Tesina di Calcolo Numerico 2

Corso di laurea triennale in Ingegneria Elettronica

Docente : Giuseppe Rodriguez.

Studenti: Nessi Davide
Cuccu Roberto

Introduzione

Lo scopo della fattorizzazione è quello di scomporre una matrice in un prodotto di matrici aventi una struttura più semplice (matrici triangolari, diagonali ,ortogonali).
In questo elaborato presenteremo alcune fattorizzazioni fra le più comuni e ne forniremo una implementazione in Matlab .

Fattorizzazione QR

E' possibile fattorizzare ogni matrice mxn nella forma

$$A= QR$$

Dove Q è una matrice mxm ortogonale ed R è triangolare superiore con le stesse dimensioni di A.
Quando A è una matrice quadrata non singolare , la fattorizzazione QR puo' essere utilizzata per la risoluzione del sistema lineare Ax = b.
Sostituendo alla matrice A la sua fattorizzazione QR è possibile ottenere i due sistemi:

$$\begin{cases} Qc = b \\ Rx = c \end{cases}$$

Essendo Q ortogonale la soluzione del primo sistema è data da

$$c = Q^T b$$

Mentre il secondo puo' essere risolto per sostituzione all'indietro.

Matrici elementari di Householder.

Una matrice elementare di Householder reale è del tipo:

$$H = I - 2ww^T, w \in \mathbf{R}^n, \|w\| = 1.$$

dove la norma utilizzata qui e nel seguito è quella euclidea.Osserviamo che $2ww^T$ rappresenta una matrice nxn di rango 1.Si dimostra facilmente che che H è una matrice simmetrica e ortogonale.

Assegnato un vettore x, vogliamo determinare w in modo tale che $Hx = ke_1$.Dove e_1 è il primo versore della base canonica di $\mathbf{R}^n$ e $k \in \mathbf{R}$.

Distinguiamo tre fasi per la costruzione di H:

1. Poiché H è ortogonale, allora $\|Hx\| = \|x\| = |k|$, quindi $k = \pm\sigma$ con $\sigma = \|x\|$
2. Applicando la definizione di H si ha: $Hx = x - 2ww^T x = x - 2(w^T x)w = ke_1$ che implica $w = \frac{x - ke_1}{2w^T x} = \frac{x - ke_1}{\|x - ke_1\|}$ in quanto w ha norma unitaria. A questo punto il problema della determinazione di w è risolto. Restano alcune considerazioni numeriche.
3. Dalla relazione $x^T Hx = x^T x - 2(w^T x)^2 = kx_1$ otteniamo

$$(w^T x)^2 = \frac{\sigma^2 - kx_1}{2}$$

L'espressione trovata implica dunque che $\|x - ke_1\| = 2w^T x = \sqrt{2(\sigma^2 - kx_1)}$

L'uso di questa espressione consente di ridurre il calcolo computazionale richiesto per la normalizzazione di w.

Inoltre ,per evitare pericoli di cancellazione,è opportuno scegliere il segno di k, finora arbitrario,in modo che $-kx_1$ sia positivo.

Questo porta alle due espressioni

$$k = -sign(x_1)\sigma, \|x - ke_1\| = \sqrt{2\sigma(\sigma + |x_1|)},$$

definendo $sign(y)$ pari a +1 o -1 a seconda che y sia non negativo oppure negativo.

L'algoritmo risultante è il seguente

1. $\sigma = \|x\|$
2. $k = -sign(x_1)\sigma$
3. $\lambda = \sqrt{2\sigma(\sigma + |x_1|)}$
4. $w = (x - ke_1)/\lambda$
5. $H = I - 2ww^T$

Nella costruzione di H è possibile eliminare la radice quadrata e la normalizzazione del vettore w ,con conseguente riduzione degli errori introdotti dall'algoritmo , osservando che

$$2ww^T = \frac{(x-ke_1)(x-ke_1)^T}{\sigma(\sigma+|x_1|)}$$

Questo consente di esprimere la matrice H nella forma:

$H = I - \frac{1}{\beta}vv^T, v = x - ke_1, \beta = \sigma(\sigma + |x_1|)$.

Osserviamo ,inoltre ,che non è indispensabile costruire esplicitamente H,dato che la conoscenza di w (o di v)determina questa matrice in modo univoco.

Nel seguito sarà utile applicare la matrice H ad un vettore z.

Questo può essere fatto ,senza costruire la matrice H con complessità $O(n)$, nel modo seguente:

$$Hz = (I - 2ww^T)z = z - \gamma w$$

o utilizzando la seconda rappresentazione di H,

$$Hz = (I - \frac{1}{\beta}vv^T)z = z - \delta v$$

Implementazione in Matlab della trasformazione di Householder.

Forniamo ora una possibile implementazione in Matlab relativamente alla trasformazione di Householder.

```
Function [H,v,Hz]=trasfhouseholder(z);
%Trasformazione di Householder
%Input: vettore che si vuole trasformare (z)
%Output: matrice di Householder relativa alla trasformazione (H)
%       vettore inerente alla trasformazione (v=z-ke1)
%       vettore z trasformato (Hz)
Hz=z;
v=z;
[b,a]=size(z);
H=eye(a);

%sigma è uguale alla norma di z
sigma=sqrt(sum(z.^2));

%se z è un vettore nullo imposto v e beta nulli
if sigma==0
   v=0;
   beta=0;
```

```
%se z è diverso da zero opero nel modo seguente
else
   lambda=sigma*(sigma+abs(z(1)));
   beta=1/lambda;

   %assegno a k il valore k=-sign(z1)sigma
   if z(1)<0;
      k=sigma;
   else
      k=-sigma;
   end;
%il vettore v assume il valore v=z-ke1
v(1)=v(1)-k;

%calcolo il vettore trasformato Hz=z-(beta*v'*z*v)
s=sum(v.*z);
s=s*beta;
Hz=z-(s*v);

%costrisco la matrice H=I-(beta*v*v')
g=beta*(v'*v);
H=H-g;
end;
```

Fattorizzazione QR di Householder.

Le matrici elementari di Householder consentono di costruire la fattorizzazione QR di una matrice A.

Supponiamo che A sia una matrice quadrata.

Genereremo una successione di matrici

$A^{(i)}$ i =1,…,n

tale che $A^{(n)}$ sia triangolare superiore.

Al passo 1 dell'algoritmo ,scriviamo la matrice A in termini delle sue colonne.

$$A = A^{(1)} = [a_1^{(1)} a_2^{(1)} ... a_n^{(1)}]$$

E' possibile costruire una matrice , detta matrice elementare di Householder ,tale che

$$H_1 a_1^{(1)} = k_1 e_1$$

Dove e_1 è il primo versore della base canonica.

