Seminario di algebra lineare numerica
Prof. G. Rodriguez

Studente: Luca Fanni
Mat. : 70/87/43396
CdL: ing. elettrica ed elettronica

Fattorizzazione QR di Givens

Generalità

Una qualsiasi matrice A può essere scomposta nel prodotto di due particolari matrici, cioè fattorizzata: ad esempio si può scomporre nel prodotto di una matrice triangolare inferiore e di una seconda matrice triangolare superiore (fattorizzazione LU), o anche nel prodotto di una matrice ortogonale e di una matrice triangolare superiore. Quest'ultima costituisce la fattorizzazione QR della matrice A ($A=QR$) e si può vedere che tale fattorizzazione si rivela particolarmente utile ed efficiente nella risoluzione di alcuni problemi nell'analisi numerica.
In particolare, se la matrice A di partenza ha dimensioni $m \times n$, la fattorizzazione QR produrrà una matrice Q di dimensioni $m \times m$ e una matrice R di dimensioni $m \times n$.
La matrice Q è ortogonale, con la seguente proprietà:

$$Q^T Q = I$$

cioè la sua inversa è pari alla sua trasposta, dato che il prodotto delle due restituisce l'identità; questa proprietà è importante in quanto normalmente il calcolo dell'inversa di una matrice non è banale, e non è sempre possibile in quanto la matrice di partenza potrebbe essere singolare. Al contrario, trovare l'inversa di una matrice ortogonale è particolarmente semplice in quanto richiede solo di considerare la sua trasposta, senza effettuare calcoli. Ne consegue anche che l'inversa di una matrice ortogonale esiste sempre: ciò è confermato anche dal fatto che il suo determinante può assumere solo i valori 1 e -1, quindi si ha sempre $det(Q) \neq 0$.
Come già accennato, la matrice R è triangolare superiore.
La fattorizzazione QR può essere ottenuta in diversi modi: uno dei metodi più famosi fa uso delle matrici di Householder, ma esiste anche un altro metodo che utilizza le matrici di Givens, ed è proprio con quest'ultimo metodo che si illustrerà come calcolare la fattorizzazione QR.
Qui di seguito si analizzerà l'utilità di tale fattorizzazione per la risoluzione di sistemi lineari.

Introduzione al problema

La fattorizzazione QR di una generica matrice A non singolare (di dimensioni $m \times n$, con $m \geq n$) permette di risolvere il sistema $Ax = b$ tramite la risoluzione di due sistemi più semplici:

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

dove dal primo sistema, essendo Q una matrice ortogonale, si ricava la soluzione intermedia c:

$$c = Q^T b$$

trovata c, essa si utilizza come vettore dei termini noti nel secondo sistema:

$$Rx = c$$

con R matrice triangolare superiore, ricavando la soluzione finale x, che è la soluzione del sistema $Ax = b$ di partenza.
Si preferisce questo approccio, nonostante i sistemi da risolvere siano due e non più uno, in quanto il calcolo di Q^T è immediato (non richiede al calcolatore l'esecuzione di somme e prodotti), richiedendo solo $O(n^2)$ operazioni per la risoluzione del primo sistema, mentre il secondo sistema si risolve per sostituzione all'indietro, che richiede $O\left(\frac{n^2}{2}\right)$ operazioni.
La risoluzione del sistema con un minor numero di operazioni comporta due vantaggi:

- ogni operazione eseguita introduce errore, perciò un minor numero di operazioni introducono un errore più piccolo;
- ogni operazione richiede un certo tempo per poter essere eseguita dal calcolatore, perciò un minor numero di operazioni riduce il tempo di esecuzione totale, velocizzando il calcolo stesso.

Nonostante la fattorizzazione QR richieda un numero doppio di operazioni rispetto alla fattorizzazione LU (cioè rispetto all'algoritmo di Gauss), la fattorizzazione QR è preferita rispetto alla fattorizzazione LU per via della maggiore stabilità: infatti nella fattorizzazione QR il numero di condizionamento rimane invariato:

$$k_2(A) = k_2(R)$$

ciò rende la fattorizzazione QR particolarmente utile nella risoluzione dei sistemi malcondizionati, a differenza della fattorizzazione LU dove il numero di condizionamento può crescere.

Qui di seguito si illustrerà un algoritmo per il calcolo delle matrici Q ed R tramite le matrici elementari di Givens.
La matrice elementare di Givens G_{ij} è nella forma:

$$G_{ij}=\begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ & & \vdots & & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

dove $c=\cos(\theta)$ e $s=\sin(\theta)$.
La matrice in particolare è costruita nel seguente modo, per $\mathrm{i}>\mathrm{j}$:

$$\begin{cases} g_{kk}=1 \ per \ k\neq i,j \\ g_{ii}=g_{jj}=c \\ g_{ji}=-s \\ g_{ij}=s \end{cases}$$

Moltiplicando questa matrice per un vettore colonna si andranno a ricalcolare solo i coefficienti x_i e x_j di tale vettore, lasciando gli altri coefficienti invariati (in particolare tale matrice opera una rotazione del vettore sul piano identificato dai versori e_i ed e_j, per questo la matrice elementare di Givens è nota anche come matrice di rotazione).
L'approccio dell'algoritmo di fattorizzazione mira a calcolare una serie di matrici di Givens ognuna con i coefficienti c ed s calcolati in modo tale da annullare la componente x_j di ogni colonna della matrice A, con la componente x_i facente parte della diagonale principale di A. Dopo l'applicazione di tutte le matrici di Givens, la matrice A è stata trasformata in una matrice triangolare superiore R:

$$G_{n,m}\cdots G_{n,n+1}\cdots G_{2m}\cdots G_{23}G_{1m}\cdots G_{13}G_{12}A=R$$

ma essendo $R=Q^TA$, si può trovare Q:

$$Q=G_{12}^TG_{13}^T\cdots G_{1m}^TG_{23}^T\cdots G_{2m}^T\cdots G_{n,n+1}^T\cdots G_{n,m}^T$$

ottenendo quindi le due matrici Q ed R il cui prodotto costituisce proprio la fattorizzazione di A cercata.
Rimane ora da affrontare il calcolo dei coefficienti c ed s in base alle componenti x_i ed x_j.

