Università degli studi di Napoli “Federico II”

Facoltà di Scienze MM.FF.NN. - Informatica

Corso di Calcolo Scientifico

Elaborato 1

Calcolo del prodotto di due matrici triangolari

e il determinante della matrice risultante

ottenuta mediante l'utilizzo dei moduli di BLAS

Ferrara Francesco Saverio

566/811

prof.ssa L. D'Amore

A.A. 2004/2005

Scopo

Assegnate le matrici $TI \in \Re^{nxn}$ e $TS \in \Re^{nxn}$, con TI matrice triangolare inferiore, e TS matrice triangolare superiore, si vuole calcolare:

- La matrice $RE \in \Re^{nxn}$, ottenuta dal prodotto di TI e TS,
- Il determinante della matrice RE.

Analisi del problema

Apparentemente questo problema sembra abbastanza costoso dal punto di vista dela sua complessità di spazio e tempo. Guardandolo meglio notiamo che le due matrici date in input dall'utente sono sempre matrici triangolari, e per la una matrice triangolare il suo determinante è dato dal prodotto di tutti gli elementi della diagonale.
Un ulteriore semplificazione la otteniamo applicando il teorema di Binet:

$$\det(RE) = \det(TI) \cdot \det(TS)$$

Siccome noi conosciamo a priori le matrici TI e TS, siamo in grado di calcolare il determinante prima di calcolare la matrice RE. Questa è una scelta ottimale perchè il calcolo del determiante di una matrice generica (come RE) è molto costoso dal punto di vista computazionale.

<u>Descrizione della strategia utilizzata:</u>
Siccome le matrici TI e TS possono essere molto grandi e non entrare completamente nella cache, si è deciso di dividere le matrici in quattro blocchi seguendo questo schema:

$$TI \equiv \begin{pmatrix} TI0 & 0 \\ TI1 & TI2 \end{pmatrix} \quad \text{e} \quad TS \equiv \begin{pmatrix} TS0 & TS1 \\ 0 & TI2 \end{pmatrix}$$

Così facendo si evita un intensa attività di paginazione tra la cache e la memoria (RAM); inoltre il blocco superiore destro di TI e il blocco inferiore sinistro di TS, essendo composti da tutti 0 non vengono proprio allocati.
La matrice risultante RE:

$$RE \equiv \begin{pmatrix} RE0 & RE1 \\ RE2 & RE3 \end{pmatrix}$$

Sarà ottenuta svolgendo i seguenti calcoli:

RE0 = TI0 * TS0
RE1 = TI0 * TS1
RE2 = TI1 * TS0
RE3 = TI1 * TS1 + TI2 * TS2

Per evitare di allocare ulteriore spazio per la matrice RE, l'algoritmo sovrascrive i blocchi TS0, TS1, TS2, e TI2 con i blocchi della matrice risultante. Le operazioni vengono svolte esattamente nel seguente ordine:

TS2 = TI1 * TS1 + TI2 * TS2
TI2 = TI1 * TS0

TS0 = TI0 * TS0
TS1 = TI0 * TS1

Il determinante sarà calcolato prima di calcolare la matrice RE in modo da sfruttare le proprietà delle matrici triangolari TI e TS.

