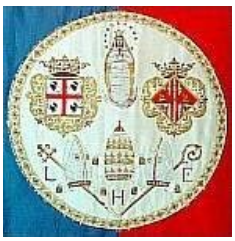


UNIVERSITA' DEGLI STUDI DI CAGLIARI

FACOLTA' DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica

Tesina di Calcolo Numerico 2

Risoluzione di sistemi lineari: confronto fattorizzazioni

Prof. Giuseppe Rodriguez

A cura di:

Patrizia Vacca 39300

Anno accademico 2010/2011

INDICE

1.1 Introduzione

Quando si affronta il calcolo della soluzione di un problema su calcolatore si hanno i seguenti tipi si errore:

- Errore inerente
- Errore analitico
- Errore algoritmico

Errore inerente. A causa dell'errore di rappresentazione sui dati anziché risolvere il sistema lineare

$$Ax = b$$

il calcolatore in realtà risolve il sistema lineare perturbato

$$(A + \delta A)(\underline{x} + \delta \underline{x}) = \underline{b} + \delta \underline{b}$$

dove $(A + \delta A)$ e $(b + \delta b)$ hanno come elementi i numeri macchina con cui sono stati approssimati i corrispondenti elementi con un errore maggiorato dalla precisione di macchina. Si può dimostrare che vale

$$e_x \leqslant \frac{K(A)}{1 - K(A)e_A}(e_A + e_b)$$

dove:

$e_x = ||\delta \underline{x}||/||\underline{x}||$ → errore relativo su $\underline{x}$

$e_A = ||\delta \underline{A}||/||\underline{A}|$ → errore relativo su $\underline{A}$

$e_b = ||\delta \underline{b}||/||\underline{b}||$ → errore relativo su $\underline{b}$

e la quantità

$$||K(A)||||K(A)^{-1}|| \geqslant 1$$

è detta numero di condizionamento della matrice A, che misura la sensibilità del problema agli errori sui dati: se K(A) è elevato l'errore relativo sulla soluzione può essere elevato per quanto e_A ed e_b siano molto piccoli. La stima è comunque

pessimistica in quanto vale per ogni possibile termine noto *b*.
L'errore inerente è indipendente dal metodo utilizzato per la risoluzione.

Errore analitico. I metodi diretti in aritmetica esatta danno soluzione esatta; non si ha quindi errore analitico.
Nel caso dei metodi iterativi occorre invece troncare la generazione della successione di vettori tramite un opportuno criterio di arresto tale da approssimare il limite e dando così origine ad un errore analitico.

Errore algoritmico. L'errore algoritmico è l'errore indotto sul risultato dall'uso dell'aritmetica finita. Anche un'operazione fra due numeri macchina può avere come risultato un numero non di macchina; tale numero risultante necessiterà di essere approssimato con un numero macchina e tale approssimazione avrà ovviamente delle conseguenze sui calcoli successivi. Tale tipo di errore è connesso alla cosiddetta stabilità del metodo di calcolo.

Un elemento importante è rappresentato dal **costo computazionale.**
Il costo computazionale si distingue fra il costo di memorizzazione e il costo in termini di operazioni moltiplicative. Per quanto concerne il costo di memorizzazione la differenza può essere significativa nel caso delle matrici sparse (matrici il cui numero di elementi diversi da zero è O(n) anziché n^2). Infatti le operazioni di fattorizzazione possono far aumentare il numero di elementi non nulli, dando luogo alla necessità di predisporre altro spazio di memoria per la loro memorizzazione; mentre i metodi iterativi visti non alterano la struttura della matrice ed eventualmente può essere caricata in memoria una sola riga di A per volta. Per quanto concerne il costo in termini di operazioni di tipo moltiplicativo si ha che è dell'ordine di $n^3/3$ per la fattorizzazione LU e pari a n^2 per ogni iterazione nei metodi iterativi. Nel caso di convergenza in meno di n passi il vantaggio può essere rilevante.

1.1.1 La stabilità dei metodi diretti per sistemi lineari

Il condizionamento intrinseco del problema, se valutato in termini di errore relativo, è legato al cosiddetto numero di condizionamento

$k_*(A)=\|A\|_*\|A^{-1}\|_*$ della matrice A rispetto alla norma $\|\bullet\|_*$

Se si perturba solo il termine noto b, la soluzione $x+\delta x$ del sistema
$A(x+\delta x) = b+\delta b$ è affetta da una perturbazione relativa

$$\frac{\|\delta x\|}{\|x\|} \leq K(A)\frac{\|\delta b\|}{\|b\|}$$

Nel caso più generale, si ha una espressione più complessa ma conclusioni qualitativamente simili.
La stabilità dei metodi diretti tipo MEG o fattorizzazione LU aumenta decisamente con la pivotazione, ed è migliore per metodi quali le fattorizzazioni QR e di Cholesky. E' comunque difficile risolvere accuratamente sistemi malcondizionati in dimensione alta. La valutazione a posteriori della accuratezza tramite il residuo dipende anch'essa dal condizionamento del sistema
Sistemi di grandi dimensioni, specie se malcondizionati, si risolvono spesso in modo più efficiente ed accurato con metodi iterativi.

1.2 FATTORIZZAZIONE DI CHOLESKY

Sia la matrice A una matrice simmetrica definita positiva[1](vedi Teorema di Cholesky). La fattorizzazione per una matrice di questo tipo risulta semplificata per due ragioni: non è necessario alcun pivoting ed è possibile operare solo sul triangolo inferiore della matrice, essendo quello superiore il trasposto dell'inferiore, essendo la matrice A simmetrica. Pertanto, la fattorizzazione della matrice consisterà nel prodotto di due matrici R e R^T, la prima triangolare inferiore, la seconda trasposta della prima. La determinazione degli elementi della matrice $R=\{R_{ij}\}$ avviene nel seguente modo:

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

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

La matrice deve risultare positiva, altrimenti si darebbe luogo a delle radici r_{jj} complesse che comporterebbero l' impossibilità di realizzazione dell'algoritmo.
Una volta calcolato R si procede alla risoluzione dei due sistemi triangolari:

$$Ry=b \quad e \quad R^T x=y;$$

In matlab la fattorizzazione di Cholesky si realizza con il comando *chol*

1Il teorema di Cholesky afferma che una matrice simmetrica è definita positiva se e solo se esiste una e una sola matrice R triangolare inferiore con elementi diagonali positivi tale che $A=R^TR$.

```
%fattorizzazione di Cholesky
 >>n=5;
 >>A=rand(n);
 >>R=chol(A);
```

Attraverso il comando chol inoltre si può sapere se A è definita positiva attraverso un flag p che risulta essere a zero:

```
[R,p]=chol(A);
```

Come scritto sopra, se A non è definita positiva e la funzione chol viene richiamata con un solo argomento in uscita, verrà visualizzato un messaggio di errore:

```
>>A=[1 2 3;2 5 4;3 4 8]
>>R=chol(A)
??? Error using ==> chol
Matrix must be positive definite.
```

Usando due argomenti in uscita, non viene segnalato alcun errore ma risulta p=3.

```
>> [R,p]=chol(A)
>>p=3
```

Il comando *chol* rappresentato in precedenza può essere anche sostituito da un algoritmo che fa uso di un'altra funzione anziché della funzione chol di matlab. Tale algoritmo può essere implementato per righe o per colonne, a seconda dell'ordine in cui vengono memorizzati gli elementi di una matrice nel linguaggio di programmazione adottato. Io ho scelto di implementare l'algoritmo per colonne:

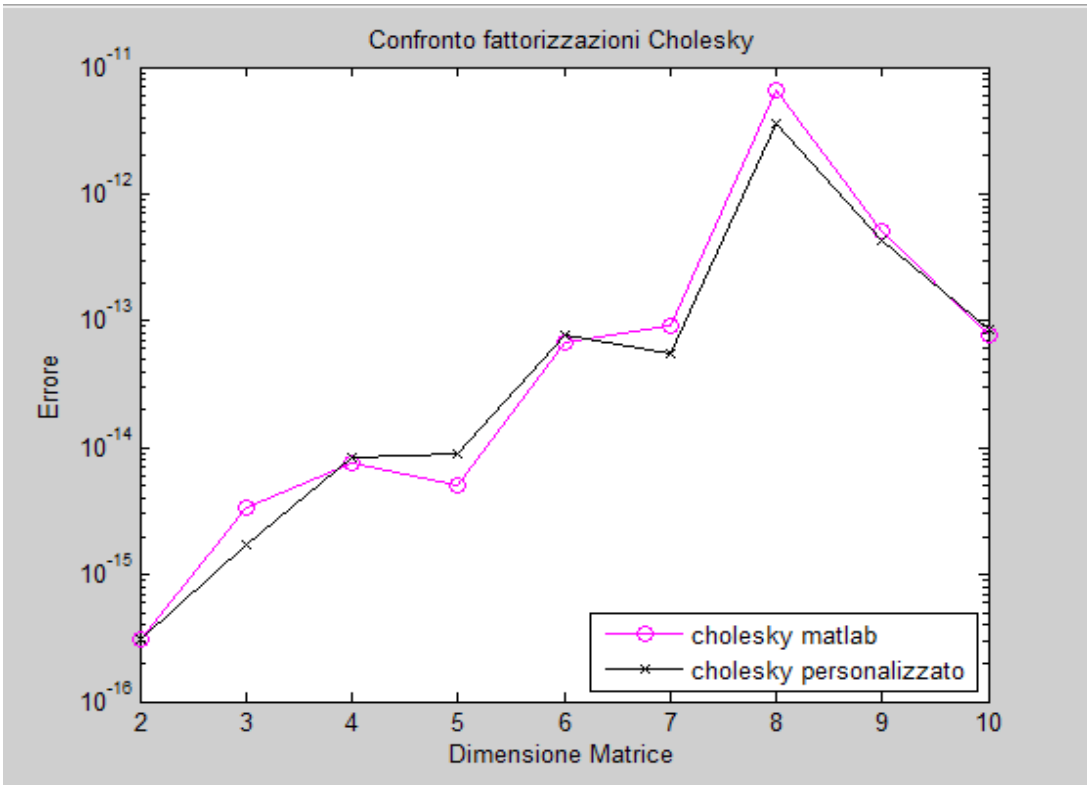
```
function [R] = cholesky(A)          % La funzione “cholesky” ha come input la matrice
                                    %originaria A e restituisce in output la matrice R
                                    %fattorizzata.
```

```
[n,n] = size(A);
R = zeros(n,n);    %La matrice R è inizialmente una matrice di zeri di dimensione n x n.
for j = 1:n
if (A(j,j)-sum(R(1:j-1,j).^2)) < 0          % Se l'argomento tra parentesi è minore di zero la
                                            %matrice A non è definita positiva e si genera un
                                            %messaggio d'errore.
error ('La matrice A 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);     %Si applica la formula scritta sopra per
                                                          %ricavare gli elementi della matrice R in
                                                          %funzione a quelli della matrice A
end
R(j,j) = sqrt(A(j,j)- sum(R(1:j-1,j).^2));
end
end
```