Applichiamo tale matrice a sinistra della matrice $A^{(1)}$ per generare la prossima matrice della successione

$$A^{(2)} = H_1 A^{(1)} = [a_1^{(2)} a_2^{(2)} ... a_n^{(2)}]$$

con

$$a_1^{(2)} = k_1 e_1$$

$$a_j^{(2)} = H_1 a_j^{(1)}, \; j = 2,...n.$$

La matrice $A^{(2)}$ avrà la seguente struttura:

$$A^{(2)} = \begin{bmatrix} k_1 & a_{12}^{(2)} & ... & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & ... & a_{2n}^{(2)} \\ \mathbf{j} & \mathbf{j} & & \mathbf{j} \\ 0 & a_{n2}^{(2)} & ... & a_{nn}^{(2)} \end{bmatrix} = \begin{bmatrix} k_1 & v_1^T \\ 0 & \hat{A}^{(2)} \end{bmatrix}$$

essendo 0 un vettore colonna di dimensioni appropriate avente tutte le componenti nulle e $\hat{A}^{(2)}$ una sottomatrice di dimensione n-1.
Al passo 2 consideriamo inizialmente la sottomatrice

$$\hat{A}^{(2)} = \left[\hat{a}_2^{(2)} \hat{a}_3^{(2)} ... \hat{a}_n^{(2)} \right]$$

Costruiamo,quindi,la matrice elementare di Householder $\hat{H}_2$
Di dimensione n-1 tale che

$$\hat{H}_2 \hat{a}_2^{(2)} = k_2 e_1$$

dove il vettore e_1 ha questa volta lunghezza n-1. A questo punto ,orliamo la matrice $\hat{H}_2$ con una riga ed una colonna della matrice identità per farle raggiungere la dimensione n

$$H_2 = \begin{bmatrix} 1 & 0^T \\ 0 & \hat{H}_2 \end{bmatrix}$$

e moltiplichiamo a sinistra questa matrice per $A^{(2)}$ ottenendo:

$$A^{(3)} = H_2 A^{(2)} = \begin{bmatrix} k_1 & v_1^T \\ 0 & \hat{H}_2 \hat{A}^{(2)} \end{bmatrix}$$

Il passo successivo operera' sulla sottomatrice $\hat{A}^{(3)}$ di dimensione n-2 .
Prendiamo ora in esame il generico passo i dell'algoritmo.
La matrice prodotta dall'iterazione precedente ha la forma

$$A^{(i)} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{A}^{(i)} \end{bmatrix}$$

mentre la sottomatrice $\hat{A}^{(i)}$ è del tipo

$$\hat{A}^{(i)} = \left[\hat{a}_i^{(i)} \hat{a}_{i+1}^{(i)} ... \hat{a}_n^{(i)} \right]$$

Creiamo ora una matrice $\hat{H}_i$ di dimensione n-i+1 tale che:

$$\hat{H}_i a_i^{(i)} = k_i e_1$$

Dopo avere orlato $\hat{H}_i$ con i-1 righe e colonne dell'identita' si avra'ù

$$A^{(i+1)} = H_i A^{(i)} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & H_i A^{(i)} \end{bmatrix}$$

Se la matrice A è quadrata ,l'algoritmo termina al passo n-1 con la matrice triangolare superiore

$$A^{(n)} = H_{(n-1)} A^{(n-1)} = R$$

Possiamo a questo punto riformulare l'intero algoritmo in forma matriciale

$$R = A^{(n)} = H_{n-1} A^{(n-1)} = H_{n-1} H_{n-2} A^{(n-2)} = ... = H_{n-1} H_{n-2} ... H_1 A^{(1)} = Q^{(T)} A$$

Poiché la matrice

$$Q = H_1 H_2 ... H_{n-1}$$

è ortogonale ,in quanto prodotto di matrici ortogonali , la precedente relazione implica :

$$A = QR$$

Che costituisce la fattorizzazione QR della matrice A.
Se implementato in maniera ottimizzata, l'algoritmo richiede $O(\frac{2}{3}n^3)$ operazioni.
Se A è una matrice rettangolare mxn (m>n) ,l'algoritmo di triangolarizzazione procede esattamenten allo stesso modo mediante prodotti a sinistra per matrici elementari di Housolder di dimensione m .
L'unica differenza consiste nel fatto che è necessario un passo ulteriore per azzerarare gli elementi della n-esima colonna di A(l'ultima).

Implementazione dell'algoritmo in Matlab.

Forniamo ora una possibile implementazione dell'algoritmo per la fattorizzazione QR di Householder.

```
Function [Q,R]=housefattorizzazioneqr(A);
%Algoritmo per la fattorizzazione QR di Householder
%Input: matrice A
%Output: matrici di fattorizzazione Q R
[m,n]=size(A);
r=min([m,n]);
Q=eye(m,m);

%se il numero delle righe supera quello delle colonne
%serve un passo in più nel ciclo for
if m>n
   r=r+1;
end;

%in ogni iterazione del ciclo for:
%si considera il vettore A(k:m,k) che viene trasformato
%con una matrice di householder;
%si aggiorna la matrice A;
%si considera la matrice di householder della trasformazione e le si fa
%raggiungere la dimensione (m,m) orlandola con righe e colonne della
%matrice identità (H);
%si costruisce Q facendo il prodotto matriciale tra le matrici H ad
%ogni iterazione del ciclo;
for k=1:r-1);

   %S è la matrice della trasformazione di Householder del vettore
   %A(k:m,k)
   [S,v,Hz]=trasfhouseholder(A(k:m,k)');

   %si considera la sottomatrice A e si applica S a ogni suo vettore
   %colonna
   for j=k:n;
      A(k:m,j)=S*A(k:m,j);
   end;

   %viene costruita la matrice H orlando S con righe e colonne della
   %matrice identità
   H=eye(m,m);
```

```
    H(k:m,k:m)=S;

    %A=H*A; (in alternativa al ciclo for si può aggiornare A
    %mediante il prodotto matriciale con H

    %Q è il risultato del prodotto matriciale tra le matrici H
    Q=Q*H;

end;

%la matrice R non è altro che la matrice A dopo che ha subito le varie
%trasformazioni
R=A;
```

Fattorizzazione QR di Givens.

Nello spazio $\mathbf{R}^2$ una rotazione piana è rappresentata dalla matrice

$$G = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

Se moltiplicata per un vettore x, tale matrice lo ruota di un angolo θ in senso orario.

La matrice elementare di Givens $G_{ij}(i<j)$ opera in $\mathbf{R}^m$ una rotazione che lascia invariato il sottospazio ortogonale al piano definito dai vettori $e_i \ e_j$ della base canonica .Essa è definita come

$$G_{ij} = \begin{pmatrix} 1 & & & & & & \\ & \mathbf{I} & & & & & \\ & & c & \mathbf{i} & s & & \\ & & \mathbf{j} & & \mathbf{j} & & \\ & & -s & \mathbf{i} & c & & \\ & & & & & \mathbf{I} & \\ & & & & & & 1 \end{pmatrix}$$

con $c^2 + s^2 = 1$.

