }/*for*/
}/*factMRAlg*/

/*  Funzione che normalizza l'elemento M[$r][$c] sulla diagonale e
    azzera gli altri elmenti tramite operazioni lineari
    IP c, colonna della matrice ideale da modificare
    IP r, riga della matrice ideale da modificare
    IOP M, Matrice da modificare
    OR { 0: nessun errore
        -1: errore di normalizzazzione      }*/
int zerosCol(int r, int c, Matrix * M){
    int i,row;  /*variabili per il ciclo, e per la riga*/
    /*prendo il valore sulla diagonale*/
    double t = (M->MCoef)[r][c];
    /*Normalizzo l'elemento sulla sua stessa riga*/
    (M->MRAlg)[r-1][r-1]=(M->MRAlg)[r-1][r-1]/t;
    /*normalizzo l'elemento M[$r][$c], inizialmente e' una diagonale*/
    if(diagNorm(r,c,M)==-1)
        return -1;
    /*Per ogni riga*/
    for(i=0;i<((M->nEq)-1);i++){
        /*indice di riga a partire da 1*/
        row = (i+r)%(M->nEq)+1;
        /*se non e' un'equazione lin dip posso dividere*/
        if(!isEqLinDip(row,M)){
            /*prima mi scrivo le operazioni che faro' sulla matrice $MRAlg*/
            factMRAlg(row, c, r, M);
            /*poi azzero la riga $row, considerando che ho normalizzato
                l'elemento M[$r][$c]*/
            zerosRow(row, c, r, M);
        }/*if*/
    }/*for*/
    return 0;
}/*zerosCol*/

/*  Funzione che risolve la Matrice M tramite il metodo di Gauss-Jordan
    IOP M, matrice da risolvere
    OR { 0: sono riuscito a risolvere il sistema
        -1: errori di normalizzazzione      }*/
int solveTheMatrix(Matrix *M){
    /*per il while*/
    int i=0,        /*contatore per il numero di incognite risolte*/
        r=1,        /*indice di riga*/
        c=1,        /*indice di colonna*/
        j=0,        /*contatore per il numero di iterazioni del while*/
```

```
        k=0,        /*indice per $(M->aEDip)*/
        iCSkip = 0; /*contatore delle colonne saltate*/
    bool isToBeSkippedR = false, /*stato per il ciclo, se la riga e' stata
saltata*/
         isToBeSkippedC = false; /*stato per il ciclo, se la colonna e'
stata saltata*/
    int n = M->nEq; /*per semplificare la lettura*/
    /*conferma che sta iniziando la risoluzione se ci sono tanti dati*/
    if(((M->nIn) * (M->nEq)) > MAX_STAMPA)
        printf("\nRISOLUZIONE DEL SISTEMA\n");
    /*Devo trovare $i =  #equazioni - #righe dipendenti*/
    /*i parte da 0, e deve avere al massimo $n-$nEDip*/
    while(i<(n-(M->nEDip))){
        /*inizializzo gli stati*/
        isToBeSkippedR = false;
        isToBeSkippedC = false;
        /*se le sono su una eqn lin dip dalle altre allora posso saltare la
riga $r ma
         non devo far avanzare la colonna $c*/
        if(M->nEDip > 0 &&  isEqLinDip(r,M)){
            isToBeSkippedC = true;
            iCSkip++;
        }/*if*/
        /*se   l'elemento   sulla   diagonale   M[$r][$c]!=0   allora   posso
continuare*/
        else if(!isZero((M->MCoef)[r][c], M->error)){
            /*normalizzo l'elemento [$r+1][$c] e azzero la colonna $c*/
            if(zerosCol(r,c,M)==-1)
                return -1;
            i++; /*lo faccio su almeno tutte le equazioni - # eq lin dip*/
        }/*if*/
        /*altimenti, l'elemento M[$r][$c]==0, ma l'equazione $r non e' lin
dip*/
        else
            isToBeSkippedR = true; /*quindi significa che non devo aumentare
$r ma provero'
                                   l'elemento M[$r][$c+1]*/
        /*indice che conta le volte che il ciclo viene eseguito*/
        j++; /*parte da 0*/
        /*aumento $r*/
        if(!isToBeSkippedR)
            /*indice per la riga, parte da 1*/
            r=((r)%(M->nEq)) +1;
        /*aumento $c*/
        if(!isToBeSkippedC){
            if((j-iCSkip)<(M->nIn))
                c++;
            else
                c = (M->aEDip)[k++];
        }/*if*/
    }/*while*/
    return 0;
}/*solveTheMatrix*/

