RELAZIONE SESTA ESERCITAZIONE

1) RISOLUZIONE DI UN SISTEMA LINEARE Ax=f, DOVE A E' UNA MATRICE TRIDIAGONALE n x n DI ORDINE n E IL TERMINE NOTO E':

$$f_i = \sum_{j=1}^{n} A_{ij} \text{ , i=1,..,n.}$$

CODICE MATLAB, ANALISI E RISULTATI:

```
function [ x ] = tridiagonale( n )

a = zeros(1,n) + 2;
b = zeros(1,n) - 1;
c = zeros(1,n) - 1;
f = zeros(1,n); %termine noto
f(1) = 1;
f(n) = 1;
x = zeros(1,n);
tic
[L,R,alfa,beta] = LR_tridiagonale(a,b,c);
%trovo il vettore y
%L*b=y
y = zeros(1,n);
y(1,1) = f(1,1);
for i=2:1:n
    y(1,i) = f(1,i) - beta(1,i)*y(1,i-1);
end
%DEBUG
%y_mat=L\f'      y fatto da matlab, fa L \ f trasposto
%x_mat=R\y_mat   x fatto da matlab
%y               y mia

%R*x=y
%trovo il vettore x
x(1,n) = y(1,n) / alfa(1,n);
for i=n-1:-1:1
    x(1,i) = (y(1,i) - c(1,i)*x(1,i+1))/alfa(1,i);
end
toc
end
```

```
function [ L,R,alfa,beta ] = LR_tridiagonale( a,b,c )

n = max(size(a));
L = zeros(n, n);
R = zeros(n, n);
alfa = zeros(1,n);
beta = zeros(1,n);
alfa(1,1) = a(1,1);
for i=2:1:n
    beta(1,i) = b(1,i) / alfa(1,i-1);
    alfa(1,i) = a(1,i) - beta(1,i)*c(i-1);
end
%creo L
o = ones(n,1);
L = diag(o) + diag(beta(2:n),-1);
R = diag(alfa) + diag(c(1:n-1),1);
end
```

ESEMPIO DI UTILIZZO:

```
x = tridiagonale(10)

Elapsed time is 0.000204 seconds.

x =

  Columns 1 through 7

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
     1.0000

  Columns 8 through 10

    1.0000    1.0000    1.0000
```

La funzione LR_tridiagonale prende in input i 3 vettori (a,b,c) che rappresentano le 3 diagonali della matrice tridiagonale (a è la diagonale centrale, b è la diagonale sotto, c è la diagonale sopra).
In output restituisce le 2 matrici fattorizzate L ed R, e i 2 vettori alfa e beta che sarebbero rispettivamente la diagonale sotto la centrale di L e la diagonale centrale di R.

Dopo aver trovato L e R, si procede con la sostituizione in avanti per trovare il vettore y:

$$Ly = b$$

$$y_1 = f_1$$

$$y_i = f_i - \beta_i y_{i-1} \quad i=2,\ldots n$$

Infine si trova la soluzione del sistema lineare Ax=f facendo la sostituzione all'indietro:

$$Rx = y$$

$$x_n = \frac{y_n}{\alpha_n}$$

$$x_i = (y_i - c_i x_{i+1}) / \alpha_i \quad i=n-1,n-2,\ldots 1$$

Se al posto dell'algoritmo di Gauss Tridiagonale si fosse usato l'algoritmo di Gauss per matrici piene ci sarebbe stato il fenomeno del fill-in, cioè la maggior parte degli elementi nulli della matrice tridiagonale sarebbero diventati non nulli con l'avanzare dell'algoritmo. Infatti se si prova a risolvere il sistema lineare Ax=f con l'algoritmo di Gauss con Pivoting, il tempo di esecuzione risulta maggiore, esempio:

Creo i tre vettori della matrice tridiagonale

```
a = zeros(1,10) + 2

a =

     2     2     2     2     2     2     2     2     2     2

>> b = zeros(1,10) -1

b =

    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1

>> c = zeros(1,10) -1

c =

    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
```

Creo il termine noto:

```
f = zeros(10,1)

f =

     0
     0
     0
     0
     0
     0
     0
     0
     0
     0

>> f(1) = 1

f =

     1
     0
     0
     0
     0
     0
     0
     0
     0
     0

>> f(10) = 1

f =

     1
     0
     0
     0
     0
     0
     0
     0
     0
     1
```