Osserviamo che per applicarla adf un vettore non è necessario eseguire esplicitamente il prodotto matriciale in quanto è evidente che è possibile determinare i parametri c ed s in modo tale da annullare la componente j-esima del vettore x , modificando solo la componente i-esima e lasciando inalterate tutte le altre .Infatti imponendo $y_j = 0$ si ottiene

$$c = \frac{x_i}{x_j} s ,$$

che ,sostituita nella relazione $c^2 + s^2 = 1$, fornisce

$$\left(\frac{x_i^2}{x_j^2} + 1 \right) s^2 = 1$$

da cui discendono

$$s^2 = \frac{x_j^2}{x_i^2 + x_j^2} \quad \text{e} \quad c^2 = \frac{x_i^2}{x_i^2 + x_j^2}$$

Le espressioni di c e s sono quindi

$c = \frac{x_i}{\sigma}$ e $s = \frac{x_j}{\sigma}$ con $\sigma = \sqrt{x_i^2 + x_j^2}$

Mediante una opportuna successione di matrici elementari di Givensè possibile trasformare un vettore di $\mathbf{R}^m$ in modo analogo all'azione di una matrice elementare di householder.Infatti ,ponendo

$$G = G_{1m} ... G_{13} G_{12} ,$$

si ha che ;

$$Gx = ke_1$$

L'eventuale vantaggio,rispetto all'approccio di Householder,sta nel fatto che se x ha molti elementi nulli possiamo operare selettivamente solo su quelli diversi da zero,riducendo cosi' il numero delle matrici di Givens utilizzate e, conseguentemente la complessità computazionale.

Se si moltiplica a sinistra la matrice G_{ij} per una matrice A , si applica la medesima trasformazione a tutte le colonne di A.

Se si moltiplica a destra G_{ij} per A,vengono trasformate in modo analogo tutte le righe di A.

Implementazione in Matlab di una rotazione elementare di Givens.

```
Function [c,s]=givrot(x1,x2);
%rotazione elementare di Givens
%Input: 2 elementi x1 e x2 di un vettore
%Output: parametri c ed s tali da annullare l'elemento x2 di un vettore

%se l'elemento x2 è già nullo non c'è bisogno di compiere alcuna
rotazione
if x2==0
  c=1;s=0;
```

```
%se x2 è diverso da 0 allora calcolo c ed s tali da annullarlo
else
  if abs(x2)>abs(x1)
    t=x1/x2; s=1/sqrt(1+t^2);c=s*t;
  else
    t=x2/x1; c=1/sqrt(1+t^2);s=t*c;
  end;
end;
```

Algoritmo di fattorizzazione QR di Givens.

Consiste nell'azzerare tutti gli elementi del triangolo inferiore di A a cominciare dal secondo elemento della prima colonna e proseguendo verso il basso con gli altri elementi ,continuando poi con le colonne successive. Se la matrice è quadrata ci si ferma con l'ultimo elemento della penultima colonna ,e il procedimento può essere rappresentato in forma matriciale nel modo seguente:

$$G_{n-1,n}G_{n-2,n}G_{n-2,n-1}...G_{2n}...G_{23}G_{1n}...G_{13}A = R$$

(naturalmente il prodotto matriciale va letto da destra verso sinistra per rispettare l'ordine di applicazione delle rotazioni di Givens).
Se invece la matrice è rettangolare con m righe ed n colonne(m>n)
È necessario azzerare anche gli elementi subdiagonali dell'ultima colonna , e l'algoritmo diventa :

$$G_{n,m}...G_{n,n+1}G_{2m}...G_{23}G_{1m}...G_{13}G_{12}A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

Con R_1 matrice triangolare superiore nxn.
In entrambi i casi il prodotto delle matrici di Givens G_{ij} fornisce la trasposta del fattore Q della fattorizzazione QR di A.
Tale fattore puo' essere costruito esplicitamente ,oppure possono essere memorizzati i parametri c_{ij} e s_{ij} che definiscono ciascuna matrice di rotazione G_{ij} , e che consentono quindi ,all'occorenza ,di calcolare il prodotto $Q^T b$ riapplicando al vettore b tutte le rotazioni già applicate alla matrice A.
Questo algoritmo è particolarmente conveniente ,rispetto a quello di Householder ,se applicato a matrici sparse ,in quanto è un metodo che

consente di ridurre la complessità computazionale in presenza di molti zeri del triangolo inferiore della matrice di fattorizzazione .
Se applicato a matrici quadrate piene richiede $O\left(\frac{4}{3}n^3\right)$ moltiplicazioni ($O\left(2n^2(m-\frac{1}{3}n)\right)$ per matrici rettangolari),il doppio di Householder.

Implementazione in Matlab dell'Algoritmo di fattorizzazione QR di Givens.

```
Function [Q,R]=fattgivensqr(A);
%Fattorizzazione QR di una matrice mediante le rotazioni di Givens
%Input: matrice che si desidera fattorizzare (A)
%Output: matrici risultanti dalla fattorizzazione (Q,R)
[m,n]=size(A);
k=min(m,n);
Q=eye(m);

%se il numero delle righe di a supera quello delle colonne è necessaria
%un'altra    simo   ione del ciclo for
if m>n
   k=k+1;
end;

%il primo ciclo for considera la colonna    simo
for i=1:k-1)

   %il secondo ciclo for considera ogni elemento della colonna    simo
   %sottostante all'elemento diagonale
   for j=i+1:m

      %se l'elemento    simo è già nullo non compio nessuna operazione,
      %pertanto opero solo nel caso A(j,i)~=0
      if A(j,i)~=0

         %ricavo i parametri c,s della rotazione di Givens tra
         %l'elemento diagonale e l'elemento    simo
         [c,s]=givrot(A(i,i),A(j,i));
```

```
        %tramite il ciclo for aggiorno la matrice A
        %(equivalente del prodotto tra la matrice di Givens e A)
        for r=i:n
           t=(-s*A(i,r))+(c*A(j,r));
           A(i,r)=(c*A(i,r))+(s*A(j,r));
           A(j,r)=t;
        end;

        %costruisco la matrice di Givens inerente alla rotazione svolta
        G=eye(m);
        G(i,i)=c;
        G(j,j)=c;
        G(i,j)=s;
        G(j,i)=-s;

        %A=G*A; (posso fare il prodotto matriciale in alternativa al
        %ciclo for eseguito sopra)

        %calcolo Q facendo il prodotto tra le matrici di Givens
        %utilizzate e mando in output la sua trasposta
        Q=G*Q;
      end;
   end;
end;

%la matrice R è pari alla matrice A che ha subito le varie trasformazioni
R=A;

%la matrice Q della fattorizzazione è pari alla trasposta della matrice
%calcolata col prodotto fra le varie matrici di Givens usate
Q=Q';
```

