Metodi ai minimi quadrati e regolarizzazione

Corso di laurea Matematica

Gabriele Stocchino

Anno di corso 2014/2015

Implementazione degli algoritmi per la fattorizzazione QR

Di seguito verranno riportati gli algoritmi per la fattorizzazione QR sfruttando i metodi di Housholder, di Givems, la Classical Gram-Schmidt (CGS) e la Modified Gram-Schmidt (MGS). Questi metodi sono implementati in MATLAB®.

Fattorizzazione QR di Householder

L'algoritmo della fattorizzazione QR di Householder è il seguente:

Algorithm 1

```
function [Q,A]=qrh(A)
n=size(A);
Q=eye(n(1));
if n(1)==n(2);
   for i=1:n(1)-1
       [u,delta]=aus(A(i:n(1),i));
       % aggiorno A
       A(i:n,:)=A(i:n,:)-u*u'*A(i:n,:)/delta;
       % aggiorno Q
       Q(:,i:n)=Q(:,i:n)-Q(:,i:n)*u*u'/delta;
   end
else
    m=min(n(1),n(2));
    for i=1:m
        [u,delta]=aus(A(i:n(1),i));
        % aggiorno A
        A(i:n(1),:)=A(i:n(1),:)-u*u'*A(i:n(1),:)/delta;
        % aggiorno Q
        Q(:,i:n(1))=Q(:,i:n(1))-Q(:,i:n(1))*u*u'/delta;
    end
```

```
end
```

Dove la funzione aus è la seguente:

Algorithm 2

```
function [x,beta]=aus(x)
n=length(x);
sigma=norm(x);
k=-sign(x(1))*sigma;
beta=sigma*(sigma+abs(x(1)));
x(1)=x(1)-k;
```

Fattorizzazione QR di Givens

L'algoritmo della fattorizzazione QR di Givens è il seguente:

Alghoritm 3

```
function [Q,A]=qrg(A)
n=size(A);
Q=eye(n(1));
if n(1)==n(2)
    for k=1:n-1
        for i=k+1:n
             if ne(A(i,k),0)==1
                [c,s]=givaux(A(k,k),A(i,k));
                %   aggiorno A
                t=c*A(k,k:n)+s*A(i,k:n);
                A(i,k:n)=c*A(i,k:n)-s*A(k,k:n);
                A(k,k:n)=t;

                %   aggiorno Q
                u=c*Q(:,i)-s*Q(:,k);
                Q(:,k)=s*Q(:,i)+c*Q(:,k);
                Q(:,i)=u;
            end
        end
    end
```

```
else
    for k=1:n(2)
        for i=k+1:n(1)
            if ne(A(i,k),0)==1
               [c,s]=givaux(A(k,k),A(i,k));
                %   aggiorno A
                t=c*A(k,k:n(2))+s*A(i,k:n(2));
                A(i,k:n(2))=c*A(i,k:n(2))-s*A(k,k:n(2));
                A(k,k:n(2))=t;

                %   aggiorno Q
                u=c*Q(:,i)-s*Q(:,k);
                Q(:,k)=s*Q(:,i)+c*Q(:,k);
                Q(:,i)=u;
            end
        end
    end
end
```

Dove Givaux è il seguente:

Algorithm 4

```
function [c,s]=givaux(x,y)
if y==0
    c=1;
    s=0;
elseif abs(y)>abs(x)
    t=x/y;
    z=sqrt(1+t^2);
    s=1/z;
    c=t*s;
else
    t=y/x;
    z=sqrt(1+t^2);
    c=1/z;
    s=t*c;
end
```

Classical Gram-Schmidt

L'algoritmo CGS è il seguente:

Algorithm 5

```
function [A,R]=cgs(A)
n=size(A);
if ne(n(1),n(2))==1
    error('Matrice non quadrata');
end
R=zeros(n);

for k=1:n
    R(1:k-1,k)=A(:,1:k-1)'*A(:,k);
    A(:,k)=A(:,k)-A(:,1:k-1)*R(1:k-1,k);
    R(k,k)=norm(A(:,k));
    A(:,k)=A(:,k)/R(k,k);
end
```

Modified Gram-Schmidt

L'algoritmo MCGS è il seguente:

Algorithm 6

```
function [A,R]=mgs(A)
n=size(A);
if ne(n(1),n(2))==1
    error('Matrice non quadrata');
end
R=zeros(n);

for k=1:n
    R(k,k)=norm(A(:,k));
    A(:,k)=A(:,k)/R(k,k);
    R(k,k+1:n)=A(:,k)'*A(:,k+1:n);
    A(:,k+1:n)=A(:,k+1:n)-A(:,k)*R(k,k+1:n);
```

```
end
```

Test degli algoritmi

Errori commessi nella fattorizzazione

Un primo test che è possibile fare è quello di verificare se le matrici Q ed R trovate dagli algoritmi visti sopra effettivamente fattorizzino A.
Il codice del test è il seguente:

Test 1

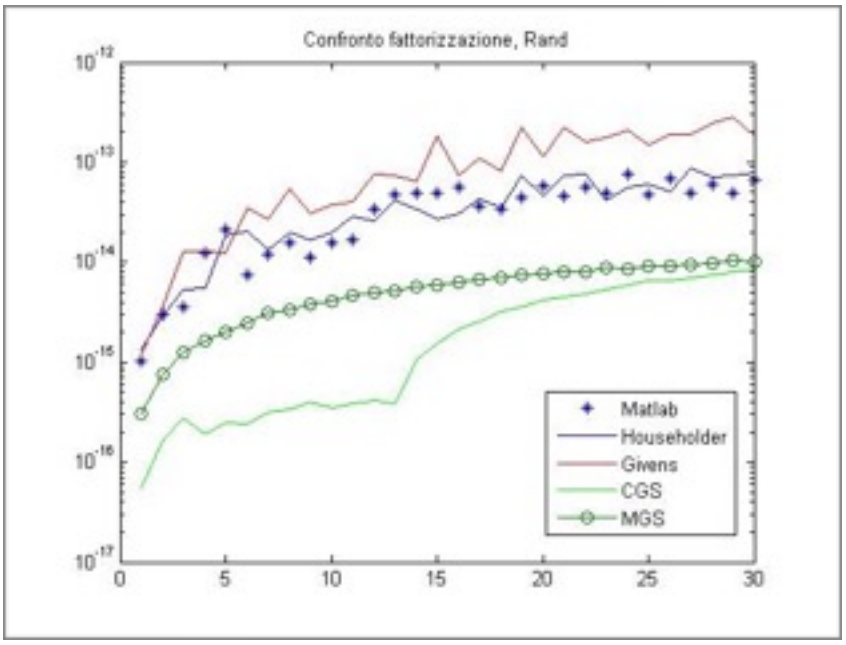
```
%    Confronto tra gli errori commessi dagli algoritmi di
     fattorizzazione QR

nmax=300;
passo=10;
i=0;

err1=zeros(round(nmax/passo),1);
err2=zeros(round(nmax/passo),1);
err3=zeros(round(nmax/passo),1);
err4=zeros(round(nmax/passo),1);
err5=zeros(round(nmax/passo),1);

for n=passo:passo:nmax
    i=i+1;
    A=hilb(n);
    [q1,r1]=qr(A);
    [q2,r2]=qrh(A);
    [q3,r3]=qrg(A);
    [q4,r4]=cgs(A);
    [q5,r5]=mgs(A);

    err1(i,1)=norm(A-q1*r1);
```

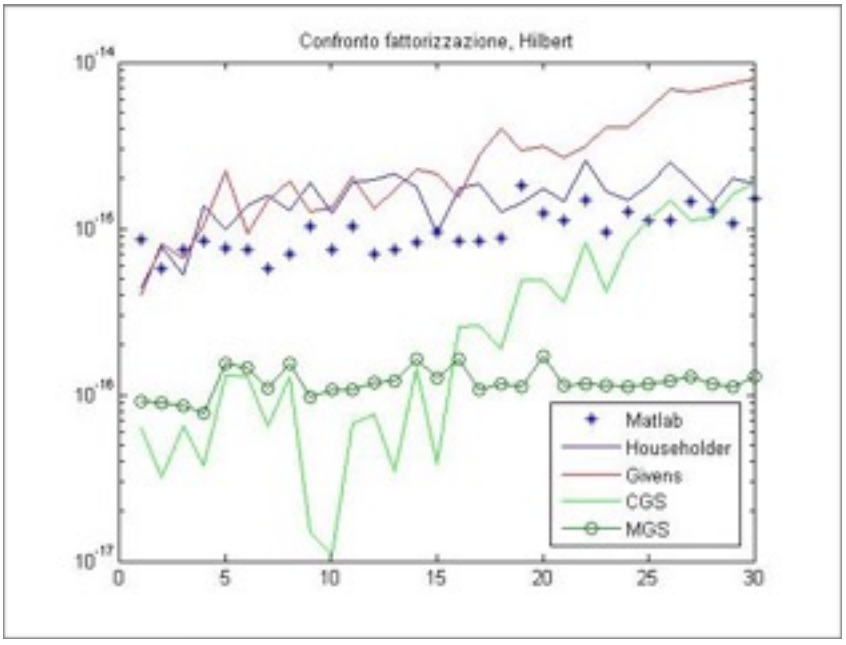
```
    err2(i,1)=norm(A-q2*r2);
    err3(i,1)=norm(A-q3*r3);
    err4(i,1)=norm(A-q4*r4);
    err5(i,1)=norm(A-q5*r5);
end

%    Realizzazione del grafico
x=[1:i]';
semilogy(x,err1,'*',x,err2,'b',x,err3,'r',x,err4,'g',x,err5,'o
-');
legend('Matlab','Householder','Givens','CGS','MGS')
```