Descrizione dell'algoritmo:
In una prima fase l'algoritmo esamina gli argomenti passati per riga di comando, in quanto se viene inserita l'opzione '-h` oppure '--help` viene stampata a video una mini guida che aiuta l'utente a inserire un inpt corretto.
Se invece il programma viene chiamato senza opzioni, allora chiede all'utente di inserire la dimensione dei blocchi delle matrici TI e TS, e in seguito legge dallo standart input il contenuto di queste matrici.
Dopo aver fatto questo vengono eseguite le seguenti istruzioni:

```
01 //Calcolo del determinante
02 det = calcolo_determinante(TI[0], TI[2], TS[0], TS[2], dim);
03
04 //Calcolo del prodotto della matrice
05 alpha = 1; //inizializzo lo scalare alpha
06 i=1; //fattore di incremento usato nelle routin saxpy() e scopy()
07 trans = 'n'; //considero le matrici non trasposte
08 diag = 'n'; //non so se la diagonale di qualche blocco e' unitaria oppure no
09 len = dim*dim; //numero di elementi per ogni blocco (usato nelle routin saxpy() e scopy())
10
11 //blocco R3 --> lo registro in TS[2]
12 // R3 <- TI[1] * TS[1] + TI[2] * TS[2]
13 side='l';
14 uplo='l';
15 strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI[2], &dim, TS[2], &dim);
16 sgemm_(&trans, &trans, &dim, &dim, &dim, &alpha, TI[1], &dim, TS[1], &dim, &alpha, TS[2], &dim);
17
18 //blocco R2 --> lo registro in TI[2]
19 // R2 <- TI[1] * TS[0]
20 scopy_(&len, TI[1], &i, TI[2], &i);
21 side='r';
22 uplo='u';
23 strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TS[0], &dim, TI[2], &dim);
24
25 //blocco R0 --> lo registro in TS[0]
26 // R0 <- TI[0] * TS[0]
27 side='l';
28 uplo='l';
29 strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI[0], &dim, TS[0], &dim);
30
31 //blocco R1 --> lo registro in TS[1]
32 // R1 <- TI[0] * TS[1]
33 strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI[0], &dim, TS[1], &dim);
34
```

Alla riga 2 viene chiamata la funzione che calcola il determinante della matrice RE. Essa contiene il seguente codice:

```
float calcolo_determinante(float *a, float *b, float *c, float *d, int dim) {
        float det=1;
        int i;

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del primo blocco
                det *= *(a+i);
                i += dim;
        }

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del secondo blocco
                det *= *(b+i);
                i += dim;
        }

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del terzo blocco
                det *= *(c+i);
                i += dim;
        }
```

```
        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del quarto blocco
                det *= *(d+i);
                i += dim;
        }

        return det;
}
```

Questa funzione puo' essere scritta utilizzando un unico ciclo, in questo modo:

```
float calcolo_determinante(float *a, float *b, float *c, float *d, int dim) {
        float det=1;
        int i;

        for (i=0 ; i<dim*dim ; i++) {
                det *= *(a+i);
                det *= *(b+i);
                det *= *(c+i);
                det *= *(d+i);
                i += dim;
        }

        Anche se questo ciclo e' molto elegante, si e' preferito dividerlo in
        quattro cicli per sfruttare al massimo la localita' dei dati.

        return det;
}
```

Anche se questo ciclo è molto elegante, si è preferito dividerlo in quattro cicli per sfruttare al massimo la località dei dati. Inoltre si può pensare di velocizzare i cicli con la tecnica del loop-unrolling, ma per rendere più portabile questo algoritmo è meglio lasciare questo compito al compilatore passandogli l'opzione '-funroll-loops`.
Dopo aver calcolato il determinante utilizzando la matrici TI e TS, l'algoritmo comincia a calcolare a matrice RE comciando dal blocco RE3 (righe 11-16). Successivamente calcola i blocchi RE2 (righe 18-23), RE0 (righe 25-29), e RE1 (righe 31-33). Notiamo che durante questo calcolo non è stato utilizzato nessun blocco temporaneo, riducendo al minimo la complessità di spazio.

Indicatori di errore

Il programma ritorna un indicatore di errore in caso di passaggio di parametri non riconosciuti. Non sono previsti ulteriori indicatori di errore.

Routine ausiliarie

Per il corretto funzionamento, il programma si avvale di diverse subroutine e function classificabili in routine interne, esterne, e appartenenti alla libreria BLAS.

Routine interne:

```
float calcolo_determinante(float *a, float *b, float *c, float *d, int dim);
```

Dati in input quattro blocchi di matrice trinagolare (a, b, c, d) e la loro dimensione (dim), questa funzione restituisce il determinante calcolato facendo il prodotto degli elementi sulla diagonale.

```
void help();
```

Questa procedura stampa a video una semplice mini-guida per aiutare l'utente a inserire i dati in input.