Si comincia con il considerare il prodotto del generico vettore colonna x estratto dalla matrice A per la matrice di Givens G_{ij} :

$$G_{ij}\, x = y$$

si ottiene un nuovo vettore colonna y le cui componenti k-esime sono costruite nel seguente modo:

$$y_k = \begin{cases} cx_\mathrm{i} + sx_\mathrm{j} & k = i \\ cx_\mathrm{j} - sx_\mathrm{i} & k = j \\ x_k & k \neq i\,,\, j \end{cases}$$

si nota infatti che per ottenere il vettore y non è necessario eseguire l'intero prodotto matriciale, in quanto la matrice di Givens è strutturata in modo tale da lasciare inalterate tutte le componenti del vettore x che non siano x_i e x_j .
Come detto sopra, si vogliono calcolare i parametri c ed s in modo tale da annullare la componente x_j , quindi si procede ponendo a zero la componente j-esima del vettore y ; questa è solo un'equazione per trovare c ed s , serve un'altra equazione per poter determinare questi parametri. I parametri c ed s della matrice di Givens corrispondono rispettivamente a $\cos(\theta)$ e $\sin(\theta)$, perciò si può utilizzare come seconda equazione la prima relazione fondamentale della trigonometria, ottenendo il sistema di due equazioni in due incognite:

$$\begin{cases} cx_\mathrm{j} - sx_\mathrm{i} = 0 \\ c^2 + s^2 = 1 \end{cases}$$

Risolvendo il sistema si trovano c ed s :

$$c = \frac{x_\mathrm{i}}{\sqrt{x_\mathrm{i}^2 + x_\mathrm{j}^2}} \qquad s = \frac{x_\mathrm{j}}{\sqrt{x_\mathrm{i}^2 + x_\mathrm{j}^2}}$$

Tuttavia, se dal punto di vista matematico queste relazioni sono perfette, dal punto di vista della loro implementazione su un calcolatore queste possono portare a problemi di underflow o overflow. Allora si preferisce riscrivere la funzione al denominatore in un altro modo tale da evitare questi problemi:

$$\sqrt{x_\mathrm{i}^2 + x_\mathrm{j}^2} = \begin{cases} \sqrt{x_\mathrm{i}^2\left(1 + (x_j / x_i)^2\right)} = |x_\mathrm{i}| \sqrt{1 + \left(\dfrac{x_\mathrm{j}}{x_\mathrm{i}}\right)^2} & per\ |x_\mathrm{i}| > |x_\mathrm{j}| \\ \sqrt{x_\mathrm{j}^2\left(1 + (x_i / x_j)^2\right)} = |x_\mathrm{j}| \sqrt{1 + \left(\dfrac{x_\mathrm{i}}{x_\mathrm{j}}\right)^2} & per\ |x_\mathrm{i}| < |x_\mathrm{j}| \end{cases}$$

questa è la funzione nota come "Hypot"; si possono quindi riscrivere le formule per il calcolo del parametri c ed s, usando una variabile temporanea $t=\frac{x_j}{x_i}$ per $|x_i|>|x_j|$ si ottiene:

$$c=\frac{1}{\sqrt{1+t^2}} \qquad s=tc$$

mentre ponendo $t=\frac{x_i}{x_j}$ per $|x_i|<|x_j|$ si ottiene:

$$s=\frac{1}{\sqrt{1+t^2}} \qquad c=ts$$

saranno queste formule ad essere implementate nel programma.

Implementazione dell'algoritmo

Si può ora implementare la prima versione dell'algoritmo, esso si basa sulle considerazioni fatte sino ad ora; come si vedrà in seguito sarà possibile apportare delle migliorie per velocizzare il calcolo rendendo l'algoritmo più efficiente.
La prima versione del programma è la seguente (qrgiv1.m):

```
function [Q, R]=qrgiv1(A)
[m n]=size(A); % m=righe n=colonne
Q=eye(m);
for j=1:n
    for i=(j+1):m
        if A(i,j)~=0
            xi=A(j,j);
            xj=A(i,j);
            G=buildGiv(m,i,j,xi,xj);
            Q=Q*G';
            A=G*A;
        end
    end
end
R=triu(A);
function G=buildGiv(m,i,j,xi,xj)
if abs(xj)>abs(xi)
    t=xi./xj;
    z=sqrt(1.+t.*t);
    s=1./z;
    c=t.*s;
else
    t=xj./xi;
    z=sqrt(1.+t.*t);
```

```
    c=1./z;
    s=t.*c;
end
G=eye(m);
G(i,i)=c;
G(j,j)=c;
G(i,j)=-s;
G(j,i)=s;
```

Il programma è costituito da due parti: la prima funzione implementa l'algoritmo di fattorizzazione vero e proprio mentre la seconda funzione si occupa di generare la matrice di Givens ad ogni passaggio.
Si vede che se la componente a_{ij} è nulla il calcolo della matrice di Givens e dei due prodotti matriciali `Q=Q*G'` e `A=G*A` viene evitato, oltre alla costruzione della matrice di Givens, risparmiando sul numero di calcoli e sul tempo di esecuzione.
Per il calcolo di Q ed R è stato utilizzato il prodotto matriciale classico, sapendo però che calcolando tali prodotti molti di essi sono prodotti per zero (dalla matrice di Givens) e vengono variate solo due componenti per volta, questi prodotti matriciali possono essere rimpiazzati con dei prodotti semplici mirati solo sulle componenti che devono essere effettivamente ricalcolate, tralasciando tutti quei prodotti che avrebbero portato a un risultato invariato o nullo; così facendo si riducono il numero di calcoli da effettuare, riducendo gli errori e il tempo di esecuzione.
Ecco quindi che è stata scritta la seconda versione dell'algoritmo (qrgiv2.m):

```
function [Q, R]=qrgiv2(A)
[m n]=size(A); % m=righe n=colonne
Q=eye(m);
for k=1:n % contatore colonne
    for j=(k+1):m % contatore righe
        if A(j,k)~=0
            xi=A(k,k); % elemento della diagonale
            xj=A(j,k); % elemento j-esimo dalla colonna k-esima
            [c s]=calcCS(xi,xj);
            for i=1:n % contatore ausiliario per il prodotto
                a=A(k,i);
                b=A(j,i);
                A(k,i)=(c*a)+(s*b);
                A(j,i)=(c*b)-(s*a);
                a=Q(i,k);
                b=Q(i,j);
                Q(i,k)=(c*a)+(s*b);
                Q(i,j)=(c*b)-(s*a);
            end
        end
    end
end
R=triu(A);

function [c s]=calcCS(xi,xj)
if abs(xj)>abs(xi)
```

```
    t=xi./xj;
    z=sqrt(1.+t.*t);
    s=1./z;
    c=t.*s;
else
    t=xj./xi;
    z=sqrt(1.+t.*t);
    c=1./z;
    s=t.*c;
end
```