Fattorizzazione LU

Definiamo matrice elementare di Gauss una matrice del tipo

$$M_k = I - m_k e_k^T$$

essendo I la matrice identità , e_k il k-esimo versore della base canonica e

$$m_k = \left(0, \mathbf{i}, 0, m_{k+1,k}, m_{k+2,k}, \mathbf{i}, m_{nk}\right)^T$$

osserviamo come M_k agisce su una matrice A rappresentata, per comodità, mediante le sue righe a^k

$$M_k A = \begin{bmatrix} 1 & & & & & \\ & \mathbf{l} & & & & \\ & & 1 & & & \\ & & -m_{k+1,k} & 1 & & \\ & & -m_{k+2,k} & & 1 & \\ & & \mathbf{j} & & & \mathbf{l} \\ & & -m_{n,k} & & & 1 \end{bmatrix} \begin{bmatrix} a^1 \\ \mathbf{j} \\ a^k \\ a^{k+1} \\ a^{k+2} \\ \mathbf{j} \\ a^n \end{bmatrix} = \begin{bmatrix} a^1 \\ \mathbf{j} \\ a^k \\ a^{k+1} - m_{k+1,k} a^k \\ a^{k+2} - m_{k+2,k} a^k \\ \mathbf{j} \\ a^n - m_{n,k} a^k \end{bmatrix}$$

Moltiplicare una matrice a sinistra per M_k corrisponde all'applicazione del k-esimo passo dell'algoritmo di Gauss .

L'intero algoritmo puo' quindi essere rappresentato in forma matriciale

$$U = A^{(n)} = M_{n-1} A^{(n-1)} = M_{n-1} M_{n-2} A^{(n-2)} = M_{n-1} M_{n-2} ... M_1 A^{(1)} = L^{-1} A$$

dove la matrice $L^{-1} = M_{n-1} M_{n-2} ... M_1$ è triangolare inferiore in quanto prodotto di matrici triangolari inferiori.

Le matrici elementari di Gauss hanno due importanti proprietà:

1. $M_k^{-1} = I + m_k e_k^T$ l'inversa è ancora una matrice elementare di Gauss.
2. $M_k^{-1} M_{k+r}^{-1} = I + m_k e_k^T + m_{k+r} e_{k+r}^T$ il prodotto di due matrici di indice crescente può essere costruito direttamente senza effettuare calcoli.

Queste proprietà consentono di scrivere analiticamente l'inversa di L^{-1}

$$L = M_1^{-1} M_2^{-1} ... M_{n-1}^{-1} = \begin{bmatrix} 1 & & & \\ m_{21} & \mathbf{l} & & \\ \mathbf{j} & \mathbf{l} & \mathbf{l} & \\ m_{n1} & \mathbf{i} & m_{n,n-1} & 1 \end{bmatrix}$$

e quindi di ottenere la cosiddetta fattorizzazione LU della matrice A come prodotto collaterale dell'applicazione dell' algoritmo di riduzione di Gauss. Se tale fattorizzazione è nota ,la risoluzionedi un sistema lineare Ax=bpuo' essere ricondotta alla risoluzione in cascata dei due sistemi triangolari:

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

Altre fattorizzazioni LU.

La fattorizzazione A=LU ,descritta nelle sezioni precedenti, esiste ed è unica se e solo se la matrice A è non singolare e tutti i suo minori principali sono non nulli. Da ciò discende anche l'esistenza ed unicità della cosiddetta fattorizzazione LDR (oLDU).

Sia A=LU . Se poniamo

$$D = diag(u_{11}, ..., u_{nn})$$

e $R = D^{-1} U$, la matrice A risulta decomposta nella forma

A=LDR,

con L triangolare inferiore ,Rtriangolare superiore , $l_{ii} = r_{ii} = 1, i = 1...n.$

Se A è simmetrica è facile osservare che risulta $R = L^T$ e quindi

$$A = LDL^T$$

L'algoritmo di Gauss può essere modificato in maniera opportuna per costruire queste fattorizzazioni.

Si definisce inerzia della matrice Auna terna di numeri naturali

$$Inerzia(A) := \{p, n, z\}$$

che indicano, rispettivamente,il numero di autovalori positivi, negativi e nulli di A. Vale il seguente teorema:

Legge di Inerzia di Sylvester *Se A è una matrice quadrata e C è una matrice non singolare ,allora*

$$Inerzia(A) = Inerzia(C^T AC).$$

Se la matrice A è simmetrica definita positiva ,applicando il teorema appena enunciato alla fattorizzazione possiamo concludere che tutti gli elementi diagonali d_i della matrice D sono positivi, ed è quindi possibile definire la matrice

$$D^{1/2} = diag(\sqrt{d_1}, ..., \sqrt{d_n})$$

e la matrice $R = D^{1/2} L^T$, con $r_{ii} > 0$, in modo che risulti

$$A = LD^{1/2} D^{1/2} L^T = R^T R$$

La fattorizzazione $A = R^T R$ della matrice definita positiva A viene detta fattorizzazione di Cholesky.

L'analisi del prodotto matriciale:

$$A = R^T R = \begin{bmatrix} r_{11} & & & & \\ \mathbf{j} & \mathbf{l} & & & \\ r_{1i} & \mathbf{i} & r_{ii} & & \\ \mathbf{j} & & & \mathbf{l} & \\ r_{1n} & \mathbf{i} & \mathbf{i} & \mathbf{i} & r_{nn} \end{bmatrix} \begin{bmatrix} r_{11} & \mathbf{i} & r_{1j} & \mathbf{i} & r_{1n} \\ & \mathbf{l} & \mathbf{j} & & \mathbf{j} \\ & & r_{jj} & & \mathbf{j} \\ & & & \mathbf{l} & \mathbf{j} \\ & & & & r_{nn} \end{bmatrix}$$

consente di esprimere gli elementi del triangolo superiore di A in funzione degli elementi di R(gli elementi del triangolo inferiore si ottengono per trasposizione, essendo A simmetrica).

$$a_{ij} = \sum_{k=1}^{i} r_{ki} r_{kj}, i \le j,$$

e quindi di esprimere gli elementi di R in funzione di quelli di A

$$r_{ij} = \frac{1}{r_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right), i < j$$

$$r_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2 \right)^{\frac{1}{2}}, i = j$$

Il fatto che la matrice sia definita positiva garantisce che il radicando che compare nell'espressione di r_{ij} risulti positivo.

Implementazione in Matlab dell'algoritmo di Cholesky.

```
Function R=fchol(A)
%Fattorizzazione Cholesky
%
%Input: matrice da fattorizzare (A)
%Output: R matrice triangolare superiore tale che A=R'*R

[n,m]=size(A);

if n~=m
   error('ERRORE: la matrice non è quadrata')
end

R=zeros(n,n);

for j=1:n

   if (A(j,j)-sum(R(1:j-1,j).^2))<0;
      error('ERRORE: la matrice di ingresso non è definita positiva')
   else

      for i=1:j-1)
         R(i,j)=(A(i,j)-sum(R(1:i-1,i).*R(1:i-1,j)))/R(i,i);
      end

      R(j,j)=sqrt(A(j,j)-sum(R(1:j-1,j).^2));
   end
end
```

Test algoritmi prodotti.