Routine esterne:
Sono tutte definite nel file “util.h”.

```
float *smtle(char uplo, float *a, int n);
```

Dato un puntatore nullo (a), questa funzione provvede ad allocare spazio (n*n) per una matrice triangolare superiore (uplo='u') o inferiore (uplo='l'), e la riempie leggendo i dati da tastiera. L'algoritmo prevede che la matrice sia memorizzata per colonne in modo da facilitare il passaggio dei parametri alle routine fortran.

```
void smast(float *a, int n, int m);
```

Data una matrice di dimensione n*m individuata dal puntatore 'a', e registrata in memoria per colonne, questa procedura stampa la matrice sullo standard output.

```
float *smale(float *a, int n, int m);
```

Dato un puntatore nullo (a), questa funzione provvede ad allocare spazio (n*m) per una matrice generica, e la riempie leggendo i dati da tastiera. L'algoritmo prevede che la matrice sia memorizzata per colonne in modo da facilitare il passaggio dei parametri alle routine fortran.

Routine esterne (BLAS):
Sono le routine messe a disposizione dalla libreria BLAS (Basic Linear Algebra Subprogram). Notiamo che sono state usate solamente subroutine e mai una function del pacchetto BLAS, quindi non ci aspetteremo mai un valore di ritorno.

```
void sgemm(char *transa, char *transb, int *m, int *n, int *k, float *alpha, float *a, int
*lda, float *b, int *ldb, float *beta, float *c, int *ldc);
```

Questa subroutine esegue il seguente calcolo:

$$C \Leftarrow \alpha \cdot A \cdot B + \beta \cdot C$$

In questa subroutine viene svolto il prodotto di due matrici generiche.

```
void strmm(char *side, char *uplo, char *transa, char *diag, int *m, int *n, float *alpha,
float *a, int *lda, float *b, int *ldb);
```

Questa subroutine esegue il seguente calcolo:

$$B \Leftarrow \alpha \cdot A \cdot B$$

Essa tiene conto del fatto che una delle due matrici è di tipo ditrangolare. Questa è una limitazione delle routine BLAS che sfruttano la struttura particolare di una sola delle due matrici considerando l'altra una matrice generica.

```
void scopy(int *n, float *x, int *incx, float *y, int *incy);
```

Questa subroutine esegue il seguente calcolo:

$$y \Leftarrow x$$

Essa non fa altro che copiare un vettore in un altro. Siccome le matrici in fortran non sono altro che lunghi vettori, usiamo questa funzone per la copia delle matrici.

Complessità

L'algoritmo è stato pensato per fare calcoli con matrici di grandi dimensioni ottimizzando la complessità di spazio in quanto il pacchetto BLAS non prevede la memorizzazione “Packed Storage” per le matrici triangolari.

Complessità di tempo:
Siccome vengono utilizzate sostanzialmente routine di BLAS 3, abbiamo che la loro complessità è dell'ordine di circa n^3 . Tutte le altre routine presenti hanno una complessità di tempo minore, quindi complessivamente:

$$T(n) = \theta(n^3) \quad .$$

Complessità di spazio:
Viene allocato lo spazio tre blocchi della matrice TI e tre blocchi della matrice TS. Quindi se le matrici hanno dimensione n, la complessità di spazio è pari a:

$$S(n) = \left(\frac{n}{2}\right)^2 \cdot 3 = \frac{3}{4} n^2 \simeq n^2$$

Inoltre viene sprecato spazio per la memorizzazione delle seguenti variabili:

```
float det, alpha;
int i, dim, len;
char uplo, trans, side, diag;
```

ma quest'ultimo è un fattore trascurabile.

Accuratezza fornita

L'algoritmo utilizza numeri float (singola precisione), e restituisce il risultato esatto a meno ad errori di round-off.

Raccomandazioni di utilizzo

Pe far funzionare il programma, non bisogna passargli argomenti dalla linea di comando. Se invece si vuole che venga stampata a video una mini-guida all'uso del programma, basta passargli l'opzione '-h` o '--help`.