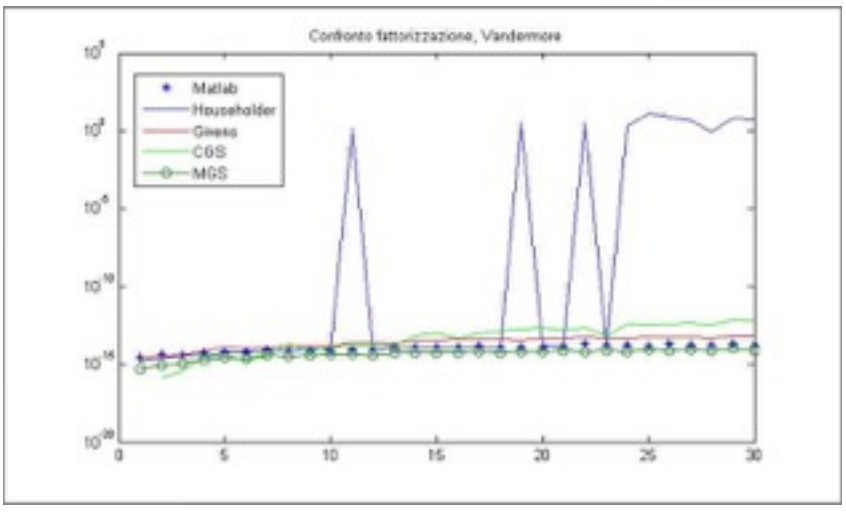
Il confronto è stato fatto con matrici casuali, di Hilbert e di Vandermonde sfruttando i quattro metodi proposti sopra e usando la fattorizzazione QR di Matlab. La decisione di confrontare i metodi usando delle matrici casuali,le matrici di Hilbert e di Vandermonde è data dal fatto di verificare se le fattorizzazioni trovate possano dipendere dal condizionamento della matrice in quanto è noto che quest'ultime abbiano un elevato numero di condizionamento.
Per quanto riguarda il caso delle matrici di Vandermonde, si è prima costruito un vettore casuale, e poi tramite la funzione predefinita di Matlab Vander si è costruita la matrice.

Come si può notare dalle immagini riportate nella pagina successiva, l'errore commesso dagli algoritmi rimane limitato senza mai raggiungere il valore di 10^{-12} nonostante si arrivi a fattorizzare una matrice 300 x 300.
Inoltre, escluso il caso della fattorizzazione di Householder nel caso delle matrici di Vandermonde, sembra che l'errore commesso dalla fattorizzazione non dipenda dal condizionamento della matrice.
In questo test le fattorizzazioni CGS e MGS sembrerebbero essere migliori delle altre, tuttavia si mostrerà in seguito che durante l'esecuzione dell'algoritmo, si perde l'ortogonalità delle colonne della matrice Q.

Un'altra caratteristica che si può notare, è che la funzione predefinita di Matlab qr non è altro che una fattorizzazione di Householder ottimizzata.


Confronto fattorizzazione, Rand
Matlab
Householder
Givens
CGS
MGS


Confronto fattorizzazione, Hilbert
Matlab
Householder
Givens
CGS
MGS


Confronto fattorizzazione, Vandermore
Matlab
Householder
Givens
CGS
MGS

Confronto delle ortogonalità della matrice Q

Di seguito viene riportato lo script riguardante il confronto dell'ortogonalità della matrice Q:

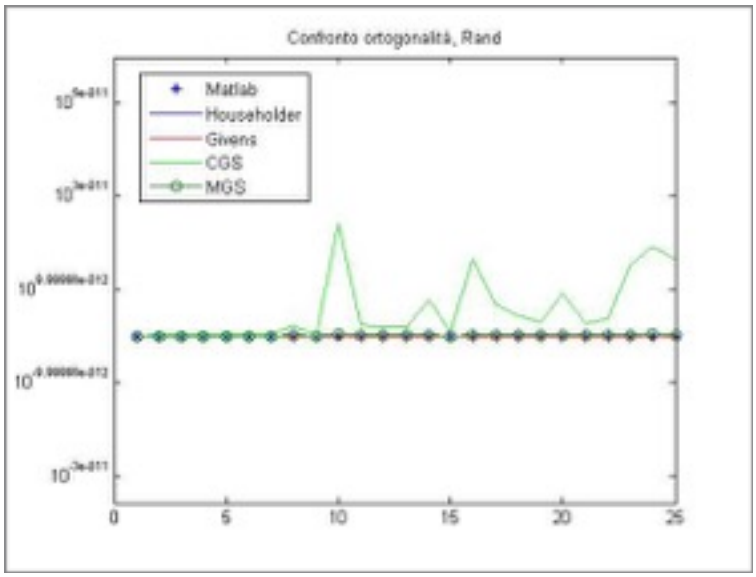
Test 2

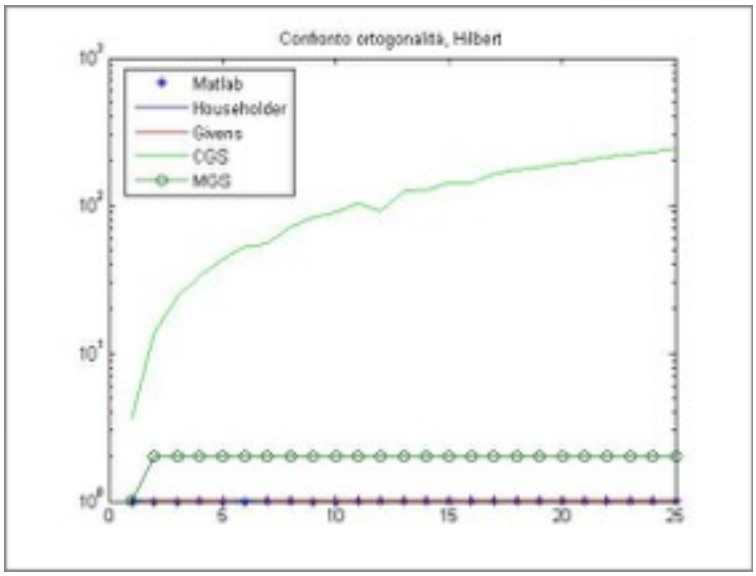
```
%    Confronto delle ortogonalità degli algoritmi di
     fattorizzazione QR

nmax=250;
passo=10;
i=0;

err1=zeros(round(nmax/passo),1);
err2=zeros(round(nmax/passo),1);
err3=zeros(round(nmax/passo),1);
err4=zeros(round(nmax/passo),1);
err5=zeros(round(nmax/passo),1);

for n=passo:passo:nmax
    i=i+1;
    A=rand(n);
    [q1,r1]=qr(A);
    q2=qrh(A);
    q3=qrg(A);
    q4=cgs(A);
    q5=mgs(A);

    err1(i,1)=norm(q1*q1');
    err2(i,1)=norm(q2*q2');
    err3(i,1)=norm(q3*q3');
    err4(i,1)=norm(q4*q4');
    err5(i,1)=norm(q5*q5');
end

%    realizzazione del grafico
x=[1:i];
```

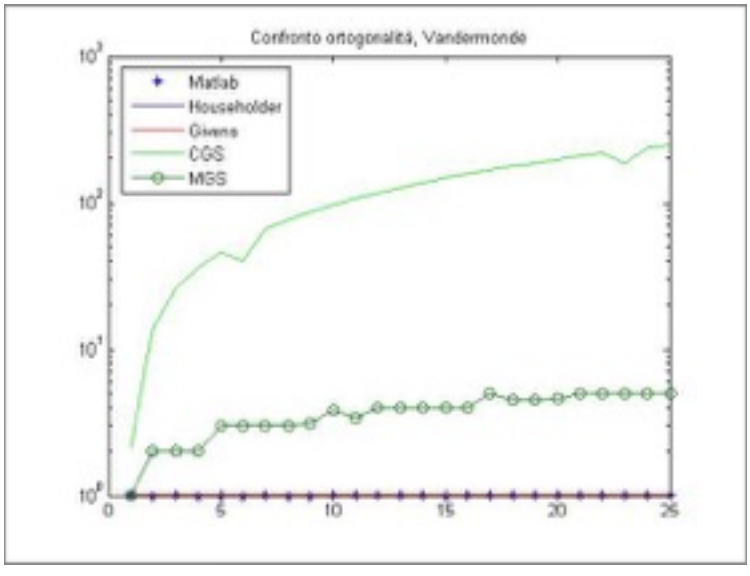
```
semilogy(x,err1,'*',x,err2,'b',x,err3,'r',x,err4,'g',x,err5,'o
-');
legend('Matlab','Householder','Givens','CGS','MGS')
```

Come prima, il test è stato effettuato confrontando i quattro algoritmi implementati nel primo capitolo e l'algoritmo qr di Matlab usando matrici casuali, di Hilbert e di Vandermonde.

Come si può notare dalle immagini riporatte nella pagina successiva, mentre la fattorizzazione di Householder, di Givems, e la fattorizzazione predefinita di Matlab conservano l'ortogonalità, le fattorizzazione di Cholesky no.
O meglio, La MGS è in grado di conservare l'ortogonalità per matrici con un numero di condizionamento basso (si veda la prima immagine). Tuttavia, quando il condizionamento aumenta, progressivamente le colonne della matrice Q non sono più ortogonali tra loro e il che rende l'algoritmo MGS inefficace e il fatto che l'errore commesso dalla MGS sia minore rispetto a quella CGS non è importante in quanto dalla teoria occorre che Q sia ortogonale.


Confronto ortogonalità, Rand
Matlab
Householder
Givens
CGS
MGS
0
5
10
15
20
25


Confronto ortogonalità, Hilbert
Matlab
Householder
Givens
CGS
MGS
0
5
10
15
20
25


Confronto ortogonalità, Vandermonde
Matlab
Householder
Givens
CGS
MGS
0
5
10
15
20
25

Confronto delle soluzioni di sistemi lineari

La fattorizzazione QR è molto utile per la risoluzione di sistemi lineari, per cui nel seguente test si sono confrontate gli errori commessi per la risoluzione di sistemi lineari quadrati.