/*  Funzione che stabilisce se la riga $r e' linearmente dipendente
    alle altre righe della matrice
    IP row, riga da controllorare
    IP M, matrice
```

```
    OR se la $row, e' linearmente dipendente alle altre righe*/
bool isEqLinDip(int row, const Matrix * M){
    int i;  /*per il for*/
    /*controllo tutto l'array con le righe lin dip*/
    for(i =0; i<(M->nEDip);i++){
        /*match trovato*/
        if(row==((M->aEDip)[i]))
            return true;
    }/*for*/
    /*match non trovato*/
    return false;
}/*isEqLinDip*/

/*  Funzione che dice se un double e' zero dato un errore
    IP a, valore da controllare se sia circa zero
    IP precisone
    OR $a==0 */
bool isZero(double a, double precisione){
    return fabs(a)<precisione;
}/*isZero*/

/*  Funzione aggiunge la riga all'array di righe indipendenti
    IP r, riga da aggiungere
    IOP M, matrice con tutto*/
void addR(int r, Matrix* M){
    M->aEDip[M->nEDip] = r;
    M->nEDip +=1;
}/*addR*/

/*  Fuznione che stampa a schermo il sistema di equazioni a schermo
    IP Matrix M
    OV il sistema di equazioni */
void printEquations(const Matrix *M){
    int i,j;
    /*stampa del forntespizio*/
    printf("\nIL SISTEMA DI EQUAZIONI e':\n");
    for(i = 0; i < M->nEq; i++){
        for(j=0;j<M->nIn-1;j++)
            printf("%5.2f * x%d + ",(M->MCoef)[i+1][j+1],j+1);
        /*stampa l'ultimo elemento*/
        printf("%5.2f * x%d ",(M->MCoef)[i+1][j+1],j+1);
        /*stampa termine noto*/
        printf(" = %5.2f\n",(M->MCoef)[i+1][0]);
    }/*for*/
}/*printEquations*/

/*  Funzione che controlla che il sistema non sia impossibile
    i.e. controlla che i coefficienti noti di tutte le eqn lin dip siano
zero
    IP M, matrice con tutti i dati
    OR se il sistema rappresentato dalla matrice M, e' risolvibile
*/
bool isZeroCoefAllEqnLinDip(const Matrix* M){
    int i,r;
    for(i=0;i<M->nEDip;i++){
        r = M->aEDip[i];
        if(!isZero(M->MCoef[r][0],M->error))
            return false;
```