Considerazioni sul software sviluppato

La strategia scelta per risolvere il problema pone una seria limitazione all'algoritmo: le matrici TI e TS date in input devono avere un numero di righe (o colonne) pari in modo da poter essere correttamente divise in blocchi.
Inoltre per matrici abbastanza piccole questo algoritmo non è molto performante perchè divide le matrici in blocchi inutilmente. In questo caso è più perfomante un algoritmo che non divide le matrici in blocchi:

```
#include <stdio.h>
#include <stdlib.h>



//Routin di BLAS usate:
void strmm(char *side, char *uplo, char *transa, char *diag, int *m, int *n, float *alpha,
float *a, int *lda, float *b, int *ldb);



void smast(float *a, int n, int m) {

        int i,j;



        for (i=0 ; i<n ; i++) {
                for (j=0 ; j<m ; j++)
                        fprintf(stdout, "  [%2d,%2d] %11f", i+1, j+1, a[(j*(n))+i]);
                fprintf(stdout, "\n");
        }
}


float *smtle(char uplo, float *a, int n) {

        int i,j;


        a = (float *) calloc(n*n, sizeof(float));
```

```
	if (a==NULL) {
		fprintf(stderr, "Errore nell'allocazione della memoria\n");
		return NULL;
	}



	if (uplo == 'u') {
		for (i=0 ; i<n ; i++) {
			for (j=0 ; j<n ; j++) {
				fprintf(stdout, "%2d,%2d>> ", i+1, j+1);
				if (i<=j)
					fscanf(stdin, "%f", &a[(j*(n))+i]);
				else {
					fprintf(stdout, "0.00\n");
					a[(j*(n))+i]=0;
				}
			}
		}
	}
	else {
		for (i=0 ; i<n ; i++) {
			for (j=0 ; j<n ; j++) {
				fprintf(stdout, "%2d,%2d>> ", i+1, j+1);
				if (i>=j)
					fscanf(stdin, "%f", &a[(j*(n))+i]);
				else {
					fprintf(stdout, "0.00\n");
					a[(j*(n))+i]=0;
				}
			}
		}
	}

	return a;
}


int main() {

	float *TI;
	float *TS;
	float det, alpha, beta;
	int i, dim;
	char uplo, trans, side, diag;



	//INIZIO Fase in inpute dei dati
	printf("Inserire la dimensione delle matrici: ");
	scanf("%d", &dim);

	printf("Inserire per righe la matrice TI (Triangolare inferiore):\n");
	TI = smtle('l', TI, dim);
	printf("Inserire per righe la matrice TS (Triangolare superiore):\n");
	TS = smtle('u', TS,  dim);
	//FINE Fase di input dei dati



	//Calcolo del determinante
	det = 1;
	for (i=0 ; i<dim*dim ; i++) {
		det *= *(TI+i);
		i += dim;
	}
	for (i=0 ; i<dim*dim ; i++) {
		det *= *(TS+i);
		i += dim;
	}



	//Calcolo del prodotto della matrice
	alpha = 1;
	beta = 0;
	trans = 'n';
	diag = 'n';
	side='l';
	uplo='l';
	strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI, &dim, TS, &dim);
```

```
            //Stampa dei risultati a video
            printf("[]================[ Stampa della matrice prodotto ]================[]\n");
            smast(TS,dim,dim);

            printf("DETERMINANTE --> %f\n", det);



            //Libero la memoria precedentemente allocata
            free(TI);
            free(TS);


            return EXIT_SUCCESS;
    }
```

Questo codice sorgente è contenuto nel file 'alternativa.c`; per compilare il file di alternativa è stato definito un target nel Makefile, quindi basta digitare il comando 'make alternativa` e verrà creato nella directory corrente un file eseguibile chiamato appunto 'alternativa`.
Non dividendo le matrici in blocchi siamo in grado di calcolare la matrice RE con un'unica chiamata alla routine strmm() di BLAS, a differenza delle sei chiamate effettuate dall'algoritmo che divide le matrici in 4 blocchi.