Oltre alle modifiche apportate di cui si è discusso sopra è stata modificata la seconda funzione: ora si limita al solo calcolo dei parametri c ed s, dato che non è più necessario costruire per interno l'intera matrice di Givens.
Si possono apportare ulteriori modifiche, implementando i prodotti dei vettori tramite le funzioni già presenti in Matlab (risparmiando un ciclo for) e facendo in modo di considerare solo le componenti non nulle di questi vettori, cioè tutte quelle componenti che vanno dalla k-esima alla n-esima (quelle che vanno da 1 a k-1 sono infatti nulle, dato che si sta costruendo una matrice triangolare superiore).
Riscrivendo l'algoritmo con queste modifiche si arriva alla terza versione (qrgiv3.m):

```
function [Q, R]=qrgiv3(A)
[m n]=size(A); % m=righe n=colonne
Q=eye(m);
for k=1:n % contatore colonne
    for j=(k+1):m % contatore righe
        if A(j,k)~=0
            xi=A(k,k); % elemento della diagonale
            xj=A(j,k); % elemento j-esimo dalla colonna k-esima
            [c s]=calcCS(xi,xj);
            a=A(k,k:n);
            b=A(j,k:n);
            A(k,k:n)=(c*a)+(s*b);
            A(j,k:n)=(c*b)-(s*a);
            a=Q(:,k);
            b=Q(:,j);
            Q(:,k)=(c*a)+(s*b);
            Q(:,j)=(c*b)-(s*a);
        end
    end
end
R=triu(A);

function [c s]=calcCS(xi,xj)
if abs(xj)>abs(xi)
    t=xi./xj;
    z=sqrt(1.+t.*t);
    s=1./z;
    c=t.*s;
else
    t=xj./xi;
```

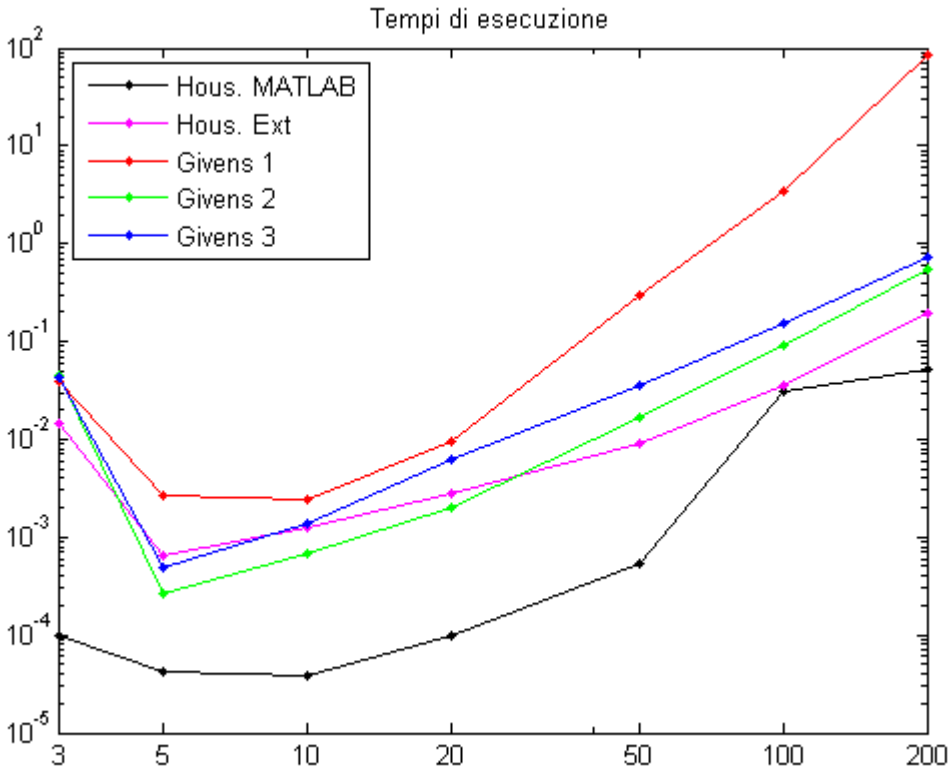
```
    z=sqrt(1.+t.*t);
    c=1./z;
    s=t.*c;
end
```

Avendo ora a disposizione le tre versioni dell'algoritmo, è stato condotto un test per mettere a confronto i tre programmi illustrati tra loro più l'algoritmo di fattorizzazione QR già implementato in Matlab: quest'ultimo è basato sull'algoritmo di Householder, in quanto risulta particolarmente efficace per questo genere di calcolo.
Inoltre, per il confronto delle velocità di esecuzione degli algoritmi, si terrà conto nel test anche di un quinto algoritmo, un'implementazione Matlab dell'algoritmo di Householder fornita dal docente, questo per far vedere la differenza nel tempo impiegato tra l'algoritmo fornito dall'esterno e quello già presente nel kernel di Matlab presente nella libreria LAPACK e scritto in linguaggio Fortran (è per quest'ultimo motivo che l'algoritmo QR di Matlab è molto veloce).
Il test quindi si occupa di generare alcune matrici random di dimensioni via via crescenti, tale matrice A viene fattorizzata da ognuno dei cinque algoritmi, cronometrando il tempo di esecuzione impiegato e calcolando le norme matriciali $\|A-QR\|$ e $\|Q^TQ-I\|$ per ognuno di essi , riportando infine i dati su grafico:

```
Nvect=[3 5 10 20 50 100 200];
temp=size(Nvect);
dimNvect=temp(1,2);
N=Nvect';
ThM=zeros(dimNvect,1);
ThE=zeros(dimNvect,1);
Tg1=zeros(dimNvect,1);
Tg2=zeros(dimNvect,1);
Tg3=zeros(dimNvect,1);
N1hM=zeros(dimNvect,1);
N2hM=zeros(dimNvect,1);
N1hE=zeros(dimNvect,1);
N2hE=zeros(dimNvect,1);
N1g1=zeros(dimNvect,1);
N2g1=zeros(dimNvect,1);
N1g2=zeros(dimNvect,1);
N2g2=zeros(dimNvect,1);
N1g3=zeros(dimNvect,1);
N2g3=zeros(dimNvect,1);
index=1;
for n=Nvect
    A=rand(n);
    n
    disp('Hous. MATLAB')
    tic
    [Q R]=qr(A);
    ThM(index,1)=toc;
    N1hM(index,1)=norm(A-Q*R);
    N2hM(index,1)=norm(Q'*Q-eye(n));
    disp('Hous. Ext')
    tic
    [Q R]=qrhous(A);
    ThE(index,1)=toc;
    N1hE(index,1)=norm(A-Q*R);
```