```
    }/*for*/
    return true;
}/*isZeroCoefAllEqnLinDip*/


#ifdef TEST
/*  Funzione per testare che effettivamente la relazione trovata
    Sia giusta
    IP S, matrice risolta
    IP T, matrice non risolta, "Originale"
    IP printTest, se vogliamo stampare a schermo le operazioni compiute dal
test
    OR : TRUE se il test e' andato a buon fine (ho trovato la vera
relazione)
         FALSE se il test non e' andato a buon fine
               (La relazione trovata nella matrice MRAlg non e' corretta)*/
bool test(Matrix* S, Matrix* T){
    int  d, j, i, iR;
    double molt, elDip, elT, sum;
    /*Se troppi elementi per stampare*/
    if(S->nIn * S->nEq > MAX_STAMPA){
        printf("\nEQUAZIONE TROPPO GRANDE DA STAMPARE\n");
    }/*if*/
    /*altrimenti stampa a schermo le matrici*/
    else{
        /*Stampa dei calcoli fatti*/
        printf("\nORIGINAL");
        printFMatrix(T);
        printf("\nSOLVED");
        printFMatrix(S);
        printf("\nRelazioni algebriche:");
        printFMatrixRAlg(S);
    }/*else*/
    /*se non ci sono relazioni allora e' automaticamente passato*/
    if(S->nEDip < 1)
        return true;
    /*Stampa di quali righe sono da controllare*/
    printf("Dobbiamo controllare R = ");
    for(i=0;i<(S->nEDip)-1;i++)
        printf("%d, ",S->aEDip[i]);
    printf("%d",S->aEDip[i]);
    printf("\n");
    /*per ogni riga dipendente*/
    for(d=0;d<S->nEDip;d++){
        /*indice della riga che stiamo controllando*/
        iR = (S->aEDip)[d];
        for(i=0;i<S->nIn+1;i++){
            /*la mia ipotesi e' che l'elemnto S[$iR][$i] sia la somma di
            $nEq-1 elementi, alcuni con il fattore moltiplicativo zero*/
            sum = 0;
            for(j=0;j<S->nEq;j++){
                /*fattore moltiplicativo che trovo dul*/
                molt = (S->MRAlg)[iR-1][j];
                /*controllo che non siamo sulle diagonali*/
                if(iR!=(j+1) && (!isZero(molt,S->error))){
                    elDip = (T->MCoef)[j+1][i];
                    sum += molt*elDip;
                }/*if*/
```

```
            }/*for*/
            /*elemento da controllare*/
            elT = (T->MCoef)[iR][i];
            /*controllo che la differenza sia Zero*/
            if(!isZero(elT-sum,S->error))
                return false;
        }/*for*/
        return true;
    }/*for*/
    return true;
}/*test*/

/*  Funzione che controlla che la colonna $col sia composta da un solo
        elemento diverso da zero
    IP col, indice di colonna {1 -> $(M->nIn)}
    IP M, Matrice con i coefficienti
*/
bool isColSolved(int col, const Matrix * M){
    int i,          /*per il for*/
        nOne = 0;   /**/
    /*controllo tutte le colonne*/
    for(i=0;i<M->nEq;i++){
        /*se l'elemento sulla colonna !=0*/
        if(!isZero((M->MCoef)[i+1][col],M->error)){
            /*se trovo solo un elemento != 0*/
            if(nOne==0)
                nOne++;
            /*ne ho trovati 2*/
            else
                return false;
        }/*if*/
    }/*for*/
    return true;
}/*isColSolved*/

/*  Funzione che scrive su $aC le colonne di $M non risolte
        (ossia con un solo elemento != 0 , e tutti gli altri 0)
    IOP aC, array da riempire
    IP M, Matrice con i coefficienti delle equazioni
*/
int whichColAreNotSolved(int* aC, const Matrix * M){
    int i,      /*indice del for*/
        iA=0;   /*indice array*/
    /*per ogni colonna controllo che sia risolta*/
    for(i=0;i< M->nIn; i++){
        /*se non e' risolta la salvo nel array delle non risolte*/
        if(!isColSolved(i+1,M))
            aC[iA++]=i+1;
    }/*for*/
    return iA;
}/*whichColAreNotSolved*/

/*  Fprintf Which Coloms are Not Solved
    Funzione che stampa su file quali colonne non sono risole
    IP Matrice da controllare
    OF file con le colonne non risolte
*/
void FprintFCRNS(const Matrix * M){
```

```
    int * aC; /*array con le colonne*/
    int iA,   /*indice per array*/
        i;    /*indice per for*/
    FILE *oF; /*puntatore a file su cui scrivere*/
    /*creazione del array delle incognite non risolte*/
    aC = malloc(sizeof(int) * M->nIn);
    assert(aC != NULL);
    /*chiamata alla funzione che riempie l'array di eqn non risolte*/
    iA = whichColAreNotSolved(aC, M);
    /*apertura del file*/
    oF = fopen("notSolvedRows.txt","w");
    if(oF == NULL)
        return;
    /*Frontespizio*/
    fprintf(oF,"Righe sbagliate:%d \n",iA);
    /*for che cicla tutte quelle non risolte e le stampa su file*/
    for(i = 0; i<iA; i++)
        fprintf(oF,"-%3d\n",aC[i]);
    fclose(oF);     /*chiusura del file*/
    free(aC);       /*eliminazione array*/
}/*FprintFCRNS*/
#endif
```

File: GaussJordan.h