Riferimenti bibliografici

“A. Murli”
Lucidi del corso di Calcolo Scientifico
http://www.dma.unina.it/~murli/didattica/mat_didattico_cs0405.html
2004-2005

Esempio d'uso

In questo esempio d'uso, daremo in input al programma le due matrici:

$$TI \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 5 & 0 & 0 \\ 4 & 3 & 3 & 0 \\ 9 & 6 & 4 & 7 \end{pmatrix} \quad ; \quad TS \equiv \begin{pmatrix} 6 & 3 & 2 & 7 \\ 0 & 6 & 9 & 7 \\ 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 6 \end{pmatrix}$$

In questo caso la dimensione dei blocchi sarà 2. Dopo aver lanciato il programma:

```
fsterrar@paolella:~/calcolo/elaborato1$ ./elaborato1
```

verrà chiesto all'utente di inserire i dati:

```
Inserire la dimensione di un qualsiasi blocco: 2
Inserire per righe il blocco trinagolare inferiore TI0:
 1, 1>> 1
 1, 2>> 0.00
 2, 1>> 2
 2, 2>> 5
Inserire per righe il blocco TI1:
 1, 1>> 4
 1, 2>> 3
 2, 1>> 9
 2, 2>> 6
Inserire per righe il blocco trinagolare inferiore TI2:
 1, 1>> 3
 1, 2>> 0.00
 2, 1>> 4
 2, 2>> 7
Inserire per righe il blocco trinagolare superiore TS0:
 1, 1>> 6
 1, 2>> 3
 2, 1>> 0.00
 2, 2>> 6
Inserire per righe il blocco TS1:
 1, 1>> 2
 1, 2>> 7
 2, 1>> 9
 2, 2>> 7
Inserire per righe il blocco trinagolare superiore TS2:
 1, 1>> 2
 1, 2>> 4
 2, 1>> 0.00
 2, 2>> 6
```

Dopo aver fatto i calcoli, l'algoritmo presenterà i risultati a sullo schermo in questo modo:

```
[]================[ Stampa del Blocco R0 ]================[]
  [ 1, 1]    6.000000  [ 1, 2]    3.000000
  [ 2, 1]   12.000000  [ 2, 2]   36.000000
[]================[ Stampa del Blocco R1 ]================[]
  [ 1, 1]    2.000000  [ 1, 2]    7.000000
  [ 2, 1]   49.000000  [ 2, 2]   49.000000
[]================[ Stampa del Blocco R2 ]================[]
  [ 1, 1]   24.000000  [ 1, 2]   30.000000
  [ 2, 1]   54.000000  [ 2, 2]   63.000000
[]================[ Stampa del Blocco R3 ]================[]
  [ 1, 1]   41.000000  [ 1, 2]   61.000000
  [ 2, 1]   80.000000  [ 2, 2]  163.000000
DETERMINANTE --> 45360.000000
```

Codice sorgente

smale.c :

```
#include "util.h"


float *smale(float *a, int n, int m) {

        int i,j;


        a = (float *) calloc(n*m, sizeof(float)); //alloco la memoria
        if (a==NULL) { //se c'e' stato un errore durante l'allocazione della memoria
                fprintf(stderr, "Errore nell'allocazione della memoria\n");
                return NULL; //termino la procedura restituendo un indicatore di errore
        }



        for (i=0 ; i<n ; i++) {
                for (j=0 ; j<m ; j++) {
                        fprintf(stdout, "%2d,%2d>> ", i+1, j+1);
                        fscanf(stdin, "%f", &a[(j*(n))+i]);
                }
        }

        return a; //ritorno il puntatore alla matrice creata
}
```

smast.c :

```
#include "util.h"


void smast(float *a, int n, int m) {

        int i,j;



        for (i=0 ; i<n ; i++) {
                for (j=0 ; j<m ; j++)
                        fprintf(stdout, "  [%2d,%2d] %11f", i+1, j+1, a[(j*(n))+i]);
                fprintf(stdout, "\n");
        }
}
```