Test 3

```
%    Confronto delle soluzioni di un sistema lineare tramite
fattorizzazione          QR

nmax=200;
passo=10;
i=0;

err1=zeros(round(nmax/passo),1);
err2=zeros(round(nmax/passo),1);
err3=zeros(round(nmax/passo),1);
err4=zeros(round(nmax/passo),1);
err5=zeros(round(nmax/passo),1);

for n=passo:passo:nmax
    i=i+1;
    A=rand(n);
    sol=ones(n,1);
    be=A*sol;
    delta=10^-14;
    b=be+norm(be)*delta*randn(n,1)/sqrt(n);
    [q1,r1]=qr(A);
    [q2,r2]=qrh(A);
    [q3,r3]=qrg(A);
    [q4,r4]=cgs(A);
    [q5,r5]=mgs(A);

    x1=r1\(q1'*b);
    x2=r2\(q2'*b);
    x3=r3\(q3'*b);
    x4=r4\(q4'*b);
    x5=r5\(q5'*b);
```

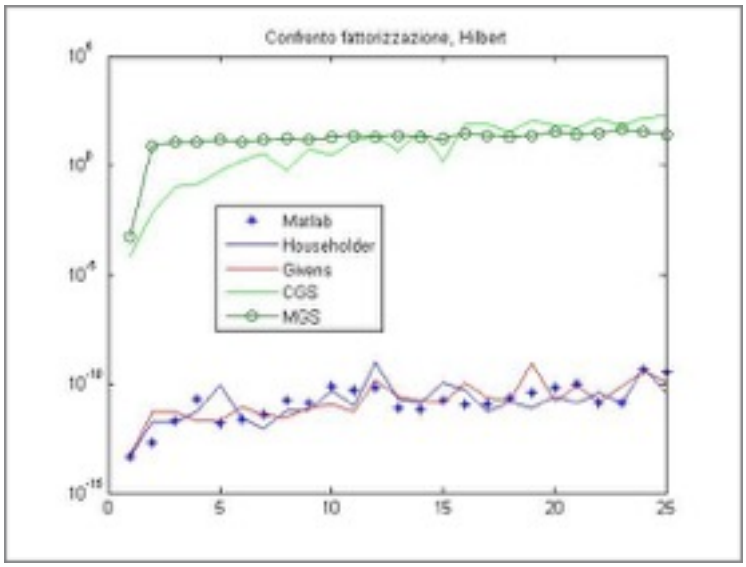
```
    err1(i,1)=norm(A*x1-be);
    err2(i,1)=norm(A*x2-be);
    err3(i,1)=norm(A*x3-be);
    err4(i,1)=norm(A*x4-be);
    err5(i,1)=norm(A*x5-be);
end

%    Realizzazione del grafico
x=[1:i];
semilogy(x,err1,'*',x,err2,'b',x,err3,'r',x,err4,'g',x,err5,'o
-');
legend('Matlab','Householder','Givens','CGS','MGS')
```

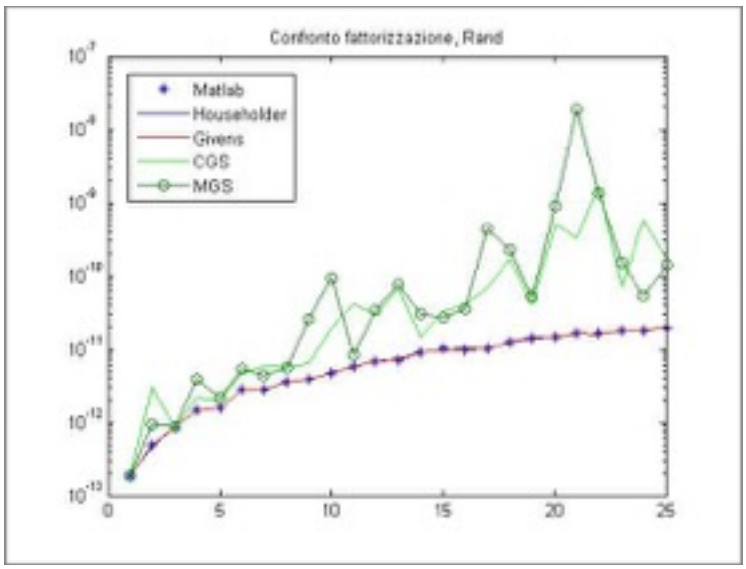
Si è considerato un errore Gaussiano dipendente da un parametro delta pari a 10^{-14} e come matrici delle casuali, di Hilbert e di Vandermonde.
Come si può notare dalle immagini riportate nella pagina successiva, la fattorizzazione QR di Householder e di Givens risultano essere molto efficaci nella risoluzione di sistemi lineari anche nel caso in cui la matrice sia di Hilbert.
Al contrario, la fattorizzazione CGS e MGS risultano inefficaci per la risoluzione di sitemi lineari e ciò è dovuto al fatto che la matrice Q perde l'ortogonalità e quindi la sua inversa tende a non coincidere con la trasposta.