A questo punto ci proponiamo di testare gli algoritmi prodotti confrontandoli con quelli di Matlab.

Procediamo in questo modo:

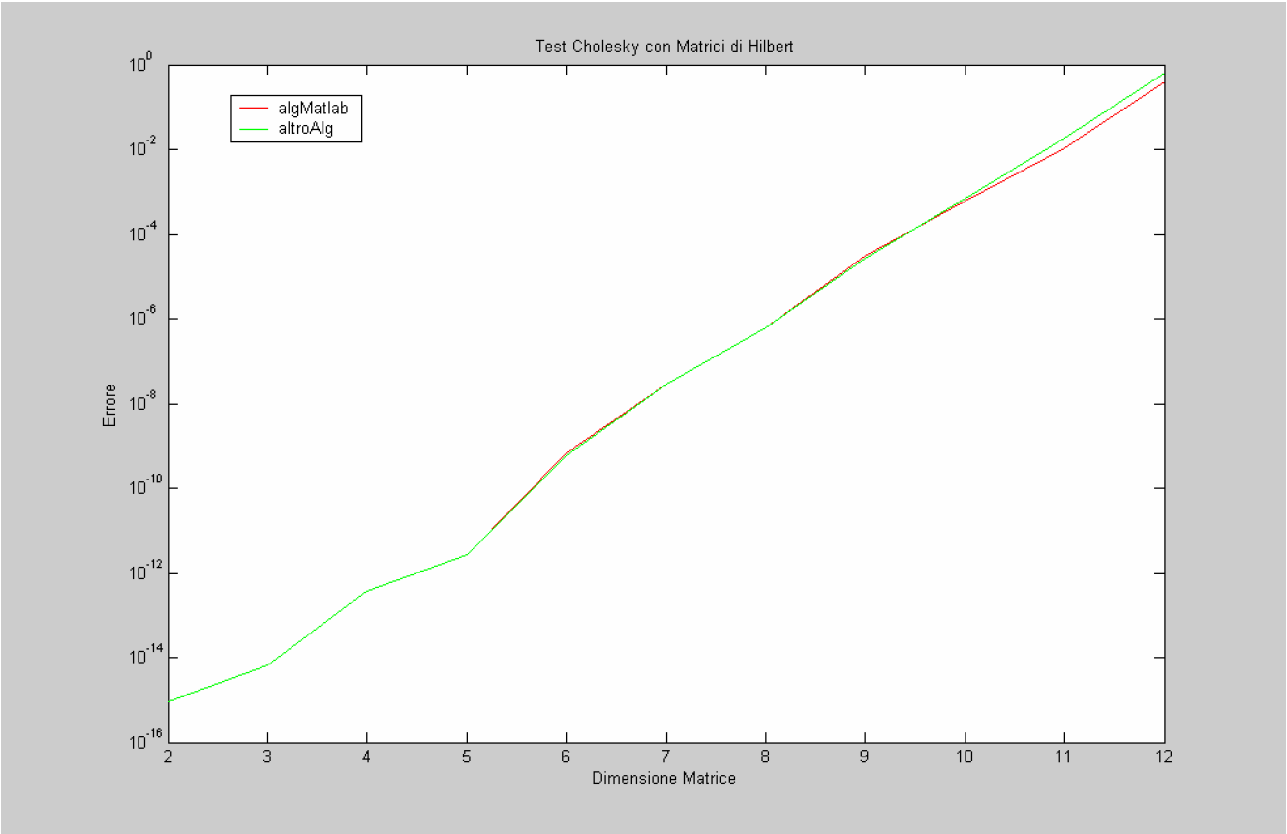
1. Creiamo un sistema lineare del tipo $Ax = b$ del quale conosciamo la soluzione esatta x.

2. Fattorizziamo la matrice A utilizzando rispettivamente l’algoritmo di Matlab e l’algoritmo implementato.
3. Andiamo a valutare l’errore ossia lo scostamento dalla soluzione esatta a noi nota .
4. Incrementiamo la dimensione del sistema e torniamo al passo 3.

Test Fattorizzazione di Cholesky.

```
%Test Cholesky con matrici di Hilbert
clear all
close all
clc

n = 12;  %oltre 12X12 va in errore

for i= 1:n


A =hilb(i);
e = ones(i,1);
b = A*e;
%Risoluzione con algoritmo di Matlab
R = chol(A);
y = R'\b;
x= R\y;
%Erroe usando l'algoritmo di matlab
err1(i) = norm(x-e);
%Risoluzione con l'algoritmo implementato

R = fchol(A);
x=soluzionesistemacholesky(R,b);
err2(i) = norm(x-e);
end

%Grafico errore
t = linspace(1,12,12);
semilogy(t,err1,'r',t,err2,'g');
legend('algMatlab','altroAlg');
xlabel('Dimensione Matrice');
```

ylabel('Errore');
title('Test Cholesky con Matrici di Hilbert');



Dal test possiamo notare che lo scostamento tra l'algoritmo da noi implementato e quello presente nella libreria di Matlab non è significativo.