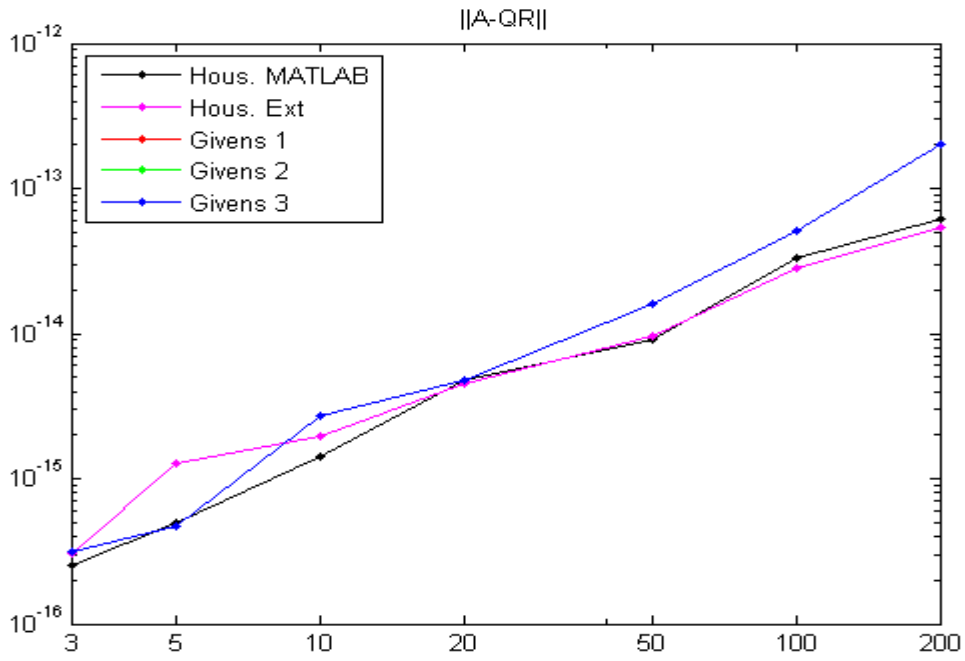
```
    N2hE(index,1)=norm(Q'*Q-eye(n));
    disp('Givens 1')
    tic
    [Q R]=qrgiv1(A);
    Tg1(index,1)=toc;
    N1g1(index,1)=norm(A-Q*R);
    N2g1(index,1)=norm(Q'*Q-eye(n));
    disp('Givens 2')
    tic
    [Q R]=qrgiv2(A);
    Tg2(index,1)=toc;
    N1g2(index,1)=norm(A-Q*R);
    N2g2(index,1)=norm(Q'*Q-eye(n));
    disp('Givens 3')
    tic
    [Q R]=qrgiv3(A);
    Tg3(index,1)=toc;
    N1g3(index,1)=norm(A-Q*R);
    N2g3(index,1)=norm(Q'*Q-eye(n));
    index=index+1;
end
figure(1)
loglog(Nvect,ThM,'k.-',Nvect,ThE,'m.-',Nvect,Tg1,'r.-',Nvect,Tg2,'g.-',Nvect
,Tg3,'b.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('Tempi di esecuzione')
legend('Hous. MATLAB','Hous. Ext','Givens 1','Givens 2','Givens
3','Location','NorthWest')
figure(2)
loglog(Nvect,N1hM,'k.-',Nvect,N1hE,'m.-',Nvect,N1g1,'r.-',Nvect,N1g2,'g.-',N
vect,N1g3,'b.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('||A-QR||')
legend('Hous. MATLAB','Hous. Ext','Givens 1','Givens 2','Givens
3','Location','NorthWest')
figure(3)
loglog(Nvect,N2hM,'k.-',Nvect,N2hE,'m.-',Nvect,N2g1,'r.-',Nvect,N2g2,'g.-',N
vect,N2g3,'b.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('||QtQ-I||')
legend('Hous. MATLAB','Hous. Ext','Givens 1','Givens 2','Givens
3','Location','NorthWest')
```

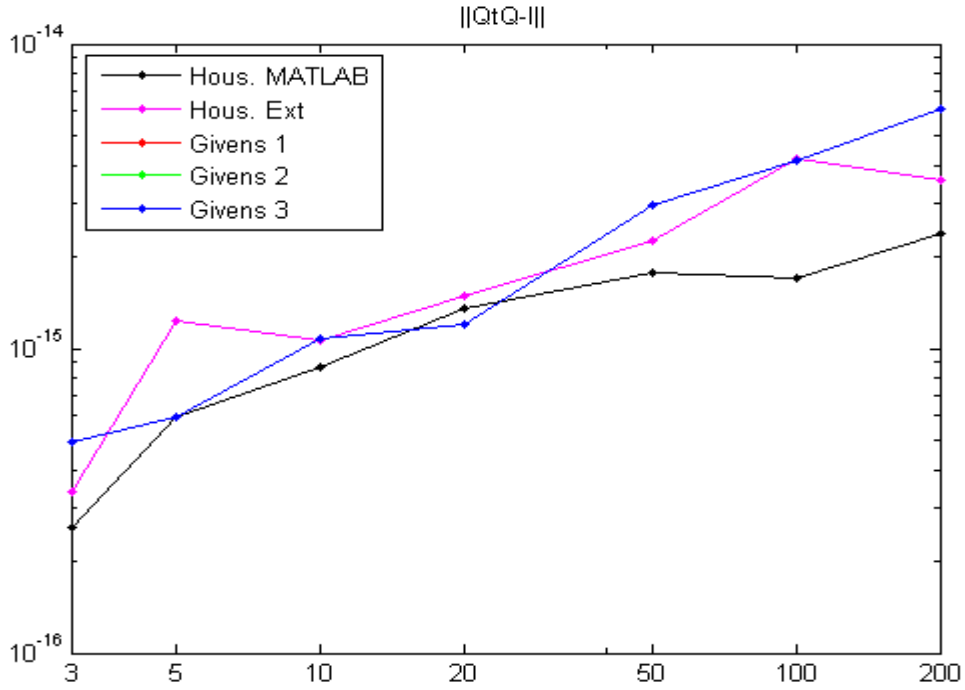
Sono stati ottenuti i seguenti grafici, su un test condotto per matrici da una dimensione minima 3×3 sino a una dimensione massima 200×200; si è preferito di non andare oltre per via del lasso di tempo impiegato dal primo algoritmo di Givens, che per matrici più grandi può diventare molto grande.



Il primo grafico è relativo al tempo impiegato da ogni algoritmo per la fattorizzazione della matrice random. In ascissa è riportata la dimensione della matrice random, sulle ordinate il tempo in secondi. Si vede chiaramente che l'algoritmo di Householder di Matlab è quello mediamente più veloce, mentre tra gli algoritmi di Givens il più veloce risulta essere il secondo (qrgiv2.m), quest'ultimo sarà utilizzato per le prove successive. Si fa notare inoltre che la prima versione di Givens è quella nettamente più lenta, in quanto è la versione meno ottimizzata dove vengono inutilmente eseguite molte più operazioni.
Si vede anche che l'algoritmo di Householder esterno a Matlab presenta dei tempi di calcolo comparabili al secondo algoritmo di Givens; nonostante sia stato implementato con Householder, questo non è risultato veloce come quello già presente in Matlab, dato che quest'ultimo è anch'esso basato su Householder. Questa differenza è dovuta al fatto che quello già presente in Matlab è ulteriormente ottimizzato, essendo inoltre scritto in Fortran, linguaggio di programmazione ottimizzato per i calcoli matematici.