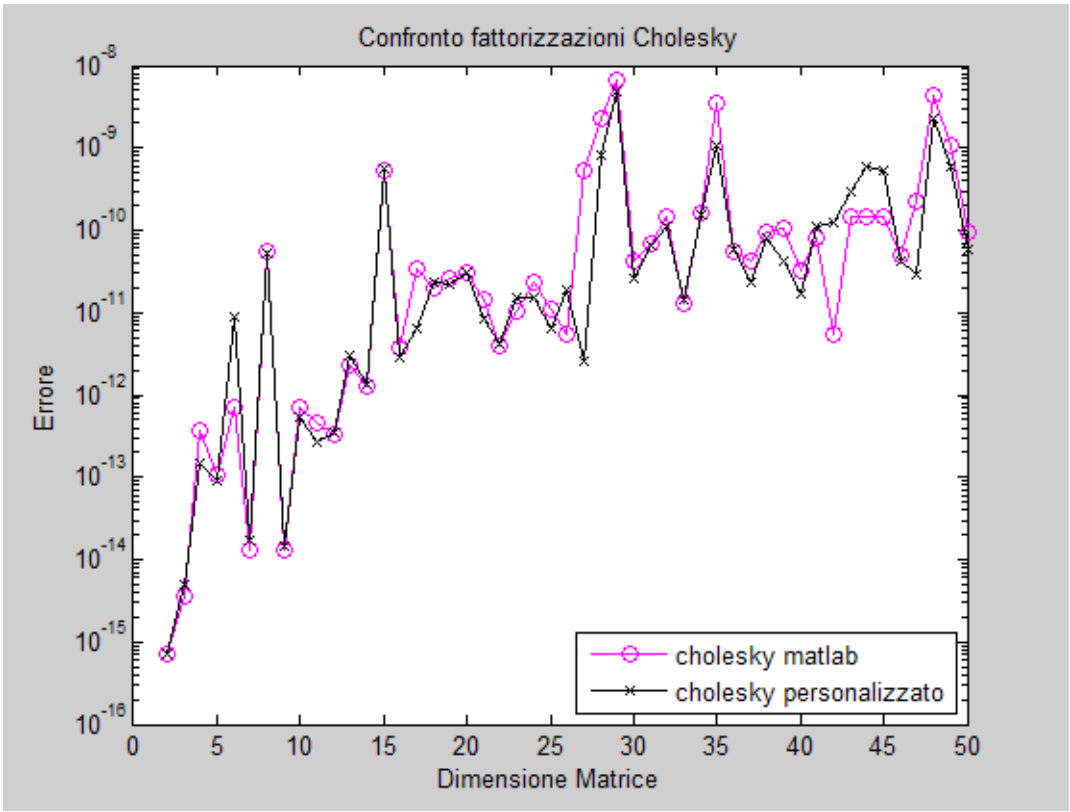
Posso confrontare i due algoritmi sopra scritti, quello da me implementato e quello di Matlab, applicandoli alla risoluzione di un sistema lineare di equazioni. Per il loro confronto calcolo la norma due dell'errore della differenza delle due soluzioni, una data dall'utilizzo del comando Matlab "chol" e l'altra data dall'algoritmo implementato da me, per verificare il comportamento differente dei due algoritmi.
Lo studio è stato effettuato considerando tre diverse grandezze di matrici, in modo da poter verificare il comportamento dei due metodi al variare della dimensione matriciale.

```
%CONFRONTO FATTORIZZAZIONI CHOLESKY: UTILIZZO ALGORITMO MATLAB CHOL E
%ALGORITMO IMPLEMENTATO
```

```
for n = 1:10;
A = rand(n);
A = A'*A;
sol = ones(n,1);
b = A*sol;

%Comando "chol" di matlab

R = chol(A);
y1 = R'\b;
x1 = R\y1;
% Errore usando usando la funzione "chol" di Matlab
errore1(n) = norm(x1 - sol);

%Comando "cholesky" implementato attraverso il mio algoritmo personalizzato

R=cholesky(A);
y2=R'\b;
x2=R\y2;
% Errore usando usando la funzione "chol" di Matlab
errore2(n) = norm(x2 - sol);

end
```

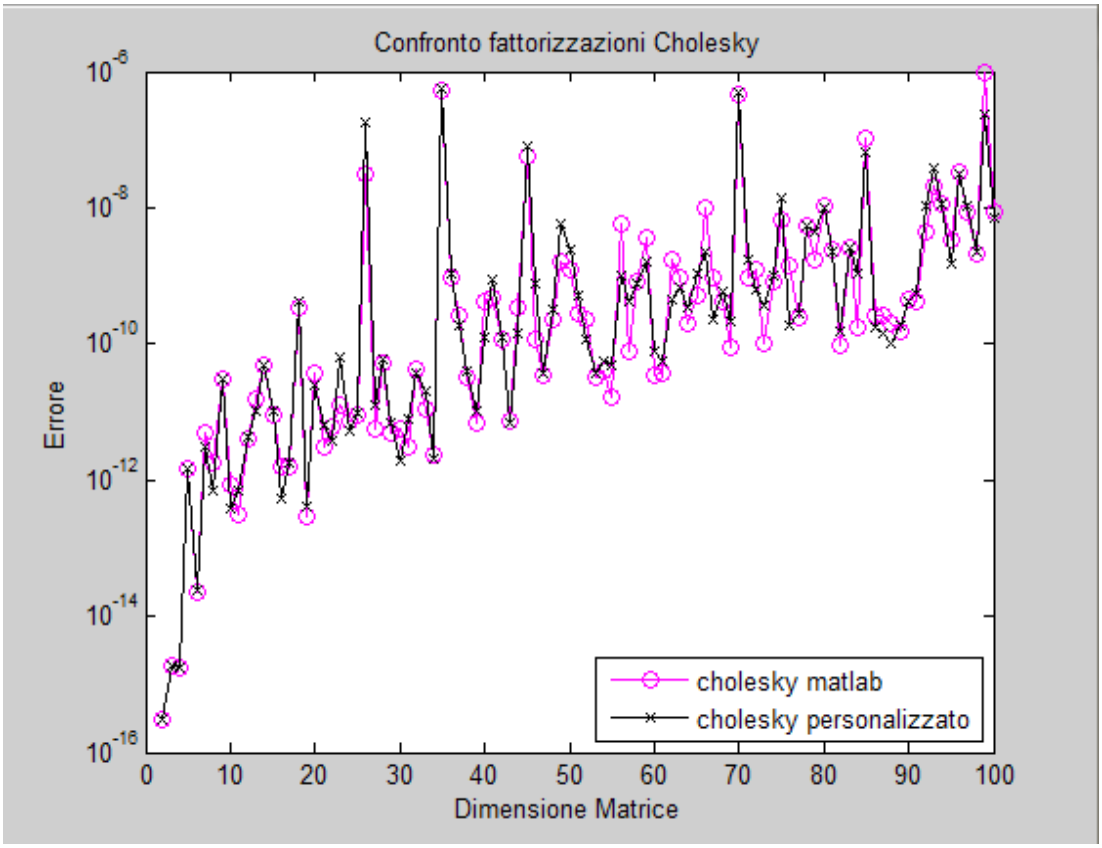


Per la dimensione matriciale che varia tra 1 e 10 si può notare che i due metodi sono entrambi efficienti e non si discostano in maniera significativa tra loro. L'errore è in scala logaritmica e ha una scala di variazione compresa tra 10^{-15} e 10^{-11}.

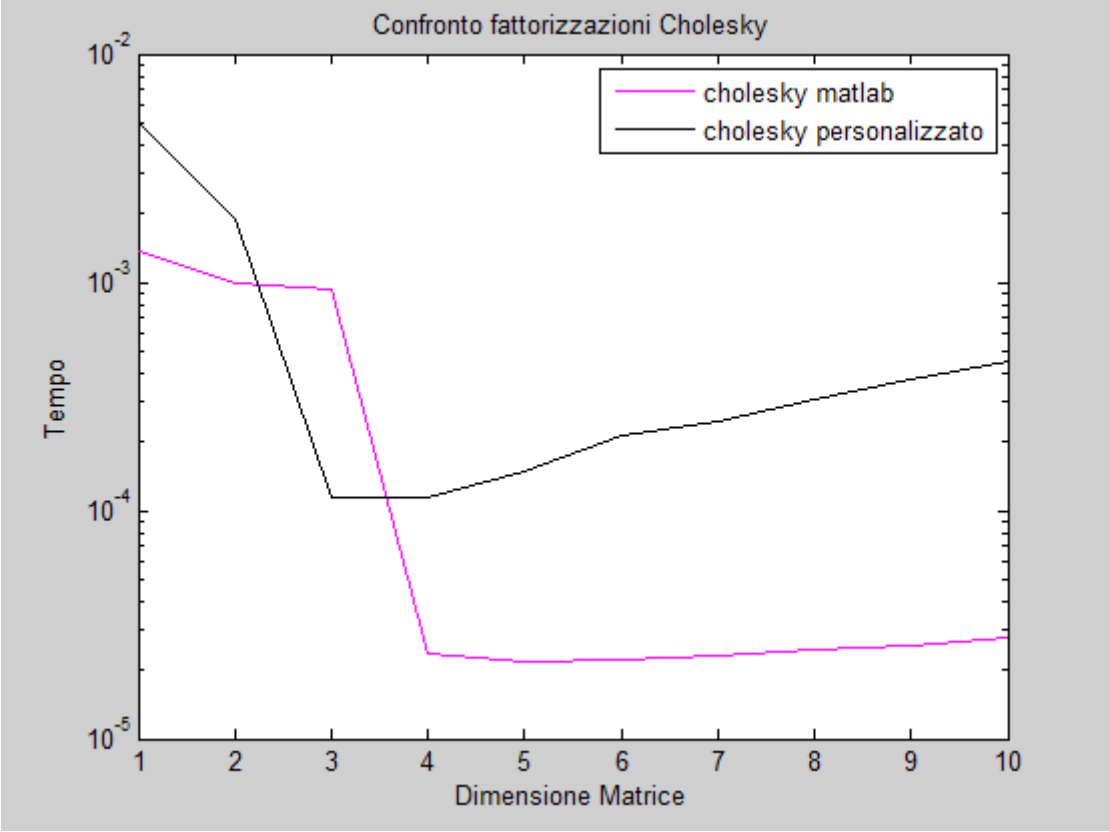


Con una matrice di dimensioni maggiori (1-50), i due metodi rimangono pressoché coincidenti, tuttavia l'errore cresce significativamente, fino a 10^{-8}. E' da notare che in corrispondenza circa dell'ascissa 43 gli errori tra i due metodi differiscono per più di un ordine di grandezza, a vantaggio dell'algoritmo facente uso del comando chol di matlab.

Con una matrice compresa tra 1 e 100 l'errore cresce fino a raggiungere il valore di 10^{-6}.

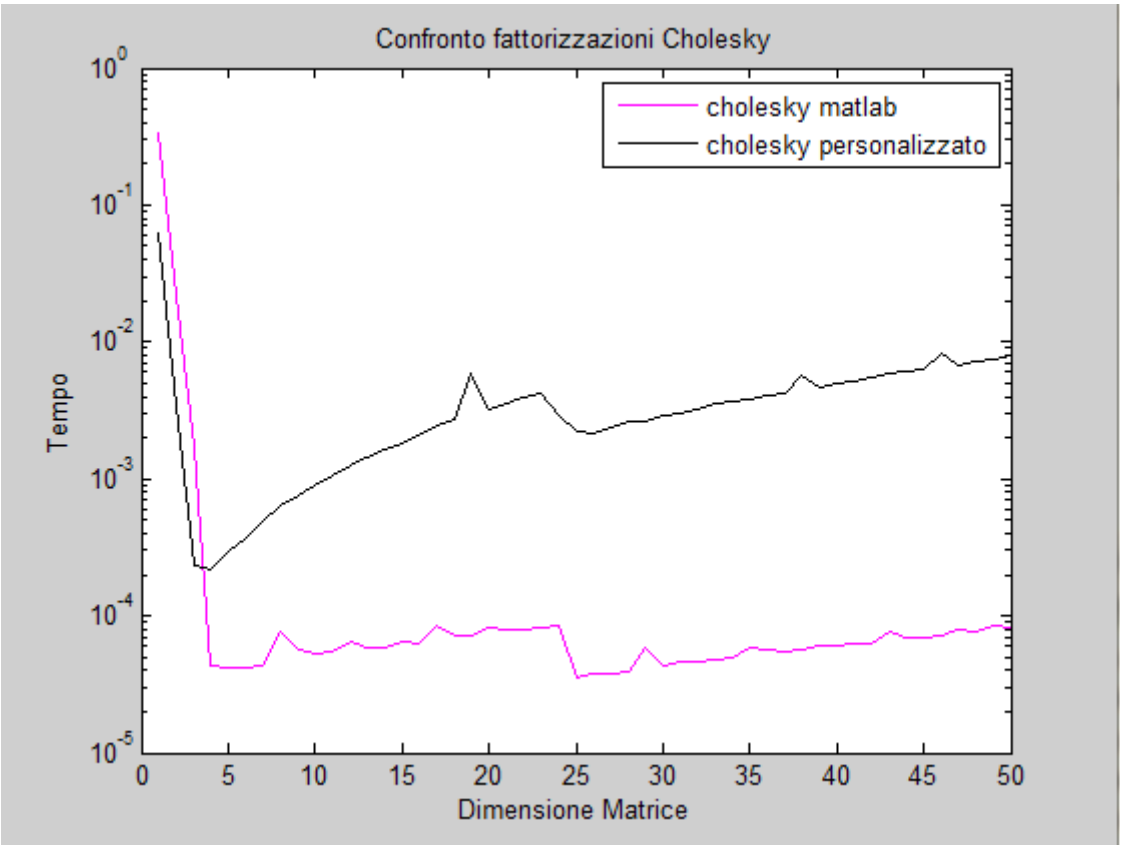


Valutiamo ora i tempi di elaborazione delle soluzioni in funzione delle dimensioni matriciali:

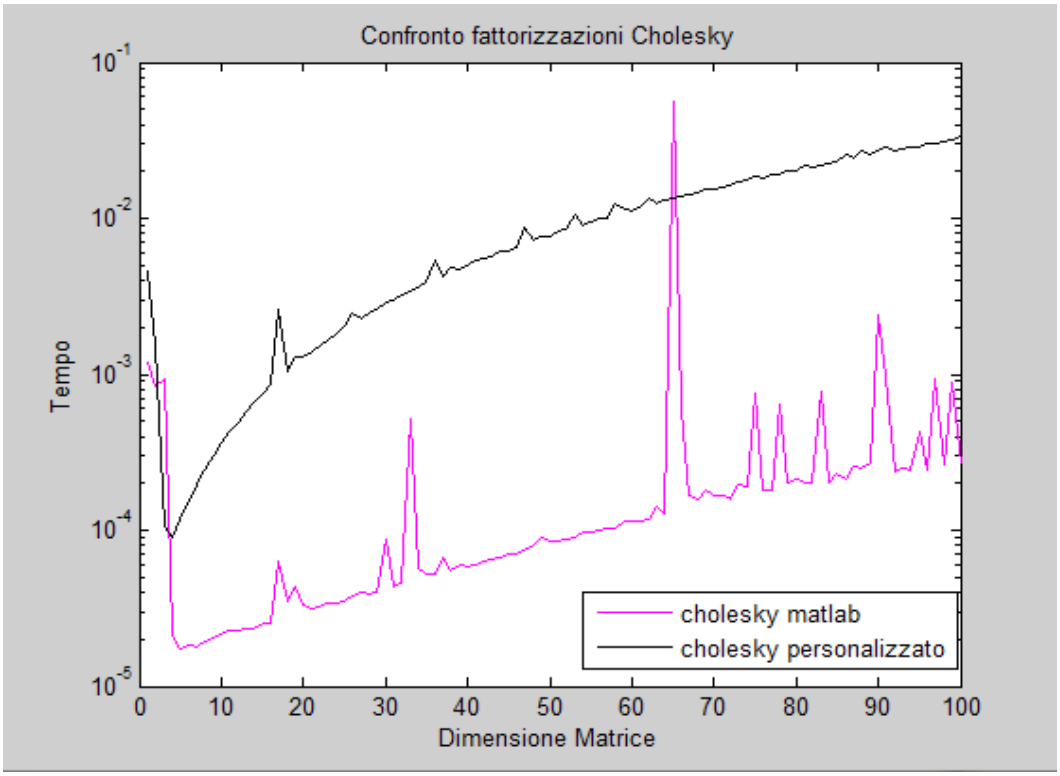


Il metodo di Cholesky implementato con la funzione propria di Matlab risulta molto più efficiente a partire da matrici di piccole dimensioni. Si vede infatti che vi è una differenza tra i due algoritmi, che tende ad aumentare con l'aumentare della dimensione della matrice. Gli andamenti dei due algoritmi sono pressoché gli stessi, tranne che per un andamento leggermente più lineare dell'algoritmo proprio di matlab all'aumentare delle dimensioni.

Con l'aumento delle dimensioni delle matrici, il tempo varia notevolmente tra i due algoritmi implementati, i quali addirittura differiscono di due ordini di grandezza: 10^{-2} sec. per l'algoritmo di matlab e 10^{-4} sec. per l'algoritmo implementato da me.



Anche per matrici comprese tra 0 e 100 i tempi si mantengono molto bassi tuttavia differiscono notevolmente per ordine di grandezza. In questo caso l'algoritmo di Matlab è almeno due ordini di grandezza più veloce.



1.3 LA FATTORIZZAZIONE QR

Un'alternativa alla fattorizzazione di Cholesky è la fattorizzazione QR.
La procedura di fattorizzazione QR è una procedura che consiste nella scomposizione della matrice A in due matrici Q ed R, in cui Q è una matrice ortogonale (ossia tale che $QQ^T = Q^T Q = I$) e R è una matrice triangolare superiore.
Le relazioni che legano gli elementi di A a quelli della matrice prodotto QR sono le seguenti:

$$a_{ij} = \sum_{j=1}^{n} q_{ij} r_{jk} = \sum_{j=1}^{k} q_{ij} r_{jk}$$

in quanto $r_{jk} = 0$ per $j = k+1,...,n$ essendo R triangolare superiore.
Esistono diversi metodi per la scomposizione della matrice in forma QR , in questa tesina mi occuperò della fattorizzazione di Householder e di quella di Givens, mettendone in relazione i punti in comune e le differenze fondamentali.
Una volta ottenuta la fattorizzazione QR con i metodi sopra citati si può passare alla risoluzione del sistema lineare

$$A\underline{x} = \underline{b} ,$$

procedendo allo stesso modo della fattorizzazione LU.
Più precisamente, si ottiene che si devono risolvere i due sistemi lineari

$$Qc = b,$$
$$Rx = c.$$

Il vantaggio dell'introduzione della matrice Q al posto della matrice A originaria, consiste nel fatto che, essendo Q ortogonale, per unicità dell'inversa, $Q^{-1} = Q^t$, cosicché il vettore $\underline{c} = Q^{-1}\underline{b}$ viene ottenuto semplicemente considerando il prodotto matrice-vettore $c = Q^T\underline{b}$ (e non la risoluzione del sistema lineare).
Per quanto riguarda il sistema $R\underline{x} = \underline{c}$, essendo R una matrice triangolare superiore, esso viene risolto con la procedura di risoluzione backward, come nel caso della fattorizzazione LU.

1.3.1 FATTORIZZAZIONE DI HOUSEHOLDER

La fattorizzazione di Householder consiste nella decomposizione della matrice A in due matrici Q ed R a partire dalle matrici elementari di Householder.
Una matrice elementare di Householder reale ha la seguente forma:

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

H è una matrice ortogonale e simmetrica.
Questo metodo presenta alcuni vantaggi pratici, in particolare esso non richiede il calcolo dell'inversa di una matrice triangolare superiore.
Poiché ogni trasformazione di Householder è ortogonale, ci basta far vedere che esiste un prodotto di matrici del tipo $H(\mathbf{x_i}; \mathbf{e_j})$ che trasforma A in una matrice triangolare superiore con entrate sulla diagonale principale tutte positive. A questo punto, l'unicità della fattorizzazione garantisce che quanto abbiamo ottenuto sia effettivamente la decomposizione QR cercata, che è detta matrice di Householder (alternativamente riflessione di Householder, trasformazione di Householder) associata a **w**.

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

Dobbiamo creare una successione di matrici $A^{(i)}$, con i=1,...n in modo che $A^{(n)}$ sia triangolare superiore. Moltiplichiamo la matrice di Householder H_1 a sinistra della matrice $A^{(1)}$ e otteniamo:

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

in cui

$$a1(2) = k1e1,$$

mentre i restanti elementi

$$aj(2) = H_1 aj(1) \quad \text{con } j = 2,...,n.$$

La matrice A(2) ha la seguente struttura:

$$A^{(2)} = \begin{pmatrix} k_1 & a_{12}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix} = \begin{pmatrix} k_1 & v_1^T \\ \mathbf{0} & \hat{A}^{(2)} \end{pmatrix},$$

con 0 vettore colonna e $\hat{A}^{(2)}$ una sottomatrice di dimensione appropriata che andremo a sottoporre alle stesse operazioni a cui abbiamo sottoposto H_1 , dopo averla "orlata" con una riga e una colonna di una matrice identità per farle raggiungere la dimensione n x n e ripetiamo il procedimento come sopra, fino ad arrivare al generico passo i in cui avremo la matrice nella forma:

$$A^{(i)}=\begin{pmatrix} k_1 & \bullet & \cdots & \cdots & \cdots & \bullet \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & k_{i-1} & \bullet & \cdots & \bullet \\ \vdots & & 0 & & & \\ \vdots & & \vdots & & \hat{A}^{(i)} & \\ 0 & \cdots & 0 & & & \end{pmatrix} = \begin{pmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & \hat{A}^{(i)} \end{pmatrix}$$

in cui gli elementi indicati con • non verranno più modificati e la sottomatrice $\hat{A}^{(i)}$ ha la forma:

$$\hat{A}^{(i)}=[\hat{a}_i^{(i)} \quad \hat{a_{i+1}}^{(i)} \quad \cdots \quad \hat{a}_n^{(i)}]$$

A questo punto poniamo:

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

quindi orliamo $\hat{H}_i$ con i-1 righe e colonne della matrice identità fino ad ottenere:

$$A^{(i+1)}=\begin{pmatrix} k_1 & \bullet & \cdots & \cdots & \cdots & \bullet \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & k_i & \bullet & \cdots & \bullet \\ \vdots & & 0 & & & \\ \vdots & & \vdots & & \hat{A}^{(i+1)} & \\ 0 & \cdots & 0 & & & \end{pmatrix}$$

Indichiamo con

$$(H_n H_{n-1} \, H_1)A = R_n$$

dove $H = (H_n H_{n-1} \, H_1)$ è una matrice ortogonale, mentre R_n è una matrice triangolare

superiore con valori positive sulla diagonale principale.
Moltiplicando per $Q = H^T = H^{-1}$ l'uguaglianza di cui sopra si giunge ora alla fattorizzazione

$$A = QR$$

Quanto scritto sopra è valido per matrici quadrate, che sono quelle che ho trattato in questa tesina. Per le matrici rettangolari occorre effettuare un passo in aggiuntivo per azzerare gli elementi della n-esima colonna della matrice A.

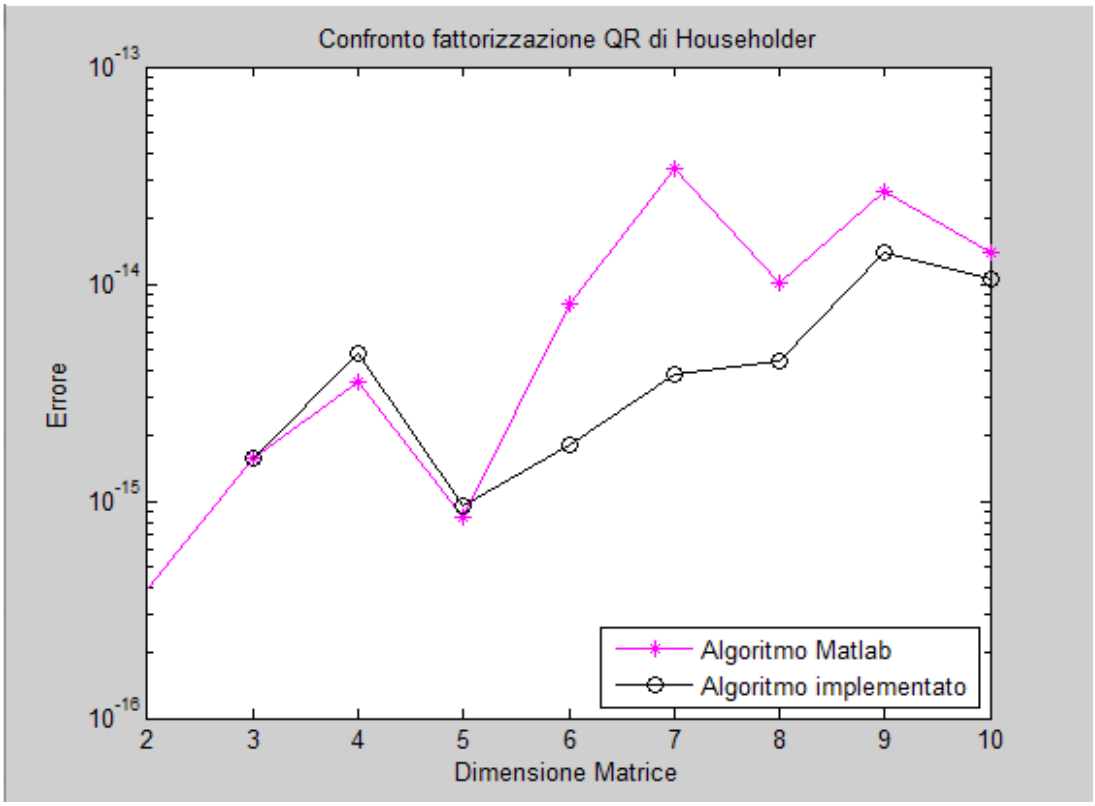
In matlab la fattorizzazione QR di una matrice A mediante l'algoritmo di Householder si ottiene mediante il comando:

```
[Q,R]=qr(A)
```