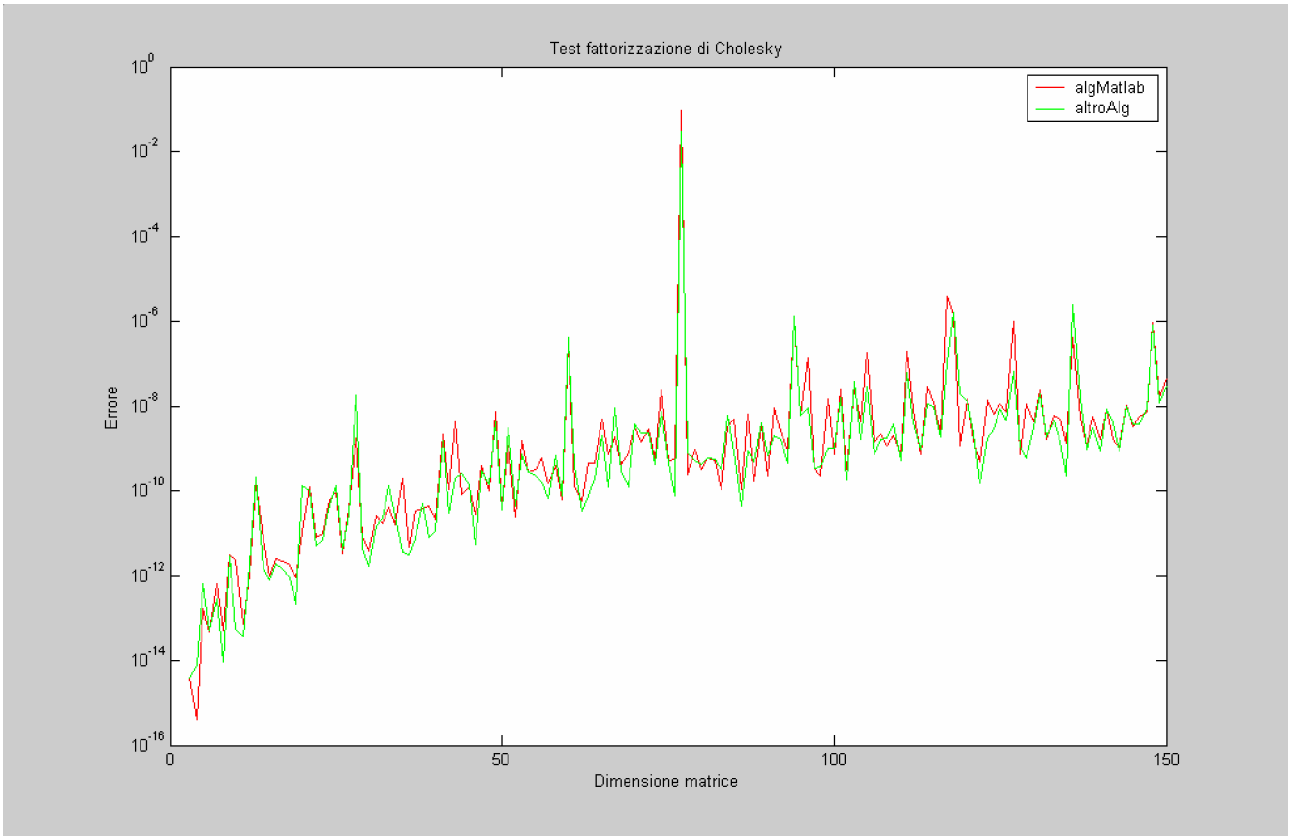
Ora testiamo con matrici definite positive (random).

%test Cholesky con matrici definite positive
clear all
close all
clc

n = 150;

for i= 1:n
A =rand(i);
*A=A*A';*
e = ones(i,1);

```
b = A*e;
%risoluzione con algoritmo di Matlab
R = chol(A);
y = R'\b;
x= R\y;
%errore  usando l'algoritmo di Matlab
err1(i) = norm(x-e);
%risoluzione con altro algoritmo
R = fchol(A);
x=soluzionesistemacholesky(R,b);
%errore con altro algoritmo
err2(i) = norm(x-e);
end



t = linspace(1,n,n);
semilogy(t,err1,'r',t,err2,'g')
legend('algMatlab','altroAlg');
xlabel('Dimensione matrice');
ylabel('Errore');
title('Test fattorizzazione di Cholesky');
```

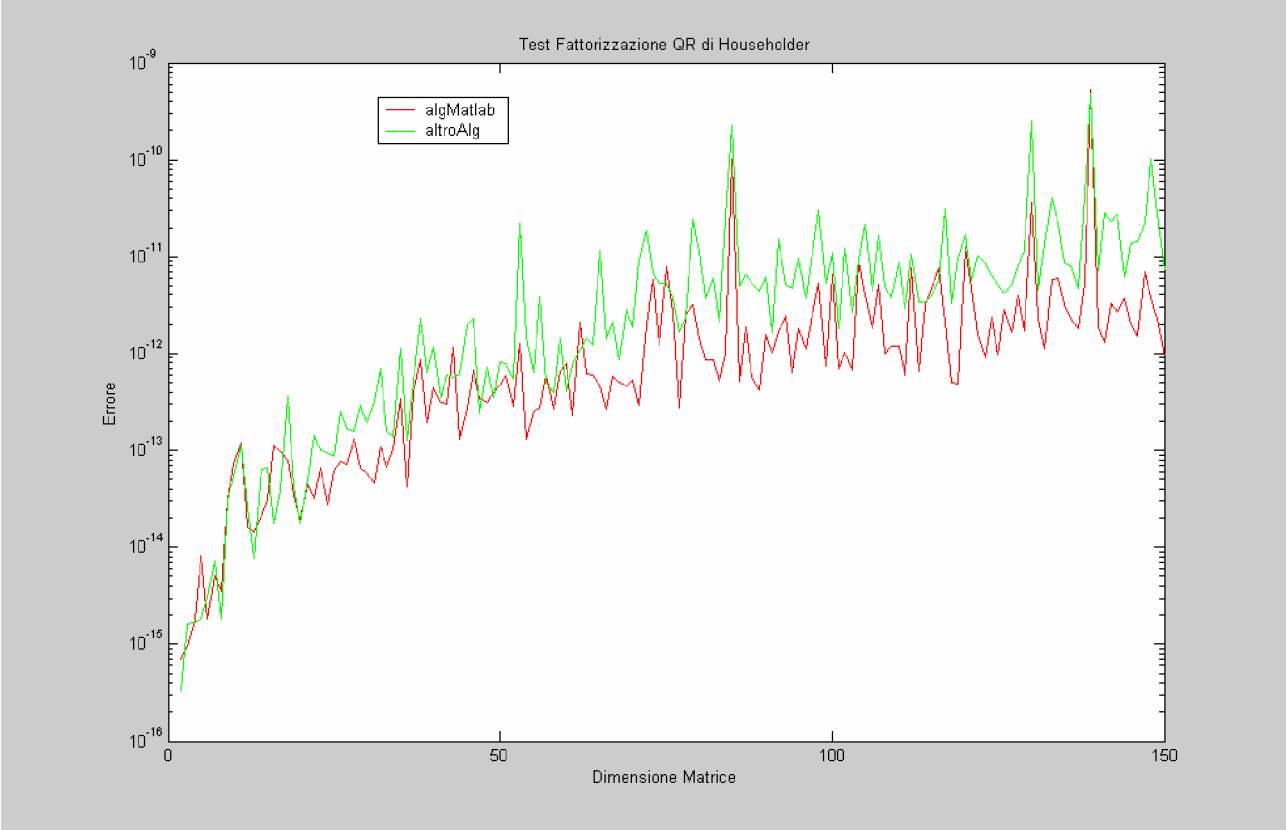


Anche in questo test l'errore prodotto dai due algoritmi è simile.