Grafici relativi alle norme:





I tre algoritmi di Givens danno come risultato una norma praticamente identica. E' interessante notare che essa ha lo stesso ordine di grandezza delle norme relative all'algoritmo di Householder, inoltre per matrici di dimensioni medio-piccole l'errore è piuttosto vicino alla precisione della macchina. Ciò significa quindi che l'errore è relativamente contenuto, dando in uscita un risultato attendibile ed utilizzabile.
I grafici inoltre fanno vedere che c'è una leggera differenza tra le norme calcolate con Householder, ma tale discrepanza non ha un grande peso in quanto l'ordine di grandezza è prossimo alla precisione della macchina.

Risoluzione di sistemi lineari

Questo secondo test consiste nell'utilizzare la fattorizzazione QR ottenuta tramite Householder e Givens per risolvere un sistema lineare di n equazioni in n incognite.
In realtà il sistema verrà generato a partire da una soluzione nota, quindi si calcolerà la soluzione con entrambi gli algoritmi e si confronteranno gli errori.
Il vettore soluzione nota è il seguente:

$$e = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}$$

viene quindi generata una matrice random A di ordine n per calcolare il vettore dei termini noti b:

$$b = Ae$$

quindi si fattorizza la matrice A in due matrici Q ed R. Risolvendo il sistema si otterrà la soluzione della macchina x; sostituendo:

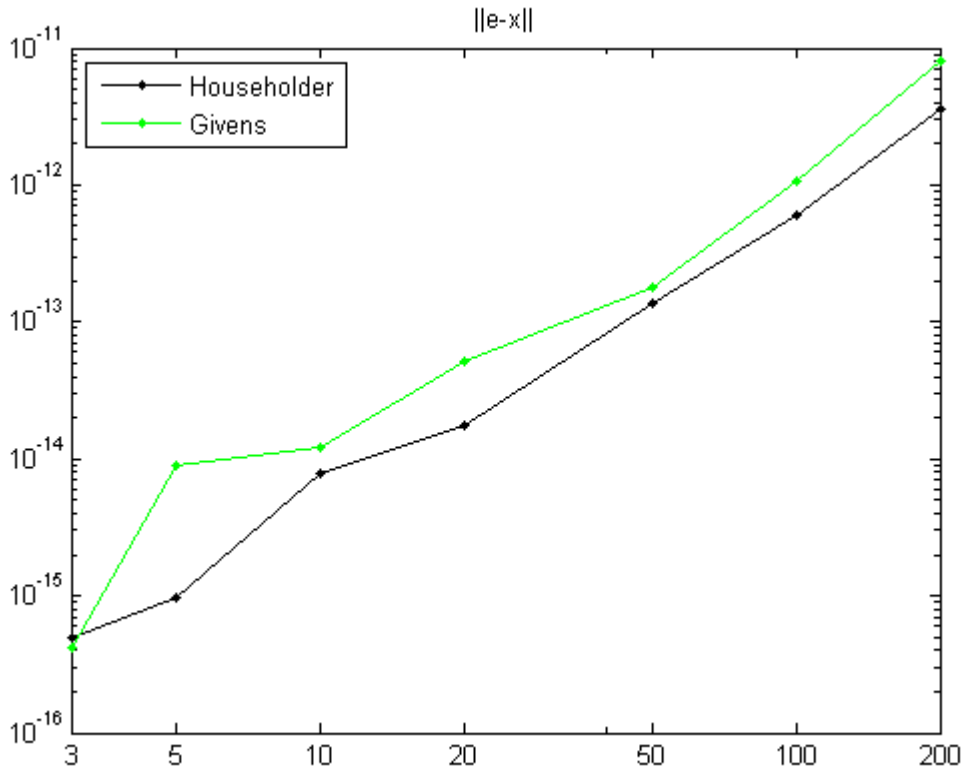
$$b = Ax \rightarrow b = QRx \rightarrow Q^T b = Rx$$

il sistema con l'istruzione `x=R\(Q'*b)`, calcolando il vettore soluzione x. Infine viene calcolata la norma $\|e - x\|$ per vedere l'errore che è stato introdotto dalla fattorizzazione.
Programma del test:

```
Nvect=[3 5 10 20 50 100 200]; % test QR su risoluzione di sistemi nxn
temp=size(Nvect);
dimNvect=temp(1,2);
Nh=zeros(dimNvect,1);
Ng=zeros(dimNvect,1);
index=1;
for n=Nvect
    A=rand(n);
    e=ones(n,1);
    b=A*e;
    [Qh Rh]=qr(A);
    [Qg Rg]=qrgiv2(A);
    xh=Rh\(Qh'*b);
    xg=Rg\(Qg'*b);
    Nh(index,1)=norm(e-xh);
    Ng(index,1)=norm(e-xg);
```

```
    index=index+1;
end
figure(1)
loglog(Nvect,Nh,'k.-',Nvect,Ng,'g.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('||e-x||')
legend('Householder','Givens','Location','NorthWest')
```

Grafico:



In generale l'algoritmo di Householder introduce meno errore rispetto a quello di Givens, ma nonostante ciò gli errori introdotti rimangono comparabili essendo dello stesso ordine di grandezza. Nel complesso l'errore massimo introdotto è dell'ordine di 10^{-11} per un sistema lineare 200×200.