```
#include <stdbool.h> /*bool ovunque*/
#define FREE_ROWS 1
#define COLONNA_TERMINE_NOTO 0
#define MAX_STRING 100000
#define MAX_STAMPA 1000
#define MAX_STAMPA_EQN 100
/*#define TEST*/
typedef struct {
    double** MCoef;     /*Matrice con allocazione dinamica memoria*/
    int nEq; /*n*/           /*# di equazioni, # Righe*/
    int nIn; /*m*/           /*# di Incognite, # Colonne*/
    int* aEDip;            /*array con gli indici delle equazioni dipendenti
*/
    int nEDip;            /*# di elementi in $dipRow*/
    double** MRAlg;     /*matrice con relazioni algebriche tra righe*/
    double error;       /*precisione di errore*/
} Matrix;
/*funzioni di inizializzazzione e liberazione*/
void initMatrix(int , int , Matrix *);
void oneMatrixRAlg(Matrix *);
void freeMatrix(Matrix * );
/*funzioni di stampa video delle matrici/sistema*/
void printFMatrix(const Matrix *);
void printFMatrixRAlg(const Matrix *);
void printEquations(const Matrix *);
/*funzioni per la risoluzione del sistema*/
int diagNorm(int , int, Matrix *);
void zerosRow(int , int, int, Matrix *);
void factMRAlg(int, int, int, Matrix *);
int zerosCol(int ,int,  Matrix *);
bool isEqLinDip(int, const Matrix *);
int solveTheMatrix(Matrix *);
/*funzioni accessorie*/
bool isZero(double , double);
void addR(int , Matrix *);
int min(int, int);
bool isZeroCoefAllEqnLinDip(const Matrix *);
/*funzioni di test*/
#ifdef TEST
    bool test(Matrix* , Matrix* );
    bool isColSolved(int, const Matrix *);
    int whichColAreNotSolved(int* , const Matrix * );
    void FprintFCRNS (const Matrix *);
#endif
```

File: GaussJordan_Tester.c

```
/*  Autore   : Alessandro Pisent
    Matricola: 1162916

    Descrizione file :
    In questo file di tester ci sono tutte le funzioni per la gestione dei
file
    e il main per per il risolutore lineare
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "GaussJordan.h"

#define HELP_STRING "-help"
#define TEST_STRING "-test"
#define MATLAB_FILE "results.txt"

/*  Funzione che stampa su file il sistema di equazioni
    IP nameFileOut, stringa con il nome del file da aprire
    OF outF, file dove stampare
    IP Matrice con il sistema di equazioni*/
void fprintEquazioni(const char nameFileOut[],const Matrix *M){
    FILE *outF;     /*Variabile per il file di Output*/
    int i,j;
    /*apertura del file*/
    outF = fopen(nameFileOut, "a");
    /*Errore apertura file output*/
    if (outF == NULL){
        printf("ERRORE APERTURA IN PRINT_EQN\n");
        return;
    }/*if*/
    /*scritta per la matrice risolta*/
    fprintf(outF,"Il sistema risolto:\n\n");
    /*Scansione delle righe*/
    for(i = 0; i < M->nEq; i++){
        fprintf(outF,"%3d:",i+1);
        /*stampa delle equazioni*/
        for(j=0;j<M->nIn-1;j++)
            fprintf(outF,"%5.2f * x%d + ",(M->MCoef)[i+1][j+1],j+1);
        /*stampa l'ultimo elemento*/
        fprintf(outF,"%5.2f * x%d ",(M->MCoef)[i+1][j+1],j+1);
        /*stampa termine noto*/
        fprintf(outF," = %5.2f\n",(M->MCoef)[i+1][0]);
    }/*for*/
    /*chiusura del file*/
    fclose(outF);
}/*fprintEquazioni*/