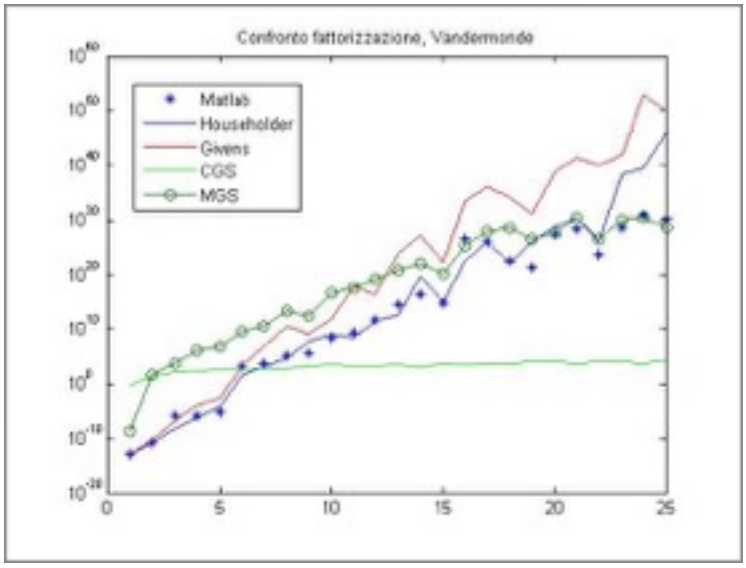
Per quanto riguarda il caso di sistemi lineari le cui matrici associate sono di Vandermonde, qualsiasi fattorizzazione risulta inefficace , il che può essere spiegato dal fatto he il condizionamento sia molto elevato.
In tal caso, un modo per evitare questo problema è quello di fattorizzare la matrice (A |b). Al termine dell'algoritmo l'ultima colonna della matrice R conterrà il vettore Q^T*b. Questo metodo migliora notevolmente la soluzione ed è ingrasso di risolvere anche il caso in cui la matrice sia di Vandermonde.

Usando sempre i sistemi lineari, è possibile fare un secondo test, facendo variare il valore delta contenuto nel test 3.
Nelle immagine riportare in seguito si è fatto il test nel caso -13, -12, -11 e si può notare come le soluzioni tendono a coincidere all'aumentare dell'errore.