Creo la matrice A dai 3 vettori a,b, c:

```
A = diag(a) + diag(b(2:10),-1) + diag(c(1:9),1)

A =

     2    -1     0     0     0     0     0     0     0     0
    -1     2    -1     0     0     0     0     0     0     0
     0    -1     2    -1     0     0     0     0     0     0
     0     0    -1     2    -1     0     0     0     0     0
     0     0     0    -1     2    -1     0     0     0     0
     0     0     0     0    -1     2    -1     0     0     0
     0     0     0     0     0    -1     2    -1     0     0
     0     0     0     0     0     0    -1     2    -1     0
     0     0     0     0     0     0     0    -1     2    -1
     0     0     0     0     0     0     0     0    -1     2


x = gauss_pivoting(A,f)

Elapsed time is 0.012197 seconds.

x =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

La soluzione del sistema lineare Ax=f è la stessa che è stata rovata con l'algoritmo di Gauss Tridiagonale, ma il tempo di esecuzione è maggiore.

N.B. La funzione gauss_pivoting è la stessa delle relazioni precedenti.

CONFRONTO DEL TEMPO FRA I 2 ALGORITMI, PER N CHE VARIA FRA 10 E 2000

GAUSS TRIDIAGONALE	GAUSS MATRICI PIENE	N
0.000204 seconds.	0.012197 seconds.	10
0.001046 seconds.	0.739642 seconds.	100
0.175321 seconds.	768.552988 seconds.	1000

Da questi 3 esempi si vede che la complessità dell'algoritmo di Gauss per matrici tridiagonali è lineare
(precisamente è 3(n – 1) operazioni),
mentre l'algoritmo di Gauss per matrici piene ha complessità n^3, questo si vede soprattutto quando si prova l'algoritmo con una matrice tridiagonale di ordine n = 100 e poi si aumenta n di un fattore 10 (n = 1000), il tempo di esecuzione aumenta di un fattore 1000 (da 0.739 secondi a 768 secondi, circa 12 minuti).

2) IMPLEMENTARE L'ALGORITMO DI FATTORIZZAZIONE DI CHOLESKY DI UNA MATRICE A SIMMETRICA E DEFINITA POSITIVA DI ORDINE N.
RISOLVERE POI IL SISTEMA LINEARE Ax=b, DOVE A E' LA MATRIC DI HILBERT E b E' IL TERMINE NOTO FORMATO DA:

$$b_i = \sum_{j=1}^{n} a_{ij} \quad \text{i=1,..,n.}$$

CODICE MATLAB, ANALISI E RISULTATI:

```
function [ x ] = cholesky( A )
n = max(size(A));
L = zeros(n,n);
b = zeros(n,1);
for j=1:1:n
    somma = 0;
    for k=1:1:j-1
        somma = somma + L(j,k) * L (j,k);
    end
    if (A(j,j) - somma < 0)
        error('matrice non definita positiva');
    end
    L(j,j) = sqrt(A(j,j) - somma);
    for i=j+1:1:n
        somma2 = 0;
        for k=1:1:j-1
            somma2 = somma2 + L(i,k)*L(j,k);
        end
        L(i,j) = (A(i,j) - somma2)/L(j,j);
    end
end
LT = L';
%DEBUG prova L*LT = hilbert

%calcolo termine noto
for i=1:1:n
    somma = 0;
    for j=1:1:n
        somma = somma + A(i,j);
    end
    b(i,1) = somma;
end
y = avanti(L,b);
x = indietro(LT,y);
end
```

ESEMPIO DI UTILIZZO:

```
h = hilb(4)

h =

    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429


x = cholesky(h)

x =

    1.0000
    1.0000
    1.0000
    1.0000
```

ANALISI:

La matrice di Hilbert è una matrice mal condizionata, per tanto ci si può aspettare una perturbazione elevata sulla soluzione calcolata rispetto alla soluzione esatta (la soluzione esatta prevede tutte le incognite = 1).
D'altro canto, l'algoritmo di fattorizzazione di Cholesky è stabile in senso forte, in quanto è possibile maggiorare gli elementi di l con una costante che non dipende dall'ordine della matrice.

Per matrici di più grande dimensione, la soluzione calcolata risulta essere diversa dalla soluzione reale, ma l'errore che si commette resta comunque basso, esempio:

```
 h = hilb(10)

>> x = cholesky(h)

x =

    1.0000
    1.0000
    1.0000
    1.0000
    0.9999
    1.0003
    0.9996
    1.0004
    0.9998
    1.0000
```

Per il calcolo dell'errore relativo si considera come x reale il vettore di ordine n con gli elementi tutti uguali ad 1.

Esempio di prima:

```
r = ones(10,1)

r =

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1

err = norm(r - x) / norm(r)

err =  2.1484e-004
```

Lo stesso sistema lineare poteva essere risolto con l'algoritmo di Gauss, esempio (con n = 2):

```
h = hilb(2)

h =

    1.0000    0.5000
    0.5000    0.3333


n = 2

b = zeros(2,1) %inizializza il vettore termine noto a zero

%calcolo termine noto (somma delle righe di h)

for i=1:1:n
    somma = 0;
    for j=1:1:n
        somma = somma + h(i,j);
    end
    b(i,1) = somma;
end

b =
    1.5000
    0.8333
```

```
x = gauss_pivoting(h,b)

x =

    1.0000
    1.0000


err = norm(ones(2,1) - x) / norm(ones(2,1))

err =  5.6610e-016

%ones(2,1) è il vettore con la soluzione reale
```

CONFRONTO FRA ALGORITMO DI CHOLESKY E ALGORITMO DI GAUSS:

ALGORITMO DI CHOLESKY	ALGORITMO DI GAUSS	N
6.3292e-016	5.6610e-016	2
3.7354e-015	1.0167e-014	3
1.8713e-013	3.7776e-013	4
1.2172e-012	1.5531e-012	5
2.4107e-010	2.5168e-010	6
1.0669e-008	8.4252e-009	7
2.1410e-007	2.5057e-007	8
9.1156e-006	8.6087e-006	9
2.1484e-004	2.2377e-004	10
0.0055	0.0046	11
0.1905	0.0955	12
ERRORE	3.0655	13
ERRORE	4.5622	14
ERRORE	3.3721	15

(matrice non definita positiva)

Come si è visto dal confronto dei 2 algoritmi per n che va da 2 a 15, non c'è molta differenza riguardo all'errore che si commette per calcolare la soluzione (anche se l'algoritmo di Gauss è stabile in senso debole).

Per n maggiore di 12, la matrice di hilbert che genera MatLab non è più simmetrica e definita positiva (è solo simmetrica), quindi la fattorizzazione di Cholesky non può essere completata.