/*  Funzione che stampa la soluzione quando il sistema ha un unica soluzione
    IP nome del file
    OF outF, dove stampare il risultato
    IP Matrice Risolta*/
void fprintSolUnic(const char nameFileOut[], const Matrix *M){
    int i,j;
    FILE *outF;     /*Variabile per il file di Output*/
```

```
    #ifdef TEST
        /*Creazione e apertura per il file matlab*/
        FILE *matlabF;
        matlabF = fopen(MATLAB_FILE,"w");
        /*Errore apertura file output*/
        if (matlabF == NULL){
            printf("ERRORE APERTURA IN PRINT_SOL_UNICA(MatlabFile)\n");
            return;
        }/*if*/
    #endif
    outF = fopen(nameFileOut, "a"); /*append*/
    /*Errore apertura file output*/
    if (outF == NULL){
        printf("ERRORE APERTURA IN PRINT_SOL_UNICA\n");
        return;
    }/*if*/
    /*scrittura frontespizio*/
    fprintf(outF,"SISTEMA CON RISULTATO UNICO\n");
    /*scrittura piu' carina del risultato*/
    for(j=0;j<M->nIn;j++){
        for(i=0;i<M->nEq;i++){
            /*stampa tutti i coefficienti che sono 1*/
            if(isZero((M->MCoef)[i+1][j+1]-1,M->error)){
                fprintf(outF,"x%2d = %f\n",j+1,(M->MCoef)[i+1][0]);
                #ifdef TEST
                    /*stampo solo se richiesto dalle direttive*/
                    fprintf(matlabF,"%f\n",(M->MCoef)[i+1][0]);
                #endif
                /*ho trovato match vai a riga successiva*/
                break;
            }/*if*/
        }/*for*/
    }/*for*/
    /*chiusura del file*/
    fclose(outF);
    #ifdef TEST
        /*chiusura del file*/
        printf("Scritto file matlab");
        fclose(matlabF);
    #endif
}/*fprintSolUnic*/

/*  Funzione che stampa su file l'insieme di soluzioni per un sistema
    indeterminato
    IP nome del file da aprire
    OF outF, file di input aperto in precedenza, dove stampare
    IP Matrix M, matrice  con i coefficienti delle equazioni lineari*/
void fprintIndet(const char nameFileOut[], const Matrix *M){
    int i=0,    /*indice di riga*/
        j=0,    /*indice di colonna*/
        k,      /*indice per scandire tutti gli elementi di una riga*/
        nP=0,   /*incide per contare quante righe ho stampato*/
        nToPrint = (M->nEq-M->nEDip); /*# di eqn da stampare*/
    double el ; /*variabile per gli elementi da salvaew*/
    FILE *outF;     /*Variabile per il file di Output*/
    outF = fopen(nameFileOut, "a");
    /*Errore apertura file output*/
    if (outF == NULL){
```