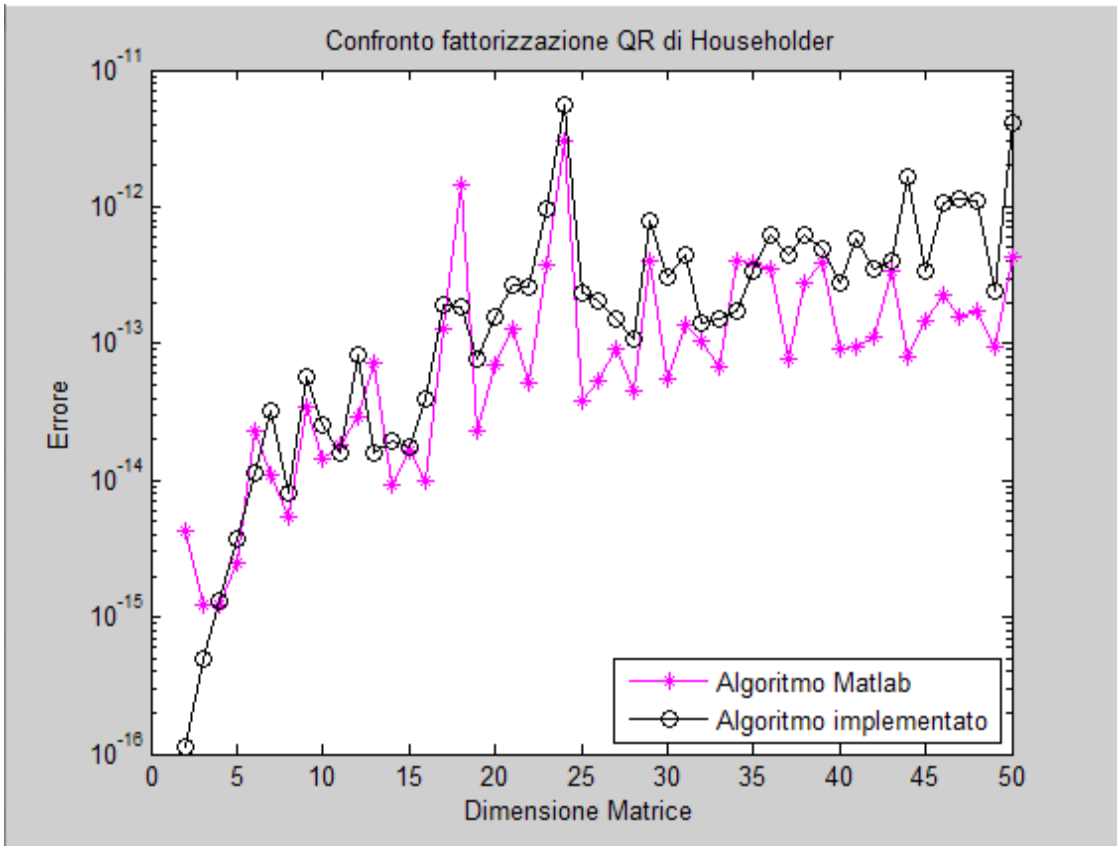
e attraverso questa è possibile trovare la soluzione di un sistema lineare, attraverso i seguenti passi:

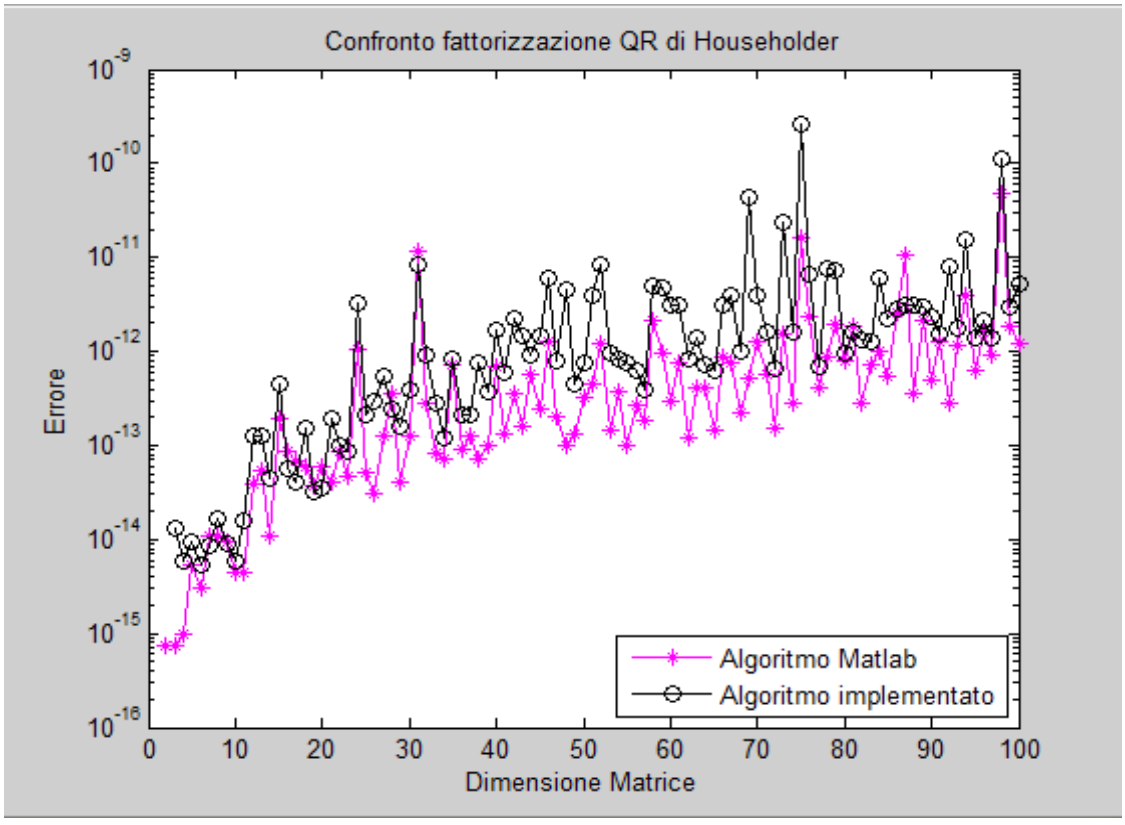
```
[Q,R]=qr(A);
c=Q'*b;
x=R\c;
```

Come già fatto per il caso dell'algoritmo di Cholesky, possiamo sviluppare un algoritmo personale di Householder e lo mettiamo in confronto con quello proprio di matlab. Anche in questo caso consideriamo un sistema di equazioni, e valutiamo l'errore attraverso il calcolo della norma due della soluzione sia nel caso dell'algoritmo personale che nel caso dell'algoritmo implementato da matlab, in funzione della dimensione della matrice. Successivamente ne valuteremo i tempi di elaborazione.
Partiamo da una matrice con n=10. Come si può vedere gli errori sono molto piccoli, compresi in un range tra 10^{-15} e 10^{-13}, tuttavia l'algoritmo personalizzato è affetto da meno errore, quindi è più stabile.

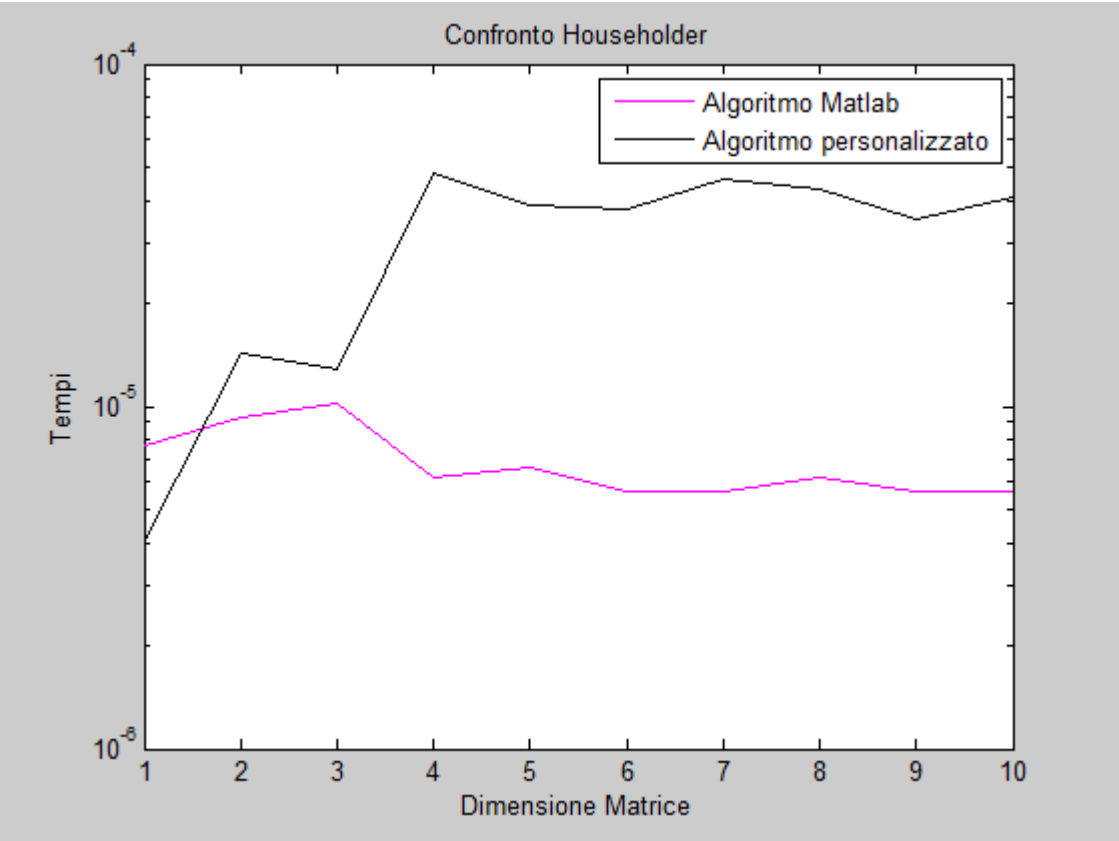


Nel caso di matrici di ordine superiore abbiamo che il range di errore si modifica, da 10^{-16} a $5*10^{-11}$ nel caso di n=50 e da $2*10^{-15}$ a $5*10^{-9}$ nel caso di matrici con n=100, tuttavia il questi ultimi casi l'algoritmo di matlab sembra essere maggiormente stabile, in quanto raggiungono picchi di errore minori rispetto al caso dell'algoritmo personalizzato.


Confronto fattorizzazione QR di Householder
Errore
Dimensione Matrice
Algoritmo Matlab
Algoritmo implementato

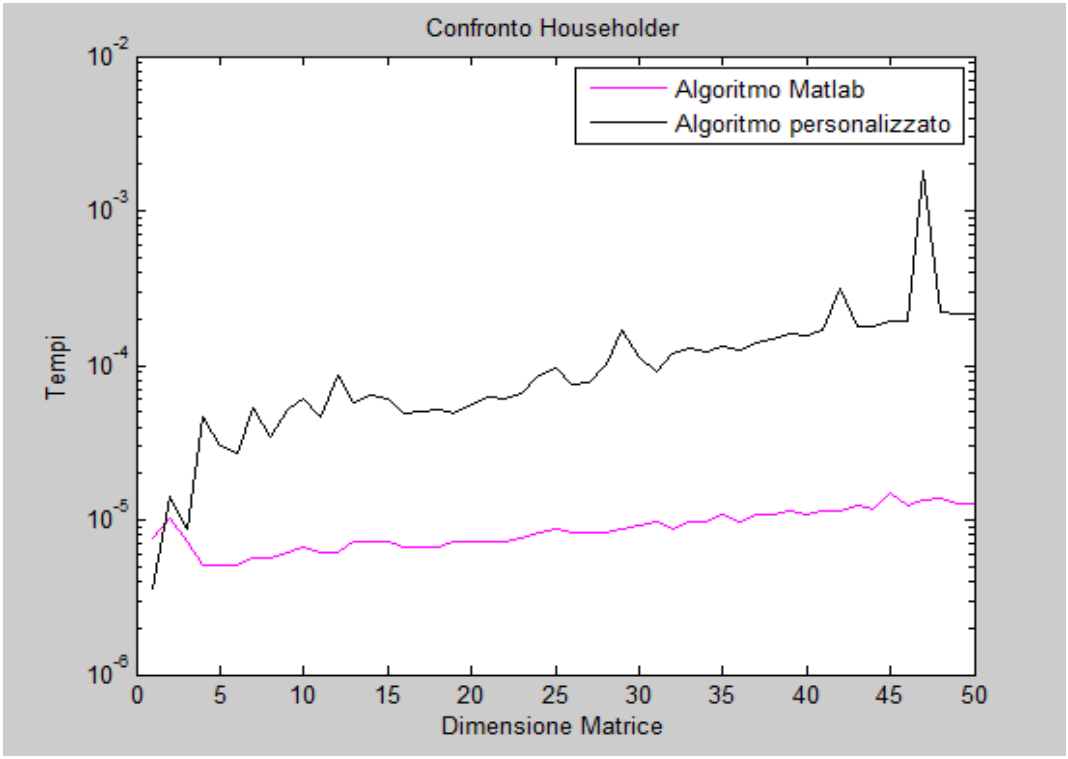

Confronto fattorizzazione QR di Householder
Errore
Dimensione Matrice
Algoritmo Matlab
Algoritmo implementato