smtle.c :

```
#include "util.h"



float *smtle(char uplo, float *a, int n) {

	int i,j;


	a = (float *) calloc(n*n, sizeof(float)); //alloco la memoria
	if (a==NULL) { //se c'e' stato un errore durante l'allocazione della memoria
		fprintf(stderr, "Errore nell'allocazione della memoria\n");
		return NULL; //termino la procedura restituendo un indicatore di errore
	}



	if (uplo == 'u') { //se devo leggere una matrice triangolare superiore
		for (i=0 ; i<n ; i++) {
			for (j=0 ; j<n ; j++) {
				fprintf(stdout, "%2d,%2d>> ", i+1, j+1);
				if (i<=j)
					fscanf(stdin, "%f", &a[(j*(n))+i]);
				else {
					fprintf(stdout, "0.00\n");
					a[(j*(n))+i]=0;
				}
			}
		}
	}
	else { //leggo una matrice triangolare inferiore
		for (i=0 ; i<n ; i++) {
			for (j=0 ; j<n ; j++) {
				fprintf(stdout, "%2d,%2d>> ", i+1, j+1);
				if (i>=j)
					fscanf(stdin, "%f", &a[(j*(n))+i]);
				else {
					fprintf(stdout, "0.00\n");
					a[(j*(n))+i]=0;
				}
			}
		}
	}

	return a; //ritorno il puntatore alla matrice creata
}
```

util.h :

```
#ifndef UTIL
#define UTIL



#include <stdio.h>



void smast(float *a, int n, int m);


float *smale(float *a, int n, int m);


float *smtle(char uplo, float *a, int n);



#endif
```