```
        printf("ERRORE APERTURA IN PRINT_INDET\n");
        return;
    }/*if*/
    /*frontespizio*/
    fprintf(outF,"\nSISTEMA INDETERMINATO\n");
    /*devo stampare esattamente $nToPrint volte, la seconda condizione
        e' per uscire nel caso di while infinito*/
    while((nP<nToPrint)&&(i<M->nEq)){
        /*ho trovato un 1, cioe' una incognita che sono riusicito a
normalizzare*/
        if(isZero((M->MCoef)[i+1][j+1]-1,M->error)){
            /*stampa dell'inizio con il termine noto*/
            fprintf(outF,"x%2d = %5.2f ",j+1,(M->MCoef)[i+1][0]);
            /*stampa dei coefficienti delle incognite libere*/
            for(k=0 ;k<M->nIn;k++){
                /*mi salvo l'elemento*/
                el = (M->MCoef)[i+1][k+1];
                /*se ha senso stampare il coefficiente*/
                if(!isZero(el,M->error) && (k!=j)){
                    /*stampa a seconda se il valore sia positivo o
negativo*/
                    /*cambio di segno perche' e' come se avessi portato al
di la'
                        del uguale*/
                    if(el>0)
                        fprintf(outF,"- %5.2f * x%d ",el, k+1);
                    else if(el <0)
                        fprintf(outF,"+ %5.2f * x%d ",-1 * el, k+1);
                }/*if*/
            }/*for*/
            /*finita la riga*/
            fprintf(outF,"\n");
            nP++; /*ho stampato una riga, quindi aggiorno il contatore*/
            j=0;  /*ricomincio alla prima incognita*/
            i++;  /*vado alla riga successiva*/
        }/*if*/
        /*continuo con il ciclo*/
        else if(j<M->nIn)
            j++; /*provo con la colonna dopo*/
            /*no match sulla riga*/
        else if(j==M->nIn){
            /*potrebbe essere una riga con tutti zeri*/
            j=0; /*ricomincio della prima colonna*/
            i++; /*vado alla riga successiva*/
        }/*else if*/
    }/*while*/
    /*chiusura del file*/
    fclose(outF);
}/*fprintIndet*/

/*  Funzione che stampa le relazioni tra le righe linearmente dipendenti
    IP nome del file
    OF File, precedentemente aperto, dove stampare le relazioni
    IP struttura con dentro le informazioni delle relazioni tra le
Equzioni*/
void fprintRel(const char nameFileOut[], const Matrix *M){
    int i,j,row,count=0;
    double el;
```

```
    FILE *outF;     /*Variabile per il file di Output*/
    outF = fopen(nameFileOut, "a");
    /*Errore apertura file output*/
    if (outF == NULL){
        printf("ERRORE APERTURA IN PRINT_REL\n");
        return;
    }/*if*/
    /*considerazione per il singolare e plurale*/
    if(M->nEDip==1)
        /*frontespizio*/
        fprintf(outF,"\nIl sistema ha una equazione linearmente dipendente:
");
    else
        /*frontespizio*/
        fprintf(outF,"\nIl sistema ha %d equazioni linearmente dipendenti:
",M->nEDip);
    /*stampa delle eqn lin dip*/
    fprintf(outF,"(");
    for(i=0;i<M->nEDip;i++)
        fprintf(outF," Eq%d", (M->aEDip)[i]);
    fprintf(outF," )");
    /*frontespizio*/
    fprintf(outF,"\n\nLe relazione tra le equazioni linearmente indipendenti
(inizio a contare da 1):\n\n");
    /*stampa delle relazioni*/
    for(i=0;i<(M->nEDip);i++){
        /*indice di riga*/
        row=M->aEDip[i];
        fprintf(outF,"R%d=",row);
        count = 0;
        for(j=0;j<M->nEq;j++){
            el = (M->MRAlg)[row-1][j];
            /*stampo solo le relazioni con le altre colonne e diverse da 0*/
            if((j!=(row-1)) && !isZero(el,M->error)){
                if(!count){
                    fprintf(outF,"%5.2f * Eqn%d ",el,j+1);
                    count++;
                }/*if*/
                else
                    if(el>0)
                        fprintf(outF,"+ %5.2f * Eqn%d ",el,j+1);
                    else
                        fprintf(outF," %5.2f * Eqn%d ",el,j+1);
            }/*if*/
        }/*for*/
        fprintf(outF,"\n");     /*fine della riga*/
    }/*for*/
    fclose(outF);
}/*fprintRel*/

/*  Funzione che in base al errore stampa un tipo di aiuto
    IP code : (0,1) = (basic,error file input)
    OV aiuto */
void printHelp(int code){
    /*se il codice e' zero significa che devo stampare
    l'inizio del programma*/
    if(code==0){
```

```
        printf("usage: GaussJordan_Tester.exe FileInput.txt FileOutput.txt
");