Circa i tempi mostriamo i seguenti grafici in funzione della dimensione n della matrice:

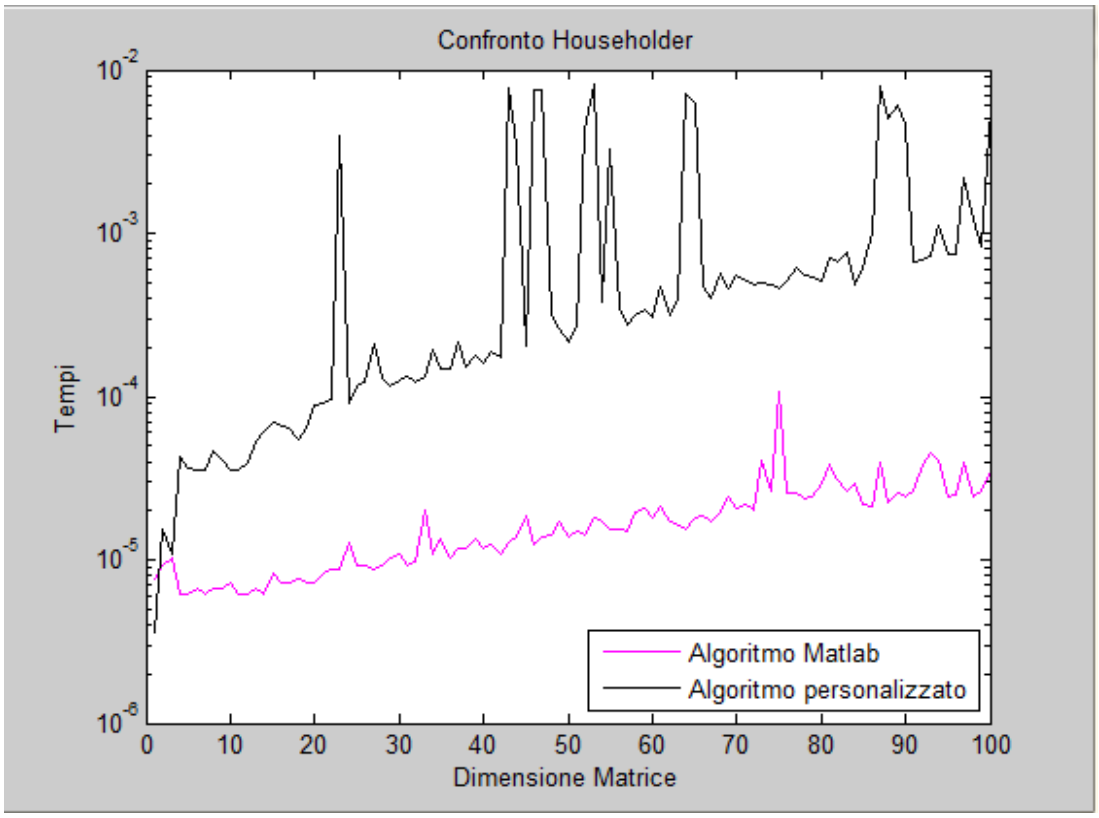


Per n=10 i tempi sono compresi in un intorno di $5*10^{-5}$ sec per l'algoritmo di Matlab, mentre variano anche di un ordine di grandezza per l'algoritmo personalizzato (10^{-5},10^{-4})sec.

Per n superiori si verifica che l'algoritmo di matlab ha dei tempi che rimangono sostanzialmente invarianti rispetto al caso precedente, mentre nel caso dell'algoritmo personalizzato si ha una variazione di circa due ordini di grandezza. L'algoritmo di matlab è dunque significativamente più veloce.

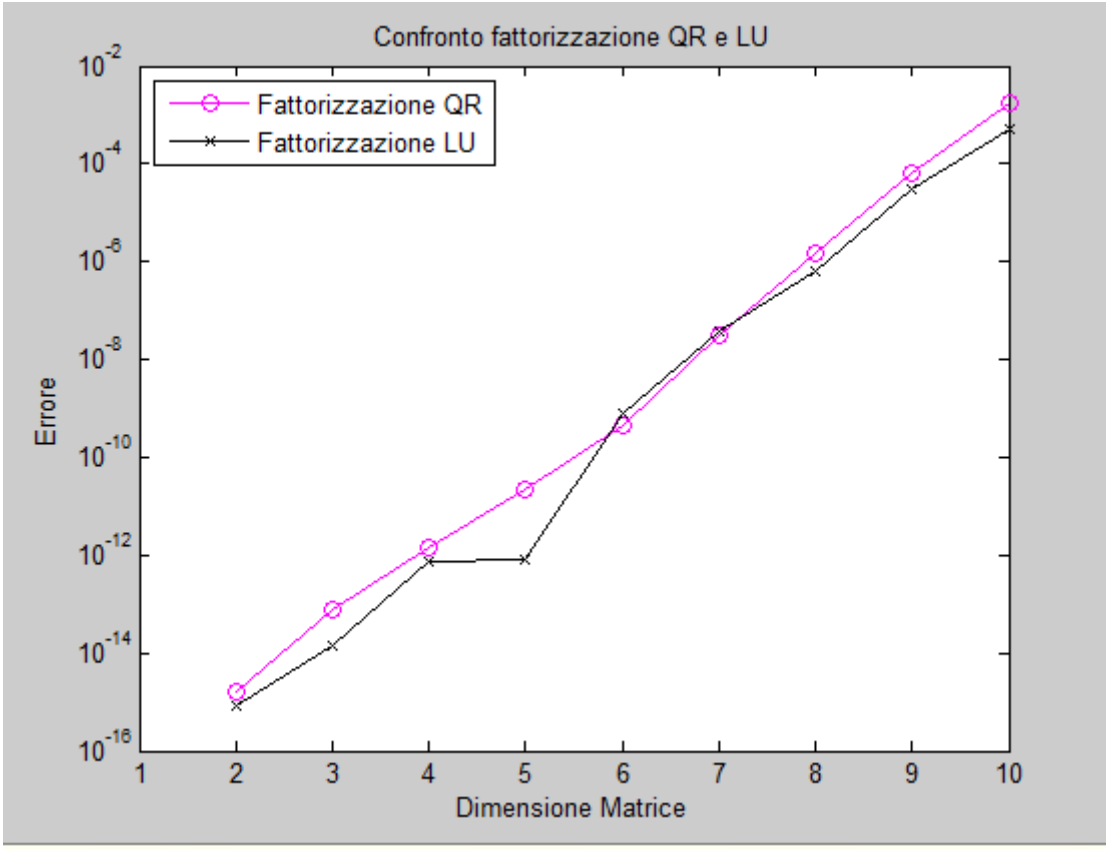


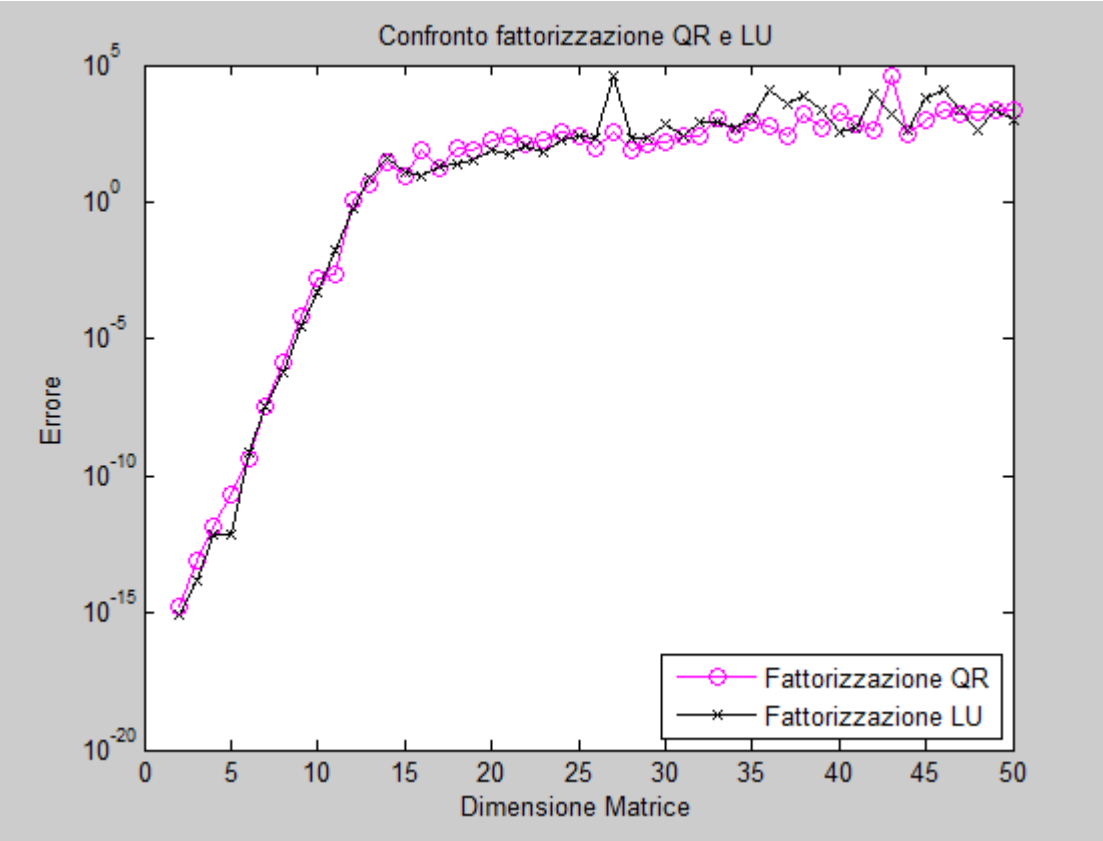
Quanto sopra scritto è reso ancora più notevole dall'ulteriore aumento di n.
Per n=100 infatti si veda come i tempi dell'algoritmo personalizzato siano molto più lenti.



1.3.2 Confronto metodo QR e LU per sistemi malcondizionati.

Quando si sceglie di effettuare una fattorizzazione, si può scegliere sia una fattorizzazione QR che una fattorizzazione LU. In realtà la scelta non è del tutto casuale, in quanto tra di essi cambia notevolmente il condizionamento.
Nelle fattorizzazioni LU il condizionamento del sistema finale potrebbe essere notevolmente superiore rispetto al sistema iniziale, mentre nelle fattorizzazioni QR, al contrario, il condizionamento non cambia tra sistema iniziale e sistema finale. Questo fatto rende preferibile la scelta di una fattorizzazione QR nei sistemi mal condizionati. Un caso in cui un sistema può essere mal condizionato è quando due equazioni sono quasi identiche. Ovviamente è ancora peggio quando sono completamente identiche. Vediamo di darne una rappresentazione.
Un sistema malcondizionato è rappresentato dalla matrice di Hilbert.
La matrice di Hilbert è una particolare matrice che ha sulle anti-diagonali i reciproci dei numeri interi. Questa matrice è fortemente mal condizionata, ossia amplifica molto gli errori al crescere della dimensione del sistema.





Come si evince dai grafici, la fattorizzazione QR è notevolmente più lineare, mentre la fattorizzazione LU è più irregolare. Questo fatto si nota maggiormente nelle matrici di piccole dimensioni, in cui i punti sono ben separati.
Tra i grafici non ho inserito quello della matrice di dimensione n=100, in quanto l'elaborazione ha dato dei problemi dovuti al fatto che matlab riconosce la matrice di Hilbert come una matrice malcondizionata, pertanto invia messaggi di warning che rallentano il processo e non lo fanno ultimare.

1.3.3 FATTORIZZAZIONE DI GIVENS

La fattorizzazione di Givens è incentrata sulla rotazione di Givens.