Test fattorizzazione di Householder.

```
%Test fattorizzazione QR di Householder con matrici casuali.
clear all
close all
clc

n = 150;

for i= 1:n

A =rand(i);
e = ones(i,1);
b = A*e;
%Risoluzione con algoritmo di Matlab
[Q,R] = qr(A);
y = Q'*b;
x= R\y;
%Errore usando l'algoritmo di matlab
```

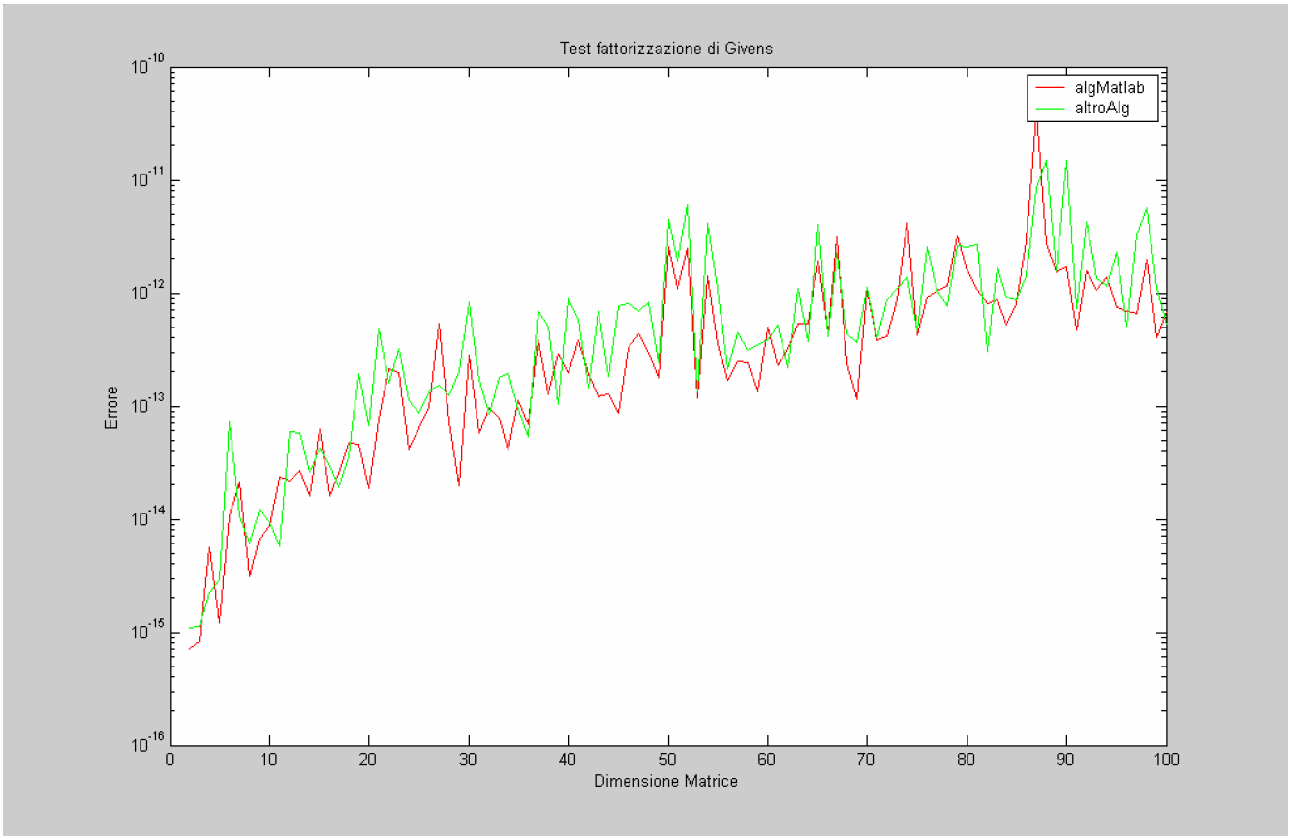
```
err1(i) = norm(x-e);
%Risoluzione con l'algoritmo implementato

[Q,R]= housefattorizzazioneqr(A);
x=soluzionesistemaqr(Q,R,b);
err2(i) = norm(x-e);
end

%Visualizza vettore degli errori(in norma 2).
t = linspace(1,n,n);
semilogy(t,err1,'r',t,err2,'g');
legend('algMatlab','altroAlg');
xlabel('Dimensione Matrice');
ylabel('Errore');
title('Test Fattorizzazione QR di Householder');
```



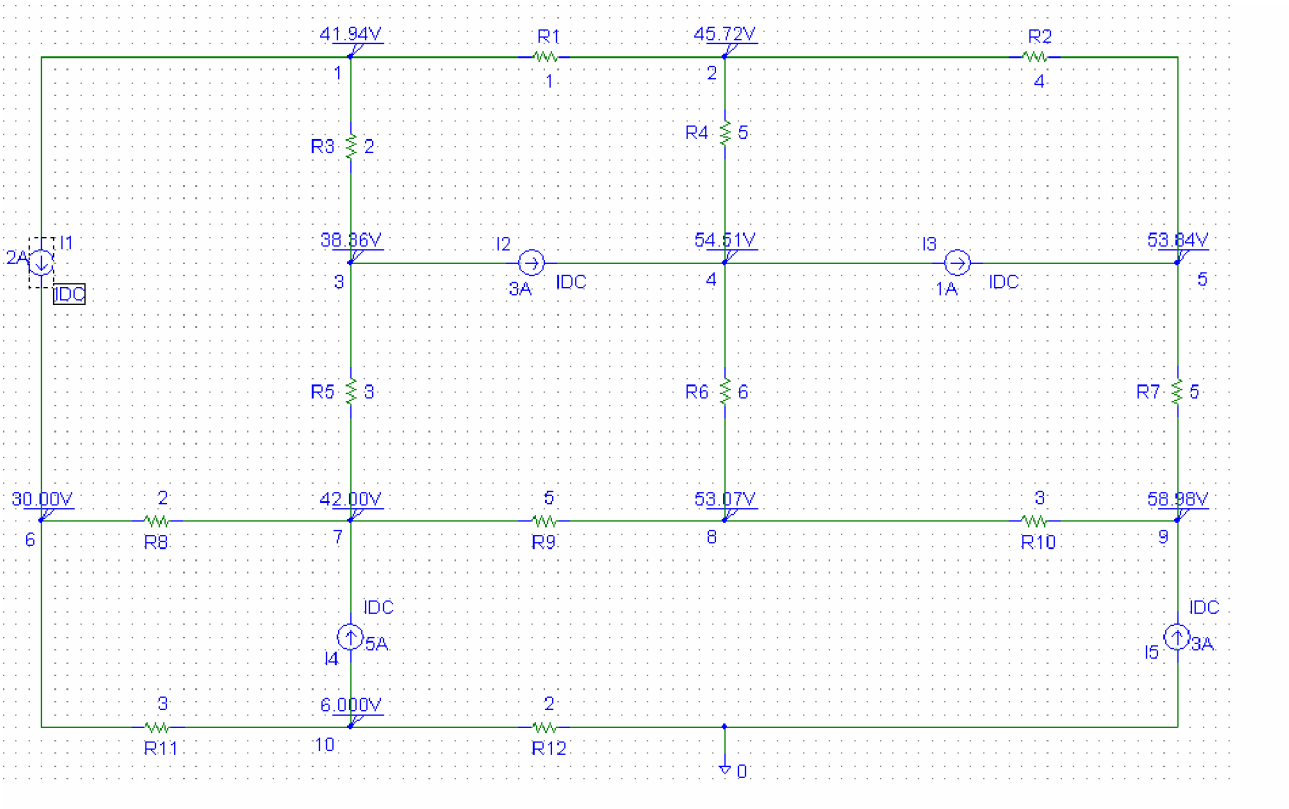
In questo caso invece notiamo che l'implementazione fornita da Matlab è decisamente migliore rispetto alla nostra.