main.c :

```
#include <stdio.h>
#include <stdlib.h>
#include "util.h"


#define NUM_BLOCCHI 3


//Routin di BLAS usate:
void sgemm(char *transa, char *transb, int *m, int *n, int *k, float *alpha, float *a, int
*lda, float *b, int *ldb, float *beta, float *c, int *ldc);

void strmm(char *side, char *uplo, char *transa, char *diag, int *m, int *n, float *alpha,
float *a, int *lda, float *b, int *ldb);

void scopy(int *n, float *x, int *incx, float *y, int *incy);



//Funzione che calcola il determinante di quattro blocchi di matrici trangolari
float calcolo_determinante(float *a, float *b, float *c, float *d, int dim) {
        float det=1;
        int i;

        /*
        Questa funzione puo' essere scritta utilizzando un unico ciclo, in
        questo modo:

                for (i=0 ; i<dim*dim ; i++) {
                        det *= *(a+i);
                        det *= *(b+i);
                        det *= *(c+i);
                        det *= *(d+i);
                        i += dim;
                }

        Anche se questo ciclo e' molto elegante, si e' preferito dividerlo in
        quattro cicli per sfruttare al massimo la localita' dei dati.
        */

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del primo blocco
                det *= *(a+i);
                i += dim;
        }

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del secondo blocco
                det *= *(b+i);
                i += dim;
        }

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del terzo blocco
                det *= *(c+i);
                i += dim;
        }

        for (i=0 ; i<dim*dim ; i++) {  //Calcolo del quarto blocco
                det *= *(d+i);
                i += dim;
        }

        return det;
}


//Stampa di un help
void help() {
        printf
("################################################################################\n");
        printf("#                      LABORATORIO DI CALCOLO SCIENTIFICO
#\n");
        printf("#                               A.A. 2004/2005
#\n");
        printf("#
#\n");
        printf("#                      Ferrara Francesco Saverio - 566/811
#\n");
        printf("#                          fsterrar@studenti.unina.it
#\n");
```

```
        printf
("################################################################################\n\n");
        printf("Questo programma date due matrici triangolari del tipo:\n\n");
        printf("              /       |       \\              /       |       \\\n");
        printf("              |  TI0  |   0   |              |  TS0  |  TS1  |\n");
        printf("        TI =  |_______|_______| ;      TS =  |_______|_______|\n");
        printf("              |       |       |              |       |       |\n");
        printf("              |  TI1  |  TI2  |              |   0   |  TS2  |\n");
        printf("              \\       |       /              \\       |       /\n\n");
        printf("esegue il loro prodotto ottenendo una matrice del tipo:\n\n");
        printf("                            /       |       \\\n");
        printf("                            |  RE0  |  RE1  |\n");
        printf("                    RE =    |_______|_______|\n");
        printf("                            |       |       |\n");
        printf("                            |  RE2  |  RE3  |\n");
        printf("                            \\       |       /\n\n");
        printf("di quest'ultima verra' calcolato il determinante.\n");
        printf("Per un corretto funzionamento inserire riga per riga tutti i
sottoblocchi\n");
        printf("cosi' come vi vengono chiesti dal programma. Chiamare il programma senza
argomenti.\n");
        printf("Per maggiorni informazioni leggere la documentazione allegata.\n");
}


int main(int argc, char *argv[]) {

        float *TI[NUM_BLOCCHI];
        float *TS[NUM_BLOCCHI];
        float det, alpha;
        int i, dim, len;
        char uplo, trans, side, diag;



        //Controllo degli argomenti passati da riga di comando
        if (argc > 1) { //Se sono stati passati argomenti
                if ((argc == 2) && ((strncmp(argv[1],"--help",7)==0) || (strncmp(argv[1],
"-h",3)==0))) {
                        help(); //Stampo a video una mini-guida all'uso del programma
                        return EXIT_SUCCESS; //Esco con un valore di successo
                }
                printf("Errore nel passaggio dei parametri\nUsa: %s --help\nper avere
maggiori informazioni\n", argv[0]);
                return EXIT_FAILURE; //Ritorno alla shell un indicatore di errore
        }


        //INIZIO Fase in inpute dei dati
        printf("Inserire la dimensione di un qualsiasi blocco: ");
        scanf("%d", &dim);

        printf("Inserire per righe il blocco trinagolare inferiore TI0:\n");
        TI[0] = smtle('l',  TI[0], dim);  //Blocco di tipo triangolare inferiore
        printf("Inserire per righe il blocco TI1:\n");
        TI[1] = smale(TI[1],  dim, dim);  //Blocco di tipo generico
        printf("Inserire per righe il blocco trinagolare inferiore TI2:\n");
        TI[2] = smtle('l',  TI[2], dim);  //Blocco di tipo triangolare inferiore
        printf("Inserire per righe il blocco trinagolare superiore TS0:\n");
        TS[0] = smtle('u',  TS[0], dim);  //Blocco di tipo triangolare superiore
        printf("Inserire per righe il blocco TS1:\n");
        TS[1] = smale(TS[1],  dim, dim);  //Blocco di tipo generico
        printf("Inserire per righe il blocco trinagolare superiore TS2:\n");
        TS[2] = smtle('u',  TS[2], dim);  //Blocco di tipo triangolare superiore
        //FINE Fase di input dei dati



        //Calcolo del determinante
        det = calcolo_determinante(TI[0], TI[2], TS[0], TS[2], dim);



        //Calcolo del prodotto della matrice
        alpha = 1; //inizializzo lo scalare alpha
        i=1; //fattore di incremento usato nelle routin saxpy() e scopy()
        trans = 'n'; //considero le matrici non trasposte
        diag = 'n'; //non so se la diagonale di qualche blocco e' unitaria oppure no
        len = dim*dim; //numero di elementi per ogni blocco (usato nelle routin saxpy() e
scopy())

        //blocco R3 --> lo registro in TS[2]
```

```
        // R3 <- TI[1] * TS[1] + TI[2] * TS[2]
        side='l';
        uplo='l';
        strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI[2], &dim, TS[2], &dim);
        sgemm_(&trans, &trans, &dim, &dim, &dim, &alpha, TI[1], &dim, TS[1], &dim, &alpha, TS
[2], &dim);

        //blocco R2 --> lo registro in TI[2]
        // R2 <- TI[1] * TS[0]
        scopy_(&len, TI[1], &i, TI[2], &i);
        side='r';
        uplo='u';
        strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TS[0], &dim, TI[2], &dim);

        //blocco R0 --> lo registro in TS[0]
        // R0 <- TI[0] * TS[0]
        side='l';
        uplo='l';
        strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI[0], &dim, TS[0], &dim);

        //blocco R1 --> lo registro in TS[1]
        // R1 <- TI[0] * TS[1]
        strmm_(&side, &uplo, &trans, &diag, &dim, &dim, &alpha, TI[0], &dim, TS[1], &dim);



        //Stampa dei risultati a video
        printf("[]================[ Stampa del Blocco R0 ]================[]\n");
        smast(TS[0],dim,dim);

        printf("[]================[ Stampa del Blocco R1 ]================[]\n");
        smast(TS[1],dim,dim);

        printf("[]================[ Stampa del Blocco R2 ]================[]\n");
        smast(TI[2],dim,dim);

        printf("[]================[ Stampa del Blocco R3 ]================[]\n");
        smast(TS[2],dim,dim);

        printf("DETERMINANTE --> %f\n", det);



        //Libero la memoria precedentemente allocata
        for (i=0 ; i<NUM_BLOCCHI ; i++) {
                free(TI[i]); //libero i blocchi di TI
                free(TS[i]); //libero i blocchi di TS
        }


        return EXIT_SUCCESS; //Ritorno un indicatore di successo
}
```