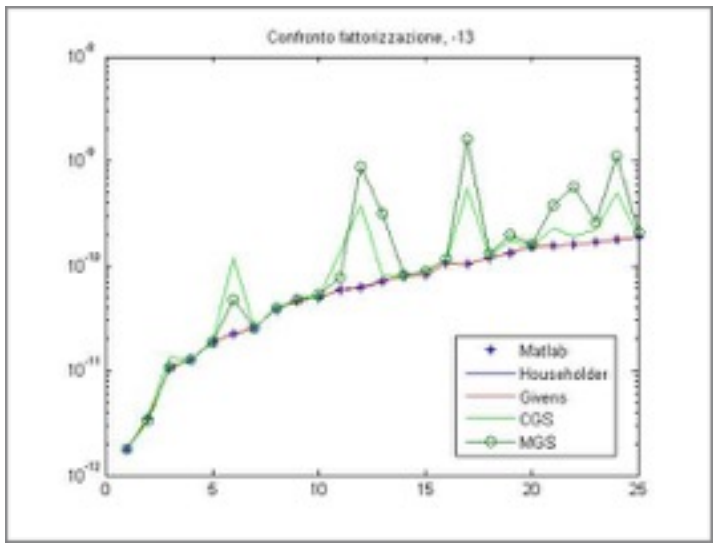
Risultati sistemi lineari

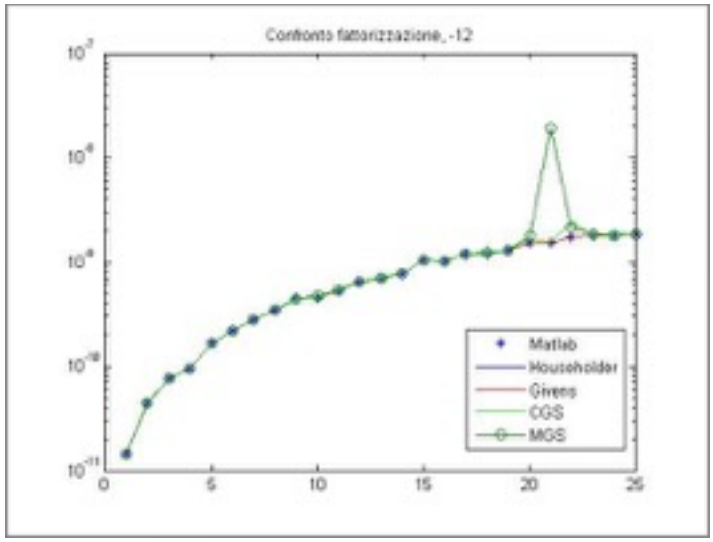


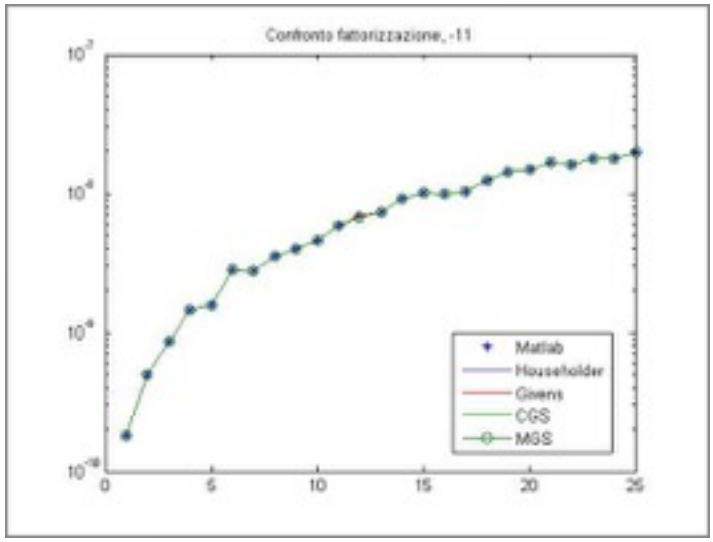




Risultati al variare del delta







Confronto dei tempi di calcolo

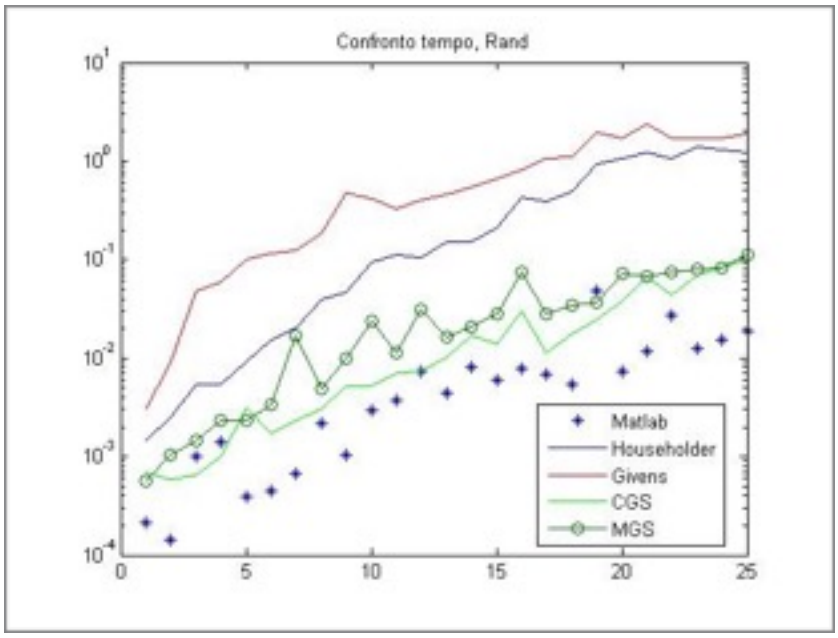
Può essere utile confrontare gli algoritmi proposti in funzione del tempo di calcolo.

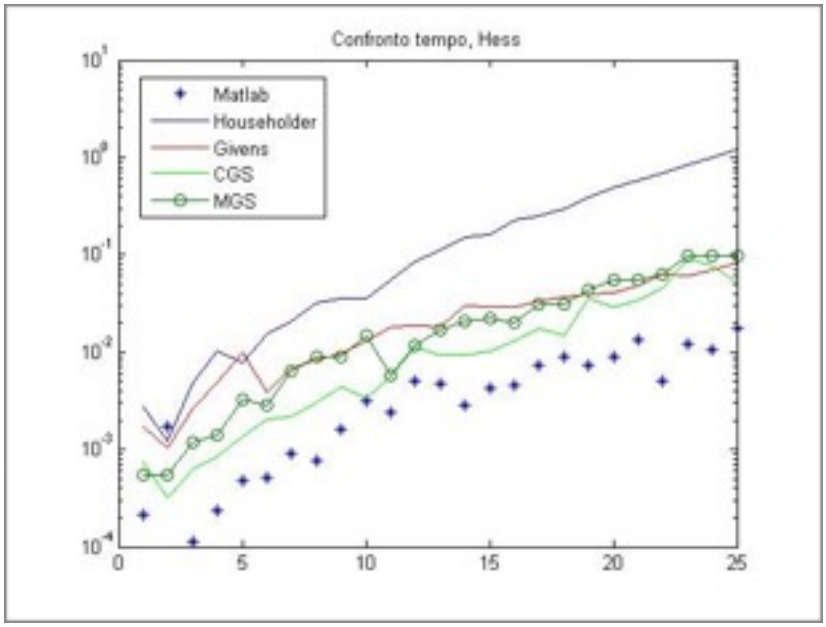
Test 4

```
%Confronto tra il tempo impiegato dagli algoritmi di
fattorizzazione QR

nmax=250;
passo=10;
i=0;

t1=zeros(round(nmax/passo),1);
t2=zeros(round(nmax/passo),1);
t3=zeros(round(nmax/passo),1);
t4=zeros(round(nmax/passo),1);
t5=zeros(round(nmax/passo),1);

for n=passo:passo:nmax
    i=i+1;
    A=rand(n);

    tic
    [q1,r1]=qr(A);
    t1(i)=toc;
    tic
    [q2,r2]=qrh(A);
    t2(i)=toc;
    tic
    [q3,r3]=qrg(A);
    t3(i)=toc;
    tic
    [q4,r4]=cgs(A);
    t4(i)=toc;
    tic
    [q5,r5]=mgs(A);
    t5(i)=toc;
end
```

```
%    Realizzazione del grafico
x=[1:i];
semilogy(x,t1,'*',x,t2,'b',x,t3,'r',x,t4,'g',x,t5,'o-');
legend('Matlab','Householder','Givens','CGS','MGS')
```