        /*se test viene anche spiegata l'opzione di test*/
        #ifdef TEST
            printf("<%s>\n\n",TEST_STRING);
            printf("\"%s\" e' opzionale: serve per confermare che la
relazione tra le righe sia corretta",TEST_STRING);
        #endif
        printf("\n\nil file di input va formattato come :\n");
        printf("n m\n");
        printf("error\n\n");
        printf("b1 a11 a12 a13\n");
        printf("b2 a21 a22 a23\n");
        printf("b3 a31 a32 a33\n");
        printf("dove:\n- n = # di equazioni\n");
        printf("- error= precisione che consideriamo accettabile per
confrontare due numeri\n\n");
        printf("- m = # di incognite\n");
        printf("- b# = termine noto b#\n");
        printf("- a## = coefficiente della equazione\n");
        printf("Regole aggiuntive:\n");
        printf("- e' possibile non inserire tutti i coefficienti
nell'equazione, verranno considerati nulli\n");
        printf("- e' necessario inserire almeno 2 numeri in una riga (cioe'
del termine noto e il coefficiente di x1)\n");
        printf("\n");
    }/*if*/
    /*errore base*/
    else if(code==1){
        printf("Inserire il nome dei file di Input e Output\n");
        printf("Per aiuto \"%s\"\n",HELP_STRING);
        #ifdef TEST
            printf("opzione di test: %s\n",TEST_STRING);
        #endif
    }/*else if*/
}/*printHelp*/
/*   Funzione che stampa su un file la Matrice M (con i termini noti a
sinitra)
    IP nameFileOut, stringa contente il nome del file in Output
    IP Matice M, matrice da stampare
    OF File $nameFileOut con la matrice
    OR Esito: (
         0: elaborazione riuscita;
        -1: apertura fallita di $nameFileOut).*/
int printFileMatrix(const char nameFileOut[], const Matrix *M){
    FILE *outF;     /*Variabile per il file di Output*/
    outF = fopen(nameFileOut, "w");
    /*Errore apertura file output*/
    if (outF == NULL)
        return -1;
    /*se ha un risultato del tipo 0*x1 = a, il sistema e' impossibile*/
    if(!isZeroCoefAllEqnLinDip(M)){
        fprintf(outF,"\nSISTEMA IMPOSSIBILE\n");
        /*chiusura del file*/
        fclose(outF);
        return -2;
    }/*if*/
```

```
    /*chiusura del file*/
    fclose(outF);
    #ifdef TEST
        /*stampa le equazioni su file*/
        fprintEquazioni(nameFileOut,M);
    #endif
    /*Se # di equazioni - # di incognite == # di equazioni linarmente
    dipendenti il sistema e' con una soluzione unica*/
    if(M->nEq - M->nIn - M->nEDip == 0)
        fprintSolUnic(nameFileOut,M);
    /*altrimenti, posso scrivere delle relazioni tra le colonne*/
    else
        fprintIndet(nameFileOut, M);
    /*stampo la relazioni lineari tra le righe se ci sono*/
    if(M->nEDip > 0)
        /*stampa sul file le relazioni tra le equazioni*/
        fprintRel(nameFileOut, M);
    /*nessun problema*/
    return 0;
}/*printMatrix*/

/*  Funzione per la lettura dei comandi nei file
    IP nameFileIn, stringa contente il nome del file in input
    OR Esito (
         0: elaborazione riuscita;
        -1: apertura fallita di $nameFileIn;
        -2: errore di inserimento dati
*/
int readFileMatrix(const char nameFileIn[], Matrix *M){
    double r;               /* per la lettura*/
    int n;                  /*dimensione righe*/
    int m;                  /*dimensione colonne*/
    int err;
    int i=0,
        j=0;              /*icontatore elementi*/
    /*Apertura e dello stream dei file di input e output*/
    FILE *inF;
    inF = fopen(nameFileIn, "r");
    /*Errore di apertura file input*/
    if (inF == NULL)
        return -1;
    /*scansione delle dimensioni della matrice*/
    fscanf(inF,"%d %d", &n, &m);
    /*inizializzazione della matice*/
    initMatrix(n,m,M);
    /*salvo l'errore dentro alla matrice*/
    fscanf(inF,"%lf",&r);
    M->error = r;
    /*scansione della matrice*/
    for(i=0;i<n;i++){
        for(j = 0; j< m+1 ;j++){
            /*lettura del dato*/
            err=fscanf(inF,"%lf",&r);
            /*memorizzazzione*/
            (M->MCoef)[i+1][j] = r;
        }/*for*/
    }/*for*/
    if(err==-1)
```