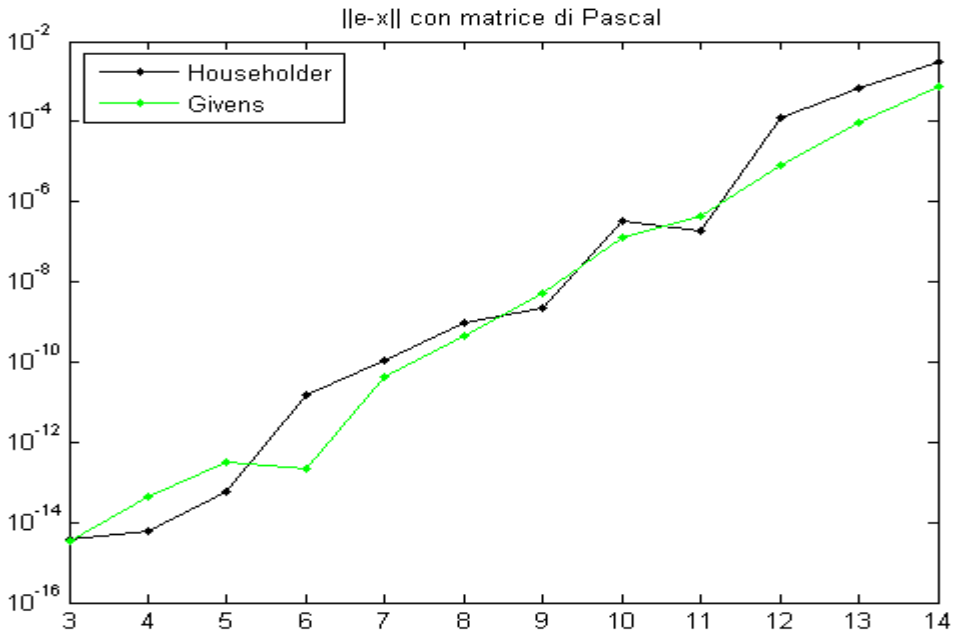
Risoluzione di sistemi compatibili malcondizionati

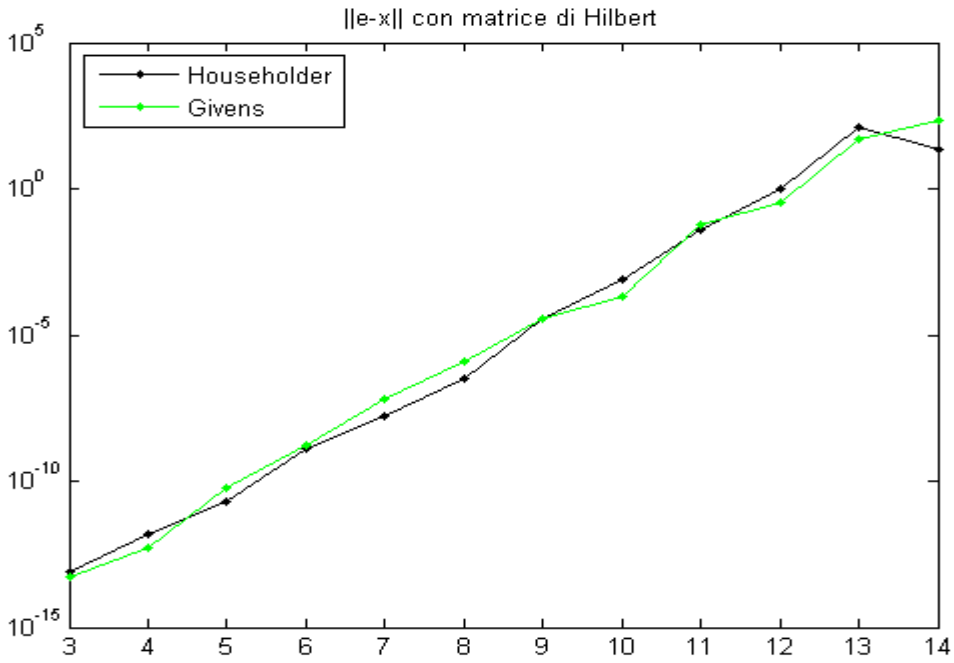
Il terzo test, dal punto di vista computazionale, è identico al secondo, ma questa volta la matrice A non sarà una matrice random ma bensì una matrice volutamente malcondizionata. Lo scopo del test consiste nel mostrare come varia l'errore della soluzione trovata tramite la fattorizzazione QR al variare della dimensione della matrice A, il cui numero di condizionamento $k_2(A)$ tende a crescere molto rapidamente al crescere delle dimensioni di A.

In questo test si farà uso di due particolari matrici note proprio per il loro numero di condizionamento molto elevato, cioè le matrici di Pascal e di Hilbert.
Il test verrà condotto su matrici di dimensioni a partire da 3×3 sino a 14×14.
Programma:

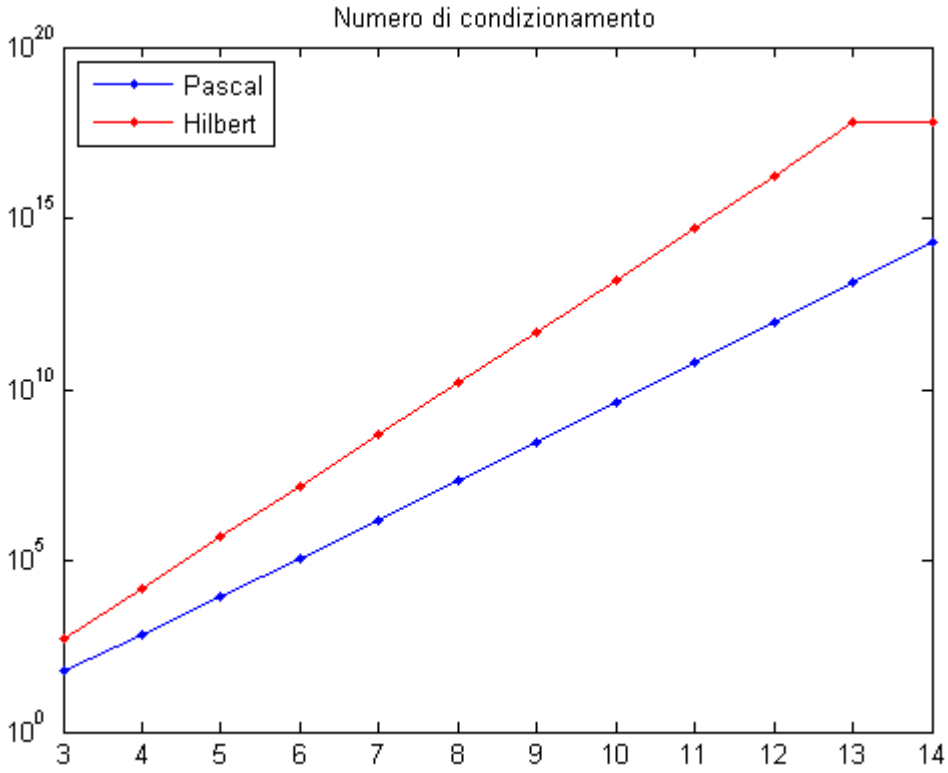
```
Nvect=[3 4 5 6 7 8 9 10 11 12 13 14]; % test QR su risoluzione di sistemi
temp=size(Nvect); % malcondizionati nxn
dimNvect=temp(1,2);
NhP=zeros(dimNvect,1);
NgP=zeros(dimNvect,1);
NhH=zeros(dimNvect,1);
NgH=zeros(dimNvect,1);
KP=zeros(dimNvect,1);
KH=zeros(dimNvect,1);
index=1;
for n=Nvect
    A=pascal(n);
    KP(index,1)=cond(A);
    e=ones(n,1);
    b=A*e;
    [Qh Rh]=qr(A);
    [Qg Rg]=qrgiv2(A);
    xh=Rh\(Qh'*b);
    xg=Rg\(Qg'*b);
    NhP(index,1)=norm(e-xh);
    NgP(index,1)=norm(e-xg);
    A=hilb(n);
    KH(index,1)=cond(A);
    b=A*e;
    [Qh Rh]=qr(A);
    [Qg Rg]=qrgiv2(A);
    xh=Rh\(Qh'*b);
    xg=Rg\(Qg'*b);
    NhH(index,1)=norm(e-xh);
    NgH(index,1)=norm(e-xg);
    index=index+1;
end
figure(1)
semilogy(Nvect,NhP,'k.-',Nvect,NgP,'g.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('||e-x|| con matrice di Pascal')
legend('Householder','Givens','Location','NorthWest')
figure(2)
semilogy(Nvect,NhH,'k.-',Nvect,NgH,'g.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('||e-x|| con matrice di Hilbert')
legend('Householder','Givens','Location','NorthWest')
figure(3)
semilogy(Nvect,KP,'b.-',Nvect,KH,'r.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('Numero di condizionamento')
legend('Pascal','Hilbert','Location','NorthWest')
```