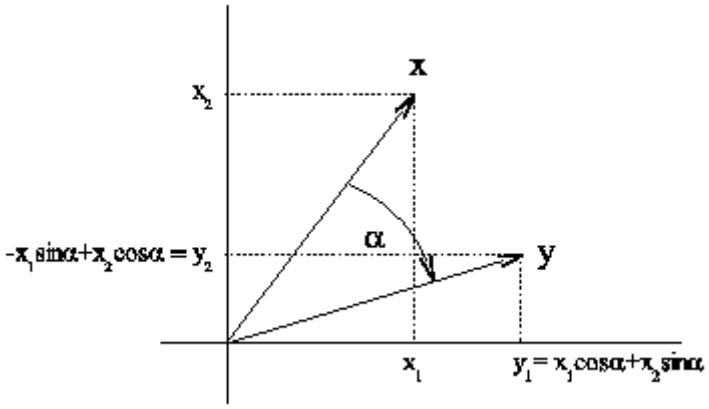
Una rotazione di Givens, o trasformazione di Givens è definita mediante una matrice ortonormale $G_{ij} \in \mathbb{R}^m$

inserisci matrice

dove c=cos θ e s=sin θ per θ appartenente ai reali e 1<=i<j<=n. La rotazione pertanto rappresenta la rotazione planare nello spazio effettuata dai vettori unitari e_i e_j rotati di un angolo θ.
Nelle due dimensioni la rotazione $G_{12}(\theta)$ data da

$$G_\theta = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

rappresenta una rotazione oraria pari all'angolo θ;



Ora, dato un vettore x appartenente a $\mathbb{R}^m$, se effettuiamo il prodotto della matrice con il vettore non è necessario effettuare il prodotto matriciale in modo esplicito, infatti risulta che

$$y_k = \begin{cases} c_{xi} + s_{xj}, & k = i, \\ -s_{xi} + c_{xj} & k = j, \\ x_k, & k \neq i, j \end{cases}$$

Si possono trovare i parametri c ed s in modo da da annullare la componente j-esima del vettore x. Si procede nel modo seguente: si pone

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

siccome vale che s2+c2=1, sostituendo l'equazione precedente otteniamo:

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

e da qui si ricavano

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

quindi otteniamo

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

L'algoritmo di fattorizzazione QR di Givens consiste nell'applicare ripetutamente matrici elementari di Givens alla matrice iniziale *A*, aventi i coefficienti *c* ed *s* calcolati in modo da azzerare sistematicamente gli elementi del triangolo inferiore di *A*. Quel che si ottiene da questa fattorizzazione è:

$$G_{n,m} \ldots G_{1,3} G_{1,2} A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

Nel caso in cui la matrice avesse dimensione mxn. Possiamo allora porre:

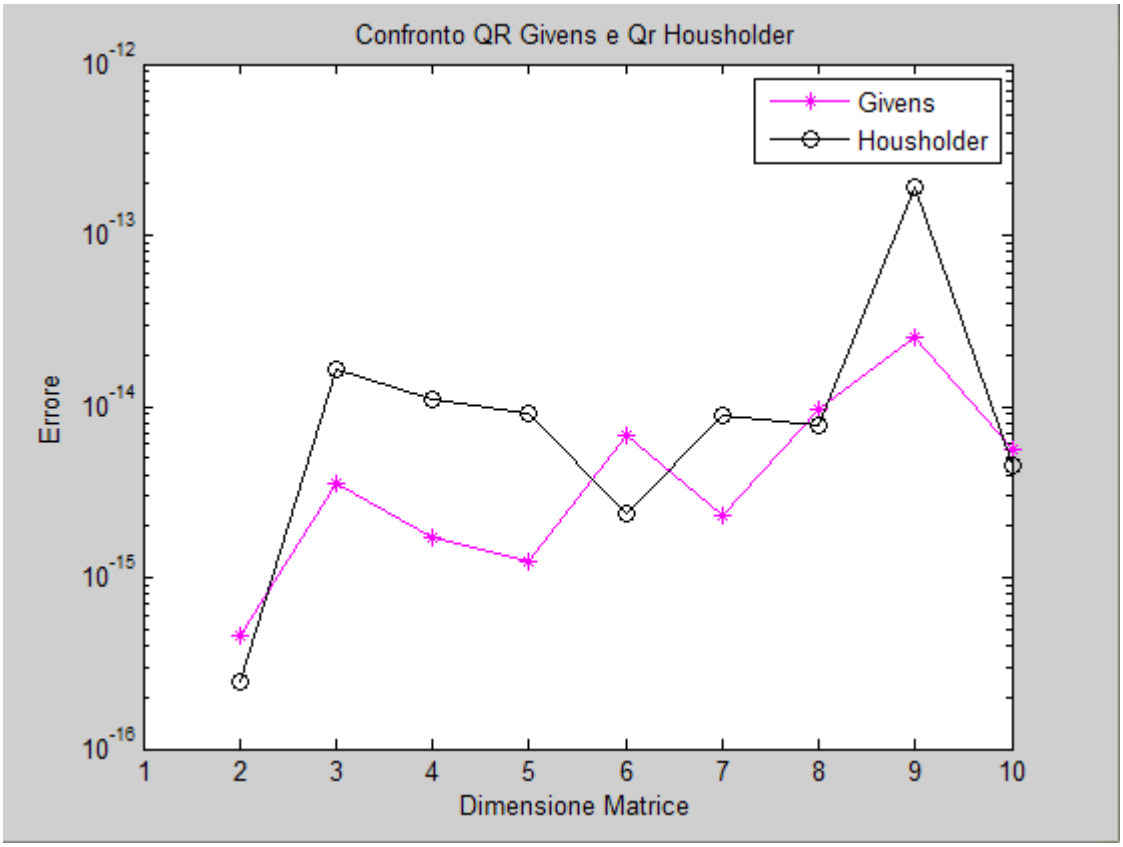
$$Q^T = G_{n,m} \ldots G_{1,3} G_{1,2}$$

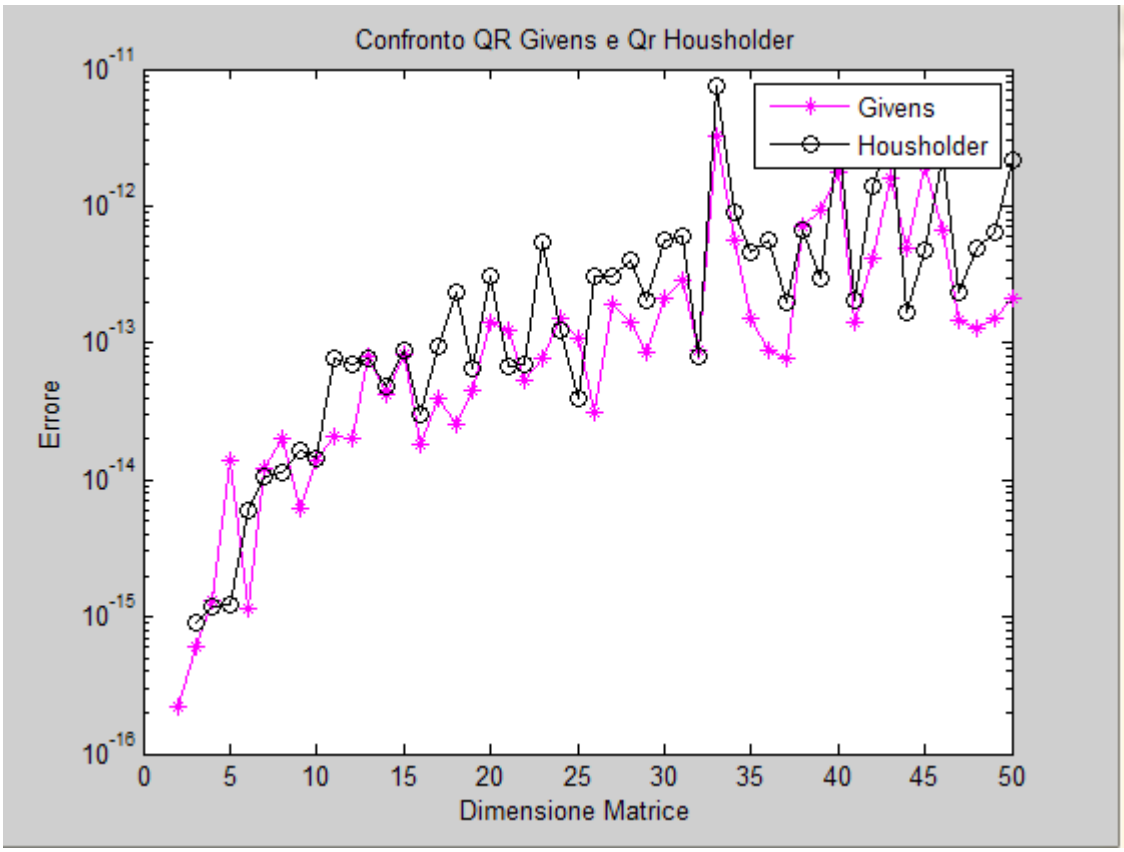
In cui Q è una matrice ortogonale in quanto prodotto di matrici ortogonali. Ciò implica del resto:

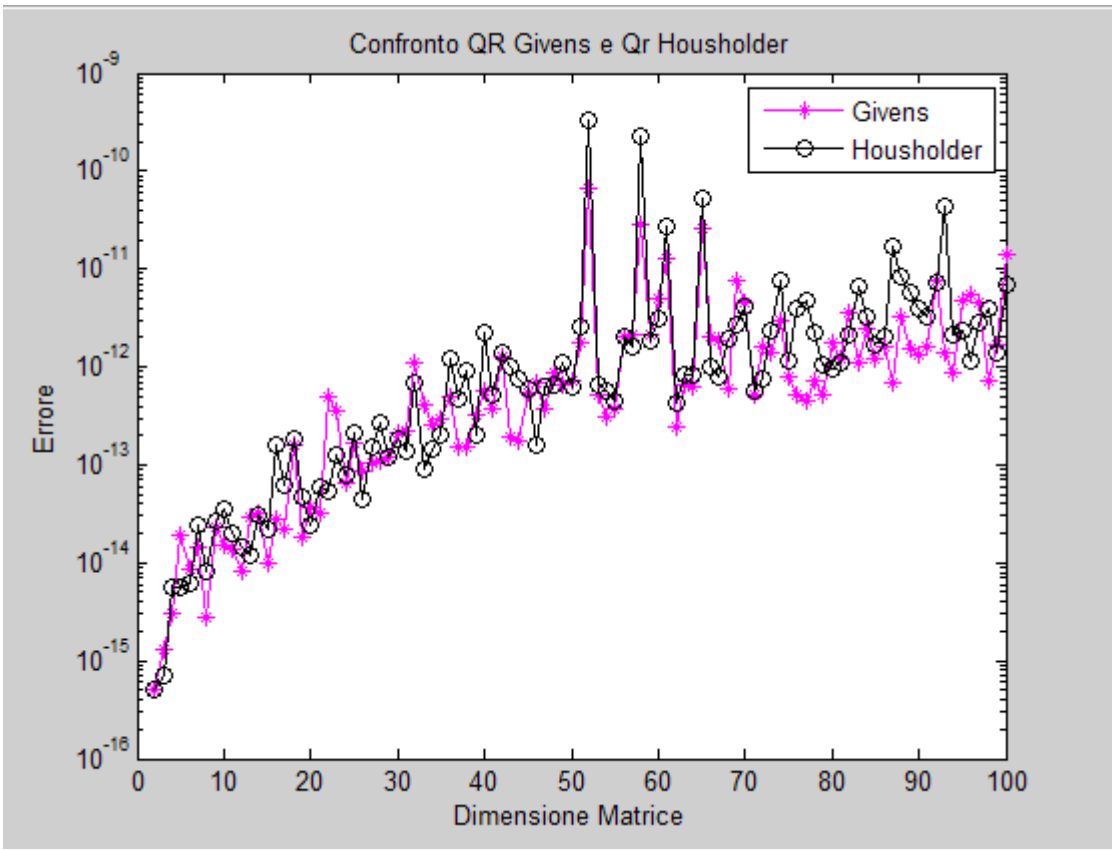
$$A = QR$$

che è ciò che volevamo ottenere.
Un metodo di questo tipo, rispetto alla fattorizzazione di Householder, richiede in generale il doppio delle operazioni in virgola mobile. Del resto però risulta molto più efficiente nel caso in cui si abbia a che fare con matrici sparse, poiché le matrici di Givens tali da annullare gli elementi già nulli non sono necessarie.