Makefile :

```
# Path delle librerie
PATH_LIB = -L/usr/local/lib/
LIB = -lblas -lg2c


# Decommentare la prossima riga per scegliere un altro compilatore
# CC = gcc


# Opzioni passate al compilatore
CFLAG = -funroll-loops


# Nome del file eseguibile da creare
MAIN = elaborato1


# Compilare con 'make DEBUG=yes` se si vogliono aggiungere informazioni per il
# debug all'eseguibile
ifeq ($(DEBUG), yes)
        CFLAG := $(CFLAG) -g
endif


# File sorgenti (.c) e i rispettivi file oggetto (.o)
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:.c=.o)


# Comando per cancellare i file usato durante il target "clean"
DEL = rm -rf


# File momentaneo usato per ricavarsi le dipendenze dei file
DIPENDENZE = .depend


# Comando usato per fare il debug del programma
GDB = gdb

# Target "all" • il target principale.
all : $(DIPENDENZE) $(MAIN)


# Calcolo delle dipendenze per tutti i file dorgenti. Per fare questo si usa
# l'opzione -M del compilatore che produce un output direttamente in formato
# Makefile
$(DIPENDENZE) :
        $(CC) $(CFLAG) -M $(SOURCES) >> $(DIPENDENZE)

# Questo "if" serve a far includere il file con le dipendenze solo quando
# esso • stato effettivamente creato
ifeq ($wildcard $(DIPENDENZE), $(DIPENDENZE))
        include $(DIPENDENZE)
endif

# Linkaggio per avere il file eseguibile
$(MAIN) : $(OBJECTS)
        $(CC) $(CFLAG) -o $(MAIN) *.o $(PATH_LIB) $(LIB)


# Definisco i target PHONY (target che non corrispondono a file)
.PHONY: clean clean_exe clean_dip clean_obj clear run debug


# Pulisce i file dopo la compilazione
clean: clean_exe clean_dip clean_obj


clean_exe:
        $(DEL) $(MAIN)

clean_dip:
        $(DEL) $(DIPENDENZE)

clean_obj:
        $(DEL) *.o
```

```
# Cancella i fastidiosi file temporanei che terminano con il carattere '~«
# creati da alcuni editor di testo.
clear:
	$(DEL) *~


# Manda in esecucione il programma
run:
	./$(MAIN)


# Manda in esecucione il programma
run_example:
	./$(MAIN) < input.txt


# Avvia il debug del programma (se si sceglie questa modalita' bisogna aver
# compilato in precedenza il programma con informazioni per il debug)
debug:
	$(GDB) $(MAIN)
```