Test Fattorizzazione di Givens.

```
%Test fattorizzazione di Givens con matrici casuali
clear all
close all
clc

n = 100;

for i= 1:n

A =rand(i);  %genero matrice random
e = ones(i,1);
b = A*e;
%Risoluzione con algoritmo di matlab
[Q,R] = qr(A);
y = Q'*b;
x= R\y;
%Errore usando l'algoritmo di matlab
err1(i) = norm(x-e);
%Risoluzione con l'algoritmo implementato

[Q,R]= fattgivensqr(A);
x=soluzionesistemaqr(Q,R,b);
err2(i) = norm(x-e);
end

%Visualizza vettore degli errori(in norma 2).
t = linspace(1,n,n);
semilogy(t,err1,'r',t,err2,'g');
legend('algMatlab','altroAlg');
xlabel('Dimensione Matrice');
ylabel('Errore');
title('Test fattorizzazione di Givens');
```



Applicazione della fattorizzazione alla soluzione di un circuito.



Ora utilizzeremo la fattorizzazione in un esempio applicativo.
In generale , se un circuito con generatori di corrente indipendenti ha N nodi oltre a quello di riferimento,le equazioni nelle tensioni di nodo possono essere scritte in termini di conduttanze come

$$Gv = i$$

esplicitamente:

$$\begin{bmatrix} G_{11} & G_{12} & \mathbf{i} & G_{1N} \\ G_{21} & G_{22} & \mathbf{i} & G_{2N} \\ \mathbf{j} & \mathbf{j} & \mathbf{j} & \mathbf{j} \\ G_{N1} & G_{N2} & \mathbf{i} & G_{NN} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \mathbf{j} \\ v_N \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ \mathbf{j} \\ i_N \end{bmatrix}$$

Dove :

- G_{kk} =Somma delle conduttanze collegate al nodo k
- $G_{kj} = G_{jk}$ =Somma cambiata di segno delle conduttanze tra i nodi k e j, $k \neq j$
- v_k =tensione incognita del nodo k
- i_k =Somma delle correnti dei generatori di corrente collegati al nodo k ,con le correnti entranti considerate positive

G è detta matrice delle conduttanze, v è il vettore delle uscite e i è il vettore degli ingressi.
Risolvendo il sistema lineare

$$Gv = i$$

è possibile ottenere le tensioni di nodo incognite .
Tutto questo vale per circuiti contenenti solo generatori di corrente indipendenti e resistori lineari.
Quello che ci proponiamo di fare è:

1. Fattorizzare la matrice G .
2. Risolvere il sistema lineare.
3. Confrontare i risultati ottenuti con quelli forniti da P-Spice.

Osservando la matrice G possiamo notare che è simmetrica,a predominanza diagonale e inoltre gli elementi diagonali sono positivi.

Quindi per il primo teorema di Gershgorin è definita positiva e pertanto fattorizzabile con Cholesky. Inoltre data la struttura del circuito, dato che non tutti i nodi sono collegati fra loro, la matrice G risulta essere sparsa.

Fattorizzando con Cholesky e risolvendo il sistema lineare utilizzando un opportuno algoritmo da noi implementato:

```
function [x]=soluzionesistemacholesky(R,b);
%risoluzione di un sistema lineare (Cholesky)
[n,m]=size(R);
R=R';
c=zeros(n,1);
x=zeros(n,1);

%risoluzione sistema triangolare inferiore
for i=1:n
      c(i)=b(i)/R(i,i);
      b(i+1:n,1)=b(i+1:n,1)-(R(i+1:n,i)*c(i));

   end;

R=R';
%risoluzione sistema triangolare superiore
for i=0:(n-1)
      x(n-i)=c(n-i)/R(n-i,n-i);
      c(1:n-i-1,1)=c(1:n-i-1,1)-(R(1:n-i-1,n-i)*x(n-i));

   end;
```

Risolto il sistema lineare otteniamo il seguente vettore delle tensioni di nodo:

VC =

41.93506493506500
45.72207792207799
38.36103896103901
54.51428571428577
53.83636363636369
30.00000000000002
42.00000000000004
53.06493506493511
58.97922077922083
6.00000000000001

Per avere un termine di confronto risolviamo il sistema lineare tramite un algoritmo che sfrutta la fattorizzazione QR. Come detto in precedenza G è una matrice sparsa pertanto è conveniente utilizzare l'algoritmo di Givens per la fattorizzazione QR.
Per la risoluzione del sistema utilizziamo il seguente algoritmo:

```
function [x]=soluzionesistemaqr(Q,R,b);
%Risoluzione di un sistema lineare mediante fattorizzazione QR
%Input: matrici della fattorizzazione (Q,R) e vettore dei termini noti (b)
%Output: vettore soluzione del sistema
[m,n]=size(R);
[z,v]=size(b);

if m~=z
  error('Il vettore dei termini noti ha dimensione diversa dalla matrice');
end;

if m>n
   fprintf('Il sistema è sovradeterminato \n');
   x=0;
end;

if m<n
    fprintf('Il sistema è sottodeterminato \n');
    x=0;
end;

%primo passo per la risoluzione del sistema: c=Q'*b
```

```
if m==n
   c=Q'*b;
   x=zeros(m,1);

   %secondo passo per la risoluzione del sistema: si risolve con il
metodo
   %di sostituzione a partire dall'ultimo elemento della matrice
   %triangolare R
   for i=0:(m-1)
      x(m-i)=c(m-i)/R(m-i,m-i);
      c(1:m-i-1,1)=c(1:m-i-1,1)-(R(1:m-i-1,m-i)*x(m-i));

   end;
end;
```

In questo caso abbiamo come soluzione il seguente vettore:

```
VG =

  41.93506493506499
  45.72207792207800
  38.36103896103901
  54.51428571428579
  53.83636363636374
  30.00000000000001
  42.00000000000003
  53.06493506493514
  58.97922077922087
   6.00000000000001
```

Calcolando lo scostamento tra i due risultati :

```
err = norm(VC-VG)

err =

   7.983731028902323e-014
```

Notiamo che i due algoritmi di fattorizzazione forniscono risultati piuttosto simili.

Inoltre entrambi i risultati sono in buono accordo con quelli forniti dal software P-Spice, pur essendo questi ultimi approssimati alla seconda cifra decimale.