```
        return -2;
    /*stampa di cioÃ² che ho letto, se il numero di elementi non e' troppo
grande*/
    if((n*m)<MAX_STAMPA_EQN)
        printEquations(M);
    else
        printf("Sistema di %d Equazioni x %d incognite, troppo grande da
stampare\n", n,m);
    printf("\nERRORE LETTO : %f\n", M->error);
    /*chiusura del file*/
    fclose(inF);
    /*nessun errore*/
    return 0;
}/*readFileMatrix*/

/*  Main per la lettura del file della matrice
    OR {    0  se la lettura e scrittura del file andata a buon fine
            -1 se l'utente ha dimenticato di insierire i nomi dei file
    }*/
int main(int argc, char const *argv[]){
    Matrix M;
    int err;
    #ifdef TEST
        Matrix T;
        bool doTest=false;  /*variabile booleana che dice se ho effetuato il
test*/
    #endif
    if((argc>1) && !(strcmp(argv[1],HELP_STRING))){
        printHelp(0);
        return 0;
    }/*if*/
    /*Se l'utente si e' dimenticato di scrivere i file di IO*/
    else if((argc < 3) && (argc > 4)){
        printHelp(1);
        return -1;  /*ritorno di un intero negativo per simulare un errore*/
    }/*else if*/
    /*codice errore*/
    err = readFileMatrix(argv[1],&M);
    /*Lettura della matrice in input*/
    if(err == -1){
        printf("ERRORE FILE INGRESSO\n");
        return -1;
    }/*if*/
    else if (err == -2){
        printf("ERRORE DI INSERIMENTO DATI\n");
        return -1;
    }/*if*/
    /*risoluzione della Ma trice*/
    if(solveTheMatrix(&M)==-1){
        printf("Errori di normalizzazzione\nSistema non risolto\n");
        return -1;
    }/*if*/
    else
        /*stampa a schermo che il sistema e' risolto*/
        printf("\nSistema risolto\n");
    #ifdef TEST
        /*stampa su file delle colonne non risolte*/
        FprintFCRNS(&M);
```

```
        /*se viene aggiunto alla fine la stringa per testare le relazioni*/
        if((argc==4) && !strcmp(argv[3],TEST_STRING)){
            /*faccio il test se ce' qualche eqn lin dip*/
            if(M.nEDip!=0){
                /*rileggo il file con il sistema originale, e riempio T*/
                readFileMatrix(argv[1],&T);
                /*variabile che mi dice che ho riempito T*/
                doTest = true;
                /*se il test e' passato stampo*/
                if(test(&M,&T))
                    printf("\n!TEST PASSATO \n");
                /*se il test non e' passato:*/
                else
                    printf("\nTEST NON PASSATO !!!!!!!\n");
            }/*if*/
            /*altrimenti non avvio nemmeno il test*/
            else
                printf("\nNessuna EQN LIN DIP\n");
        }/*if*/
    #endif
    /*Stampa su file della matrice risolta*/
    if(printFileMatrix(argv[2],&M) == -1){
        printf("ERRORE FILE USCITA");
        return -1;
    }/*if*/
    /*se il sistema e' di grandi dimensioni stampo a video la conferma di
scrittura*/
    if((M.nEq *  M.nIn) > MAX_STAMPA)
        printf("FILE SCRITTO\n");
    /*libero la memoria dalla matrice creata*/
    freeMatrix(&M);
    #ifdef TEST
        if(doTest)
            freeMatrix(&T);
    #endif
    return 0;
}/*main*/
```