Vediamo graficamente la differenza tra i due metodi di fattorizzazione QR in funzione della dimensione della matrice.
Per matrici di piccole dimensioni, n=10, vediamo che l'errore relativo alla fattorizzazione di Householder è inferiore rispetto a quello di Givens. L'errore nel primo caso infatti è compreso tra $6*10^{-15}$ e circa $9*10^{-14}$, mentre nel secondo caso subisce delle variazioni che arrivano a $9*10^{-14}$.
Per matrici di più grandi dimensioni, i due metodi tendono a coincidere, sia per n=100 che per n=50. L'errore in norma due per n=50 è compreso tra 10^{-15} e 10^{-11}, mentre per matrici con n=100 il range dei valori dell'errore in norma due è compreso tra 10^{-15} e 10^{-8}.

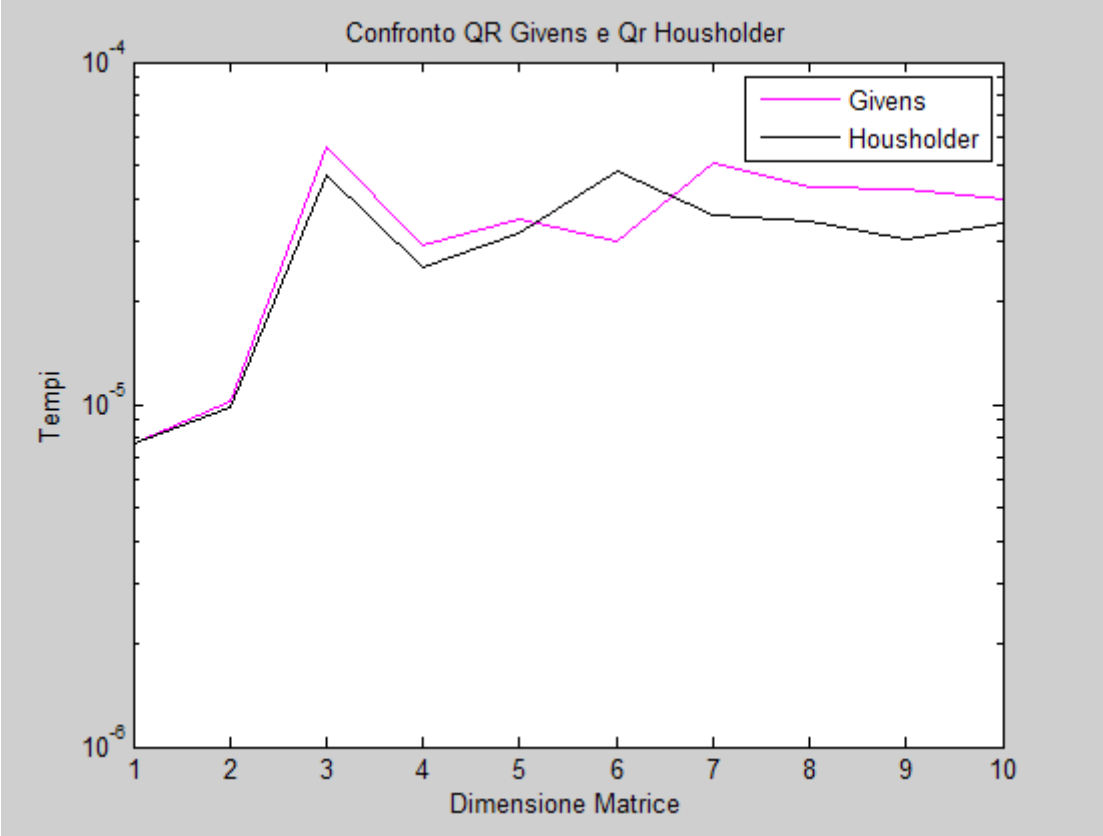



Confronto QR Givens e Qr Housholder
Givens
Housholder
Errore
Dimensione Matrice
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
0
5
10
15
20
25
30
35
40
45
50

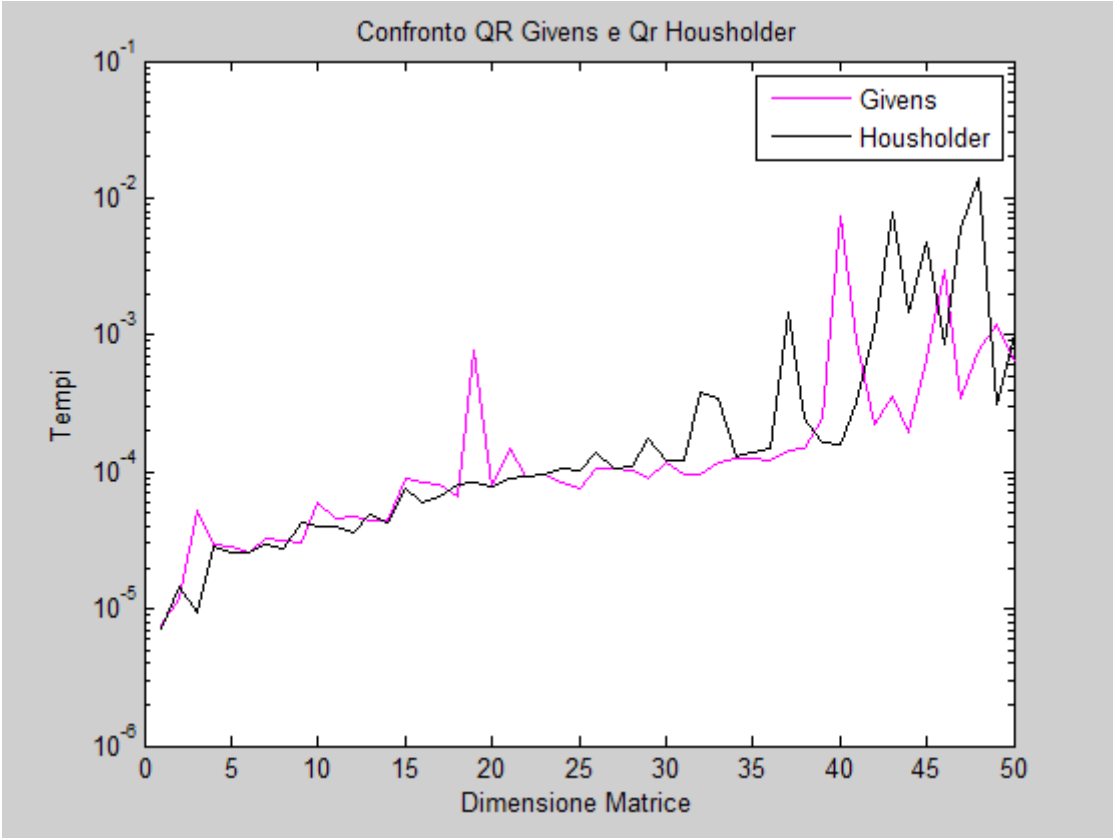

Confronto QR Givens e Qr Housholder
Givens
Housholder
Errore
Dimensione Matrice
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
0
10
20
30
40
50
60
70
80
90
100

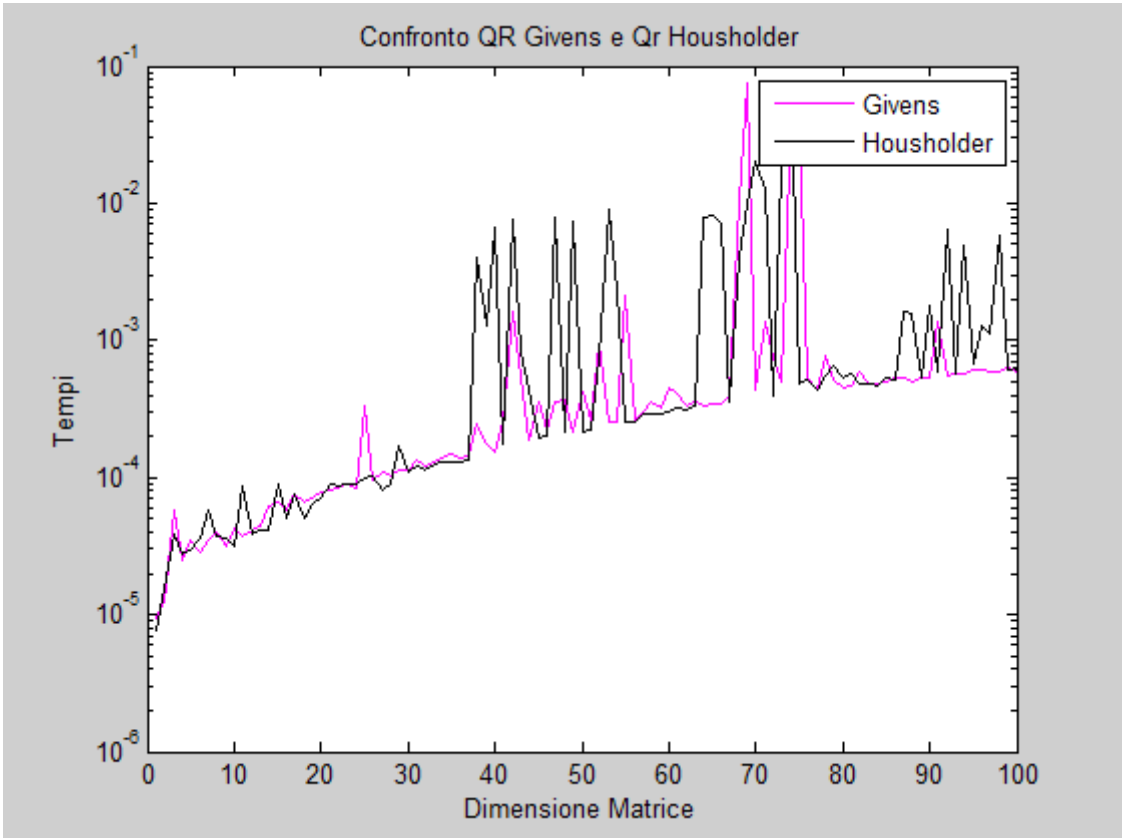
Il risultato che abbiamo ottenuto è effettivamente coerente con quanto scritto sopra. La complessità computazionale tra i due metodi, se applicati a matrici quadrate, è notevolmente differente. Il metodo di Householder richiede $O\left(\frac{2}{3}n^3\right)$ moltiplicazioni, contro le $O\left(\frac{4}{3}n^3\right)$ moltiplicazioni nel caso di Givens.

Questa differenza è visibile attraverso lo studio dei tempi di simulazione di entrambi i metodi:



Per matrici di piccole dimensioni i due metodi presentano pressoché gli stessi tempi. Al crescere delle dimensioni però accade che l'algoritmo di Householder più efficiente computazionalmente, questo perché richiede la metà delle moltiplicazioni rispetto all'algoritmo di Givens.


Confronto QR Givens e Qr Housholder
Givens
Housholder
Tempi
Dimensione Matrice
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
0
5
10
15
20
25
30
35
40
45
50


Confronto QR Givens e Qr Housholder
Givens
Housholder
Tempi
Dimensione Matrice
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
0
10
20
30
40
50
60
70
80
90
100

1.4 L' ALGORITMO QR

Per calcolare tutti gli autovalori di una matrice A si cerca di trasformarla, per similitudine, in una di forma particolarmente semplice. La forma più desiderabile sarebbe quella diagonale (gli autovalori sono gli elementi sulla diagonale) ma non è possibile ridurre per similitudine ogni matrice in forma diagonale in un numero finito di passi. Data la matrice quadrata A, si definisce

$$A_0 = A;$$

Se $A_k = Q_k R_k$, con R_k triangolare superiore e Q_k ortogonale, allora si costruisce la successione:

$$A_{k+1} = R_k Q_k$$

con k= 0, 1, 2, ….

La successione $\{A_0, A_1, A_2, ...\}$ è essenzialmente unica (questo fatto discende dall'unicità della decomposizione QR.

Tutte le matrici A_k, sono simili fra loro, dato che

$$A_{k+1} = R_k Q_k = (Q^T_k Q_k) R_k Q_k = Q^T_k (Q_k R_k) Q_k = Q^T_k (A_k) Q_k$$

e $Q^T_k = Q^{-1}$.

Se la matrice A, ha autovalori distinti, la successione Ak definita attraverso l'algoritmo QR converge ad una matrice triangolare superiore sulla cui diagonale si trovano gli autovalori di A.

$$T = \begin{bmatrix} \lambda_1 & \bullet & \cdots & \bullet \\ & \lambda_2 & \ddots & \vdots \\ & & \ddots & \bullet \\ & & & \lambda_n \end{bmatrix}$$

Negli altri casi tende alla forma seguente:

$$\begin{bmatrix} \bullet & \bullet & \bullet & \cdots & \bullet \\ \bullet & \bullet & \bullet & \ddots & \bullet \\ & & \lambda_3 & \ddots & \vdots \\ & & & \ddots & \bullet \\ & & & & \lambda_n \end{bmatrix}$$

in cui la parte sottostante una diagonale a blocchi, i cui blocchi sono di dimensione 1 e di dimensione 2, è composta da zeri . Gli autovalori dei blocchi diagonali superstiti convergono agli autovalori di A. Il limite non può essere sempre una matrice triangolare superiore perché, nel caso di matrici reali, le trasformazioni di Householder e quindi le fattorizzazioni QR e i prodotti RQ sono ancora reali ad ogni passo della successione. Non si potrebbero quindi trovare gli eventuali autovalori complessi di A. In presenza di coppie di autovalori complessi coniugati non e infatti verificata l'ipotesi. In tal caso compaiono dei blocchi diagonali a coefficienti reali di dimensione 2, i cui autovalori complessi coniugati approssimano quelli di A.

Il costo computazionale del metodo QR e notevolmente alto: $\frac{2}{3}n^3$ per ogni fattorizzazione QR e $\frac{n^3}{2}$ per ogni prodotto RQ, complessivamente $\frac{7}{6}n^3$ per ogni iterazione.

Mostro ora un esempio del modo in cui funziona l'algoritmo QR, attraverso l'uso dei comandi di matlab.

Supponiamo di avere una generica matrice A 5x5:

A =

2.2362	1.7520	1.4943	1.0274	1.0467
1.7520	1.6695	1.1969	0.6517	0.7681
1.4943	1.1969	1.7012	0.5591	1.2695
1.0274	0.6517	0.5591	0.7690	0.5843
1.0467	0.7681	1.2695	0.5843	1.1218

calcoliamo gli autovalori con il comando eig di matlab:

```
>> eig(A)

ans =

    0.0019
    0.1105
    0.4992
    0.9039
    5.9822
```

Applico ora l'algoritmo QR, attraverso la funzione “algqr”, che prende in ingresso la matrice A e un generico numero di iterazioni che stabilisco a piacere, e calcola in uscita la matrice T, nella cui diagonale compaiono gli autovalori di A, e le matrici Q ed R in cui la matrice A viene fattorizzata.

[T,Q,R]=algqr(A,niter)

T =

5.9822	-0.0000	-0.0000	-0.0000	-0.0000
-0.0000	0.9039	-0.0000	-0.0000	-0.0000
0.0000	-0.0000	0.4992	0.0000	-0.0000
0.0000	-0.0000	-0.0000	0.1105	0.0000
0.0000	-0.0000	-0.0000	0.0000	0.0019

Q =

1.0000	0.0000	-0.0000	-0.0000	0.0000
-0.0000	1.0000	0.0000	0.0000	-0.0000
0.0000	-0.0000	1.0000	0.0000	-0.0000
0.0000	-0.0000	-0.0000	1.0000	-0.0000
0.0000	-0.0000	-0.0000	0.0000	1.0000

R =

5.9822	-0.0000	-0.0000	-0.0000	-0.0000
-0.0000	0.9039	-0.0000	-0.0000	-0.0000
-0.0000	0	0.4992	0.0000	-0.0000
0.0000	0	0	0.1105	0.0000
-0.0000	0	0	0.0000	0.0019

Come si può vedere nella diagonale principale T compaiono gli autovalori della matrice A, mentre Q ed R sono il risultato della fattorizzazione QR di A, come volevamo dimostrare.

Supponiamo ora di avere una matrice A i cui autovalori non sono tutti reali, ma alcuni sono complessi coniugati. Vediamo come si comporta l'algoritmo QR:

A =

0.8244	0.9814	0.8266	0.8509	0.9649
0.9915	0.2546	0.8551	0.2461	0.9910
0.9843	0.3328	0.9510	0.2355	0.7458
0.5584	0.0367	0.3432	0.9527	0.1809
0.2175	0.8213	0.8360	0.8074	0.3675

Calcolo gli autovalori:

```
>> eig(A)

ans =

   3.2619
  -0.6887
   0.7525
   0.0123 + 0.2547i
   0.0123 – 0.2547i
```

Stabilisco un numero di iterazioni, alto a piacere:

```
 niter=150
```

e richiamo la mia funzione:

```
>> [T,Q,R]=algqr(A,niter)
```

T =

3.2619	0.4170	0.2336	0.0550	-0.5882
-0.0000	0.7525	-0.4422	0.0310	-0.1126
-0.0000	-0.0000	-0.6887	-0.4476	-0.3323
-0.0000	-0.0000	-0.0000	-0.0341	0.1632
0.0000	-0.0000	-0.0000	-0.4107	0.0587

Q =

1.0000	0.0000	-0.0000	0.0000	-0.0000
-0.0000	1.0000	-0.0000	-0.0000	-0.0000
0.0000	0.0000	1.0000	0.0000	0.0000
0.0000	0.0000	0.0000	-0.1414	-0.9900
-0.0000	0.0000	0.0000	0.9900	-0.1414

R =

3.2619	0.4170	0.2336	0.5745	0.1376
0	0.7525	-0.4422	0.1071	0.0466
0	-0.0000	-0.6887	0.3922	-0.3961
0	0.0000	-0.0000	-0.1567	-0.0568
0	0.0000	0.0000	0	-0.4149

Dopo 150 iterazioni vediamo che la matrice T non è perfettamente diagonale, tuttavia gli autovalori reali si trovano nella sua diagonale, mentre gli autovalori complessi coniugati si trovano considerando il blocco diagonale di dimensione 2x2 della matrice T. Infatti considerando tale blocco come una matrice otteniamo:

C =

-0.0341	0.1632
-0.4107	0.0587

Calcolando glia autovalori della matrice C otteniamo :

```
>> eig(C)

ans =

   0.0123 + 0.2547i
   0.0123 – 0.2547i
```

che sono gli autovalori complessi della matrice A di partenza.

1.5 FORMA DI HESSENBERG

Il metodo di fattorizzazione di Givens è più efficiente rispetto a quello di Householder quando abbiamo a che fare con matrici sparse oppure con matrici in forma di Hessenberg.
Una matrice A può essere trasformata mediante trasformazione di similitudine in una matrice strutturata, della forma:

$$\tilde{A} = \begin{bmatrix} \bullet & \cdots & \cdots & \bullet \\ \bullet & \ddots & & \vdots \\ & \ddots & \ddots & \vdots \\ & & \bullet & \bullet \end{bmatrix}$$

che prende il nome di struttura di Hessenberg, dove $\tilde{a}_{ij}$ =0 per i<j+1.
Le matrici di questa forma sono importanti perché sono invarianti rispetto a trasformazioni QR, ovvero se viene applicata la fattorizzazione QR a una matrice di Hessenberg, le matrici della successione che verranno generate saranno anch'esse di Hessenberg. Questo fatto accelera la convergenza del metodo. L'algoritmo richiede complessivamente n-2 passi e la trasformazione per similitudine Q può essere

calcolata come prodotto di matrici di Householder $P_{(1)}...P_{(n-2)}$.
In ogni passo k-esimo si trasforma A per similitudine, attraverso la matrice di Householder P_k, alla quale fa in modo di annullare gli elementi in posizione k+2,...n della colonna k-esima della matrice per k=1,...,(n-2).
Il passaggio in forma di Hessenberg richiede un costo computazionale dell'ordine di n^3 flops.
La procedura può essere così schematizzata:

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \rightarrow P_{(1)} \rightarrow \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \end{bmatrix} \rightarrow P_{(2)} \rightarrow \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & 0 & \bullet & \bullet \end{bmatrix}$$

dove gli elementi indicati con il simbolo • sono gli elementi non nulli della matrice.
Partendo dalla condizione A=A(0) si genera la successione:

$$P_{(k)}^{T} A^{(k-1)} P_{(k)} = \left(P_{(1)} ... P_{(k)}\right)^{T} A \left(P_{(1)} ... P_{(k)}\right) = Q_{(k)}^{T} A Q_{(k)} \qquad k \geqslant 1.$$

L'operazione $P_{(k)}^{T} A^{(k-1)}$ lascia invariate le prime k righe di $A^{(k-1)}$, mentre l'operazione $P_{(k)}^{T} A^{(k-1)} P_{(k)}$ lascia invariate le prime k colonne. Dopo n-2 iterazioni si arriva alla matrice $\tilde{A}$ in forma di Hessenberg superiore.

1.6 FATTORIZZAZIONE QR DI UNA MATRICE IN FORMA DI HESSENBERG

Il vantaggio di utilizzare la matrice di Hessenberg, oltre a quello già citato dell'accelerazione della convergenza è legato al metodo di Givens. Infatti per

calcolare la fattorizzazione QR di una matrice in forma di Hessenberg è possibile applicare (n-1) trasformazioni di Givens che annullano gli elementi sottodiagonali.
Partiamo da una matrice in forma di Hessenberg T(0)=H(0).
Per ogni k>=1 si calcola la fattorizzazione QR di H(k-1) con n-1 rotazioni di Givens

$$\left(Q^{(k)}\right)^T H^{(k-1)} = \left(G_{n-1}^{(k)}\right)^T \ldots \left(G_1^{(k)}\right)^T H^{(k-1)} = R^{(k)},$$

che richiede $3n^2$ flops.

Il passo successivo consiste nel completare la trasformazione per similitudine ortogonale

$$H^{(k)} = R^{(k)} Q^{(k)} = R^{(k)} \left(G_1^{(k)} \ldots G_{n-1}^{(k)}\right),$$

che richiede $3n^2$ operazioni.

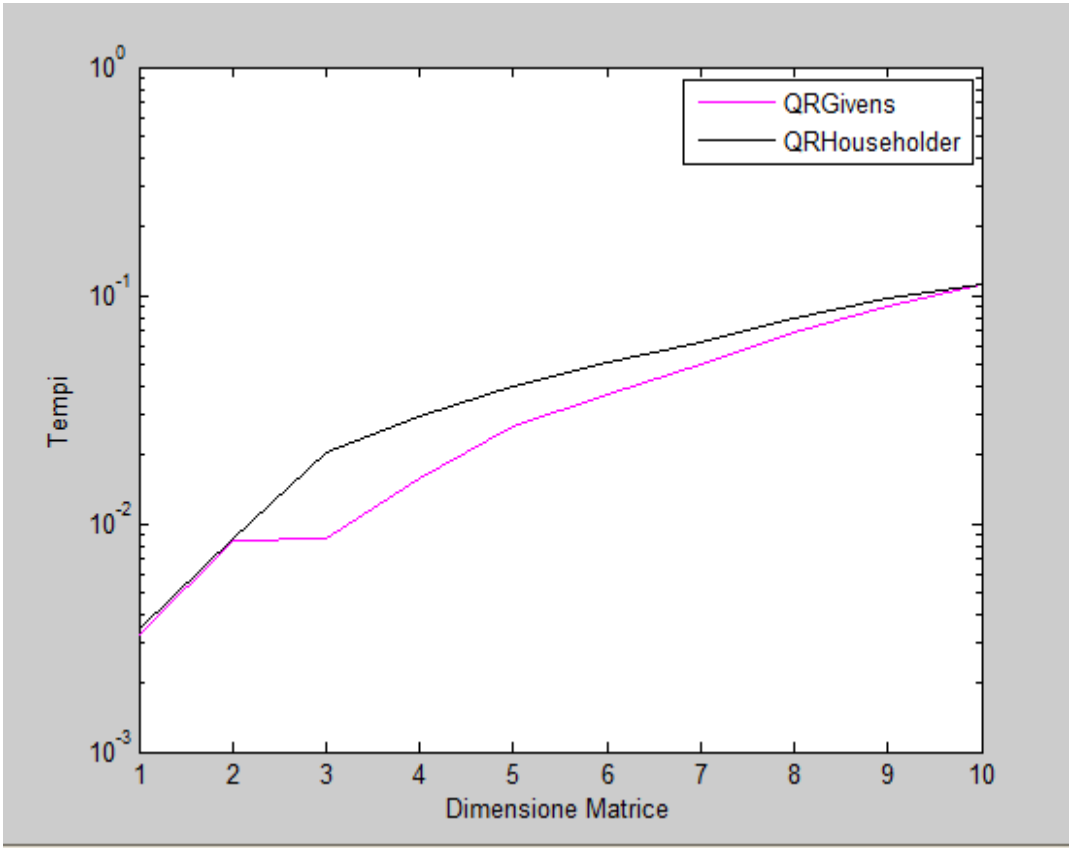
La matrice $Q^{(k)} = \left(G_1^{(k)} \ldots G_{n-1}^{(k)}\right)$ è ortogonale ed è in forma di Hessenberg superiore. Si ha infatti che