Grafici:





Al crescere della dimensione della matrice malcondizionata cresce anche l'errore della soluzione calcolata con tale matrice rispetto alla soluzione esatta, arrivando anche a numeri dell'ordine di 10^2 con l'utilizzo della matrice di Hilbert; si fa notare inoltre come l'errore sia comparabile tra i due algoritmi utilizzati, per ogni dimensione n.
Ciò è confermato dall'andamento del numero di condizionamento delle due matrici utilizzate:



Essendo l'errore, visto nei due grafici precedenti, legato al numero di condizionamento della matrice utilizzata, se quest'ultimo cresce allora cresce anche l'errore sui dati in uscita. Dal grafico qui sopra si vede che la matrice di Hilbert 14×14 ha un numero di condizionamento dell'ordine di 10^{17}, estremamente elevato. In generale un sistema risolto con una matrice fortemente malcondizionata dà luogo a una soluzione completamente inattendibile: basta una piccola perturbazione in ingresso per avere in uscita una grandissima variazione della soluzione, il numero di condizionamento della matrice A infatti agisce come una sorta di "fattore di amplificazione" della perturbazione in ingresso verso l'uscita.

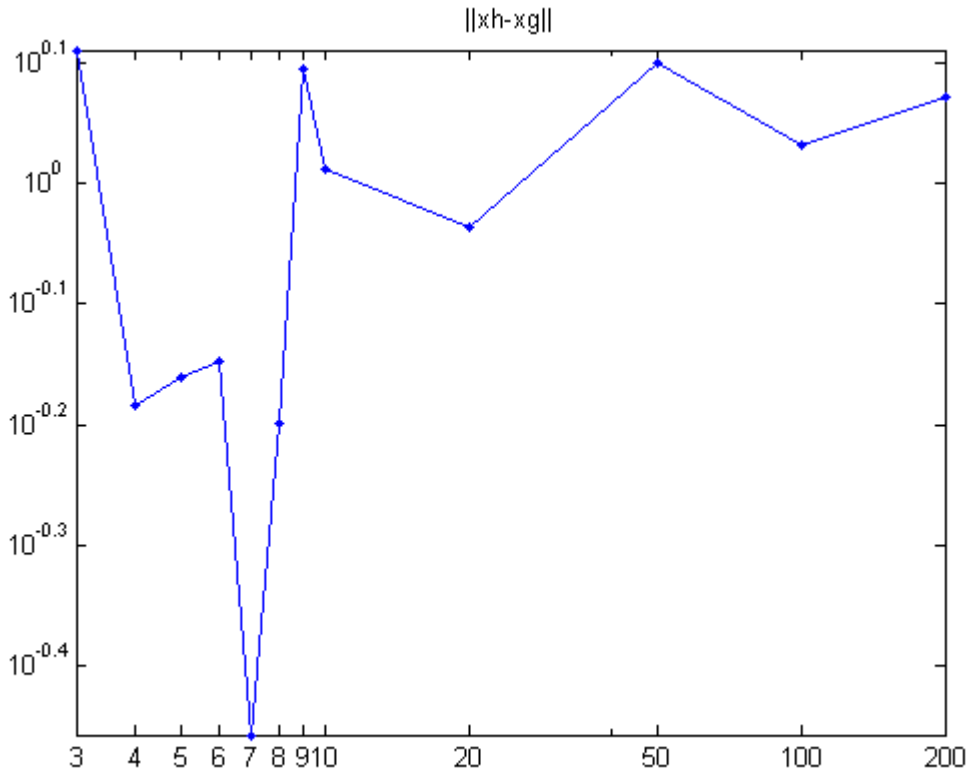
Risoluzione di sistemi incompatibili sovradimensionati

In questo test si metteranno a confronto i due algoritmi di fattorizzazione per la risoluzione di sistemi incompatibili sovradimensionati, cioè sistemi la cui matrice dei coefficienti A è di dimensioni $m \times n$ con $m > n$ (m equazioni in n incognite); il sistema sarà anche incompatibile, imponendo la risoluzione del sistema con una matrice A e un vettore dei termini noti b generati in modo random. Le matrici A avranno una dimensione $2n \times n$; il sistema così generato sarà risolto separatamente tramite le due fattorizzazioni QR, utilizzando come riferimento la soluzione generata dall'algoritmo di Householder per misurare la norma della differenza dei vettori soluzione dei due algoritmi ($||xh - xg||$), dove xh è il vettore soluzione ottenuto tramite Householder mentre xg è il vettore soluzione ottenuto tramite Givens.

Programma:

```
Nvect=[3 4 5 6 7 8 9 10 20 50 100 200]; % test QR su risoluzione di
temp=size(Nvect); % sistemi sovradimensionati incompatibili
dimNvect=temp(1,2);
N=zeros(dimNvect,1);
index=1;
for n=Nvect
    A=rand(2.*n,n);
    b=rand(2.*n,1);
    [Qh Rh]=qr(A);
    [Qg Rg]=qrgiv2(A);
    xh=Rh\(Qh'*b);
    xg=Rg\(Qg'*b);
    N(index,1)=norm(xh-xg);
    index=index+1;
end
figure(1)
loglog(Nvect,N,'b.-')
xlim([Nvect(1,1) Nvect(1,dimNvect)])
get(gca);
set(gca,'xtick',Nvect);
title('||xh-xg||')
```

Grafico:



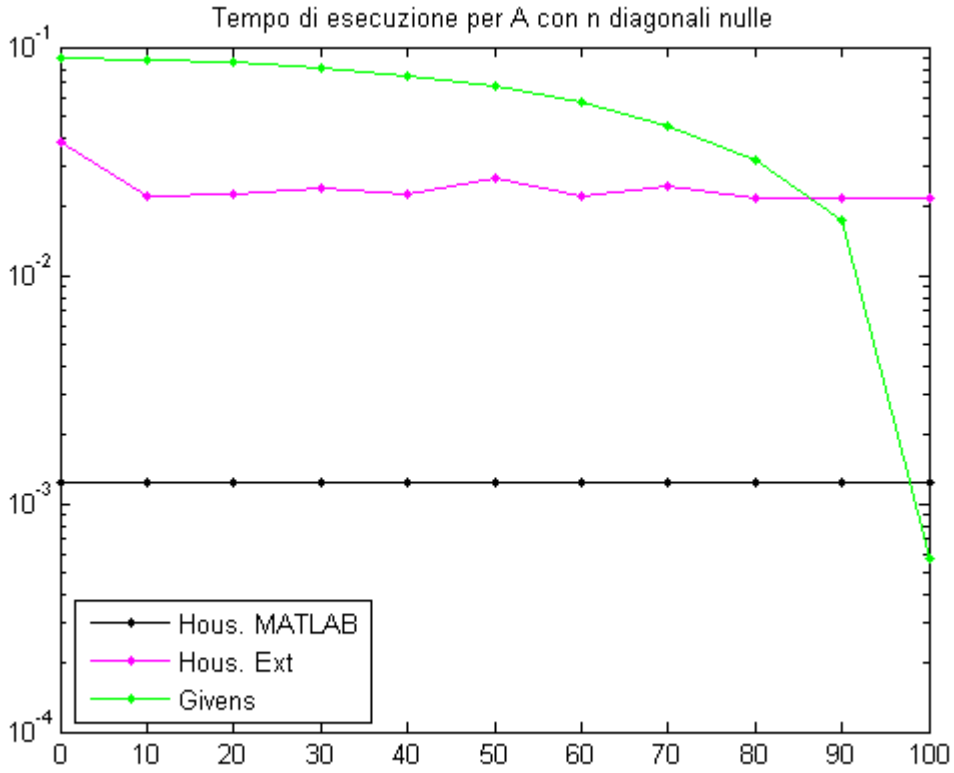
Fattorizzazione QR di matrici con diagonali nulle

In quest'ultimo test si andrà a misure e confrontare il tempo impiegato dai due algoritmi di Householder e da Givens per fattorizzare una matrice random di dimensione fissata, ma con un numero variabile di diagonali nulle, andando da una matrice piena a una matrice triangolare superiore. Si vuole mostrare infatti come la fattorizzazione QR di Givens sia più efficiente di Householder se la matrice A di partenza ha già zeri al di sotto della diagonale principale.
Il test genererà una matrice quadrata random di dimensione fissata; quindi ad ogni passaggio tale matrice sarà data agli algoritmi via via aumentando il numero di diagonali nulle al di sotto della diagonale principale e cronometrando il tempo impiegato per la sua fattorizzazione. Tuttavia, l'algoritmo nativo di Matlab impiega più o meno lo stesso tempo per fattorizzare le varie matrici, esso sarà misurato una sola volta per la matrice piena; infine i tempi misurati per i restanti algoritmi saranno riportati su un grafico per il confronto.
Si fa notare che il test si basa sempre sulla stessa matrice di partenza, dove poi ad ogni ciclo si sostituiranno sempre più diagonali con diagonali nulle: ciò dà luogo a delle matrici da fattorizzare ogni volta diverse, è ovvio quindi che il risultato della fattorizzazione sarà diverso ad ogni ciclo. Ciò nonostante si può sfruttare l'inserimento di zeri sotto la diagonale principale trasformando la matrice di partenza in un'altra che è nella forma di Hessenberg, e procedendo con la fattorizzazione di quest'ultima, questo metodo tuttavia non viene qui illustrato.

Programma:

```
A=rand(101);  % test QR su matrici con n diagonali nulle
tic
[Q R]=qr(A);
temp=toc;
ThM=temp*ones(11,1);
ThE=zeros(11,1);
Tg=zeros(11,1);
for n=0:10
    M=triu(A,-100+10.*n);
    tic
    [Q R]=qrhous(M);
    ThE(n+1,1)=toc;
    tic
    [Q R]=qrgiv2(M);
    Tg(n+1,1)=toc;
end
figure(1)
semilogy([0:10].*10,ThM,'k.-',[0:10].*10,ThE,'m.-',
[0:10].*10,Tg,'g.-')
title('Tempo di esecuzione per A con n diagonali nulle')
legend('Hous. MATLAB','Hous. Ext','Givens','Location','SouthWest')
```

Grafico:



Il tempo impiegato tramite Householder di Matlab è stato misurato una sola volta all'inizio e quindi riportato su tutti i valori in ascissa, mentre ad ogni ciclo è stato misurato il tempo di esecuzione impiegato dagli algoritmi di Householder esterno e Givens. In ascissa sono indicate il numero di diagonali nulle, in ordinata il tempo in secondi.
Le fattorizzazioni QR tramite gli algoritmi di Householder risultano mediamente più veloci di Givens, e impiegano più o meno lo stesso tempo per effettuare la fattorizzazione in tutti i passaggi, tuttavia si nota che all'ultimo passaggio, che corrisponde alla fattorizzazione di una matrice triangolare superiore, Givens ha impiegato meno tempo rispetto ad Householder nativo di Matlab, inoltre Givens è stato più veloce di Householder esterno negli ultimi due passaggi: si può affermare allora che in generale, per matrici piene, per effettuare una fattorizzazione QR conviene usare l'algoritmo di Householder, se invece si deve fattorizzare una matrice che ha già molte diagonali nulle al di sotto della diagonale principale allora può essere più conveniente utilizzare l'algoritmo di Givens dal punto di vista computazionale, sia sul minor tempo impiegato per il calcolo che sul minor numero di operazioni eseguite.
In ultimo, si fa notare ancora una volta come la differente implementazione dello stesso algoritmo (Householder) porti a tempi di calcolo differenti, con una differenza che ammonta a circa un'ordine di grandezza: l'algoritmo di Householder nativo di Matlab è risultato circa 20 volte più veloce di quello esterno.

Biblografia

- G.Rodriguez, *Algoritmi numerici*, Pitagora Editrice Bologna
- Wikipedia, *Givens rotation* e *Hypot*
- Help di Matlab
- Appunti personali