In questo caso è stato compiuto il test per matrici casuali e di Hessenberg. La scelta di usare quest'ultimo tipo di matrici è motivato dal fatto che la fattorizzazione di Givens trasforma la matrice A in una triangolare superiore premoltiplicando quest'ultima con opportune matrici di Givens in modo tale da annullare le componenti di A al di sotto della diagonale principale.
Se Alcune di queste sono già nulle la moltiplicazione non viene effettuata. Per cui se la matrice è sparsa, come nel caso delle matrici di Hessenberg, il tempo di calcolo si dovrebbe vedere diminuito.
La matrice di Hessenberg è stata calcolata tramite il comando hess di Matlab.
Come si può notare dalle immagini riportate nella pagina successiva, l'errore atteso da Givens diminuisce significativamente.
Inoltre, dalla teoria è noto che il numero di calcoli eseguiti dalla fattorizzazione di Householder è la metà di quella di Givens ma così non avviene. Ciò è spiegato dal fatto che l'algoritmo non è implementato in maniera migliore possibile.





Risoluzione di un sistema sottodeterminato

La ricerca di una soluzione di un sistema sottodeterminato Ax=b è un problema mal posto, per cui si per portarlo ad uno ben posto occorre imporre che la norma di Ax-b sia minima così come la norma. Nel seguente test si è verificato se la soluzione mediante la fattorizzazione SVD e tramite la funzione predefinita di matlab soddisfino questa richiesta.

Test 5

```
%    test numerici sul metodo dei minimi quadrati applicato ad
     un sistema sottodeterminato
clear
clc

for m=10:10:50
     n=2*m;
     A=rand(m,n);
     x=20*rand(n,1);
     be=A*x;
     [u s v]=svd(A);
     b=be;
     n1(m/10,:)=norm(x);
```

```
    %    Risoluzione mediante SVD
    s(1:m,1:m)=inv(s(1:m,1:m));
    xsvd=v*s'*u'*b;
    diffsvd(m/10,:)=norm(x-xsvd);
    n2(m/10,:)=norm(xsvd);

    %    Risoluzione mediante \
    xm=A\b;
    diffmat(mn/10,:)=norm(x-xm);
    n3(mn/10,:)=norm(xm);
end
```

Si sono trovati i seguenti risultati:

differenza con soluzione SVD	Differenza con soluzione di matlab	dimensione
20,506133934668583	71,9540529891105	10
21,469775541939779	142,1334471544165	20
31,240406942639545	145,3775887472788	30
39,853216888462505	211,1144657648952	40
43,240406942639587	146,5119464500126	50

norma esatta	norma SVD	norma Matlab
56,4161576676515	52,5574097253161	76,5132955448115
75,7761497933502	72,6709959675154	154,2530754431558
83,8854697815655	77,7698147360065	162,3584618985225
107,7591537575971	98,7031024131883	218,3571849526532
110,5539451029527	103,120782957426	189,5407846777145

Come si può notare, la soluzione mediante fattorizzazione SVD è più competitiva rispetto a quella fornita da Matlab. Questo è dovuto al fatto che se il sistema è sottodeterminato, Matlab inserisce tanti zeri quant'è la quantità m-n.

Test sul principio di discrepanza

Se è noto il noise, un valido metodo di regolarizzazione per i la risoluzione di sistemi lineari è il principio di discrepanza. Nel seguente test si è verificato empiricamente questo fatto confrontando con la soluzione esatta data usando la fattorizzazione SVD.

Test 6

```
%    test numerico sul principio di discrepanza
clear
clc

for n=10:10:50
     A=rand(n);
     xe=ones(n,1);
     be=A*xe;
     delta=10^-2;
     b=be+norm(be)*delta*randn(n,1)/sqrt(n);
     e=norm(be-b);

     [u s v]=svd(A);

     %    Scelgo il k
     k=0;
     for i=1:n
         if s(i,i)>e
            k=k+1;
         end
     end

     x=zeros(n,1);
     for i=1:k
         x=x+u(:,i)'*b*v(:,i)/s(i,i);
     end

     y=zeros(n,1);
     for i=1:n
         y=y+u(:,i)'*b*v(:,i)/s(i,i);
     end
```

```
      n1(n/10,:)=norm(x-xe);
      n2(n/10,:)=norm(x-y);
end

a=[1:n/10];
semilogy(a,n2,'b',a,n1,'r');
title('Confronto, Rand');
legend('esatta','discrepanza');
```

Il test è stato fatto su matrici casuali e di Hilbert, in quanto in questi ultimi dovrebbe essere più evidente, e così avviene come si può notare dalle immagini riportate. Si può notare come la conoscenza dell'errore introdotto dai dati risulti vantaggiosa per la soluzione di un sistema lineare, in particolare se è fortemente mal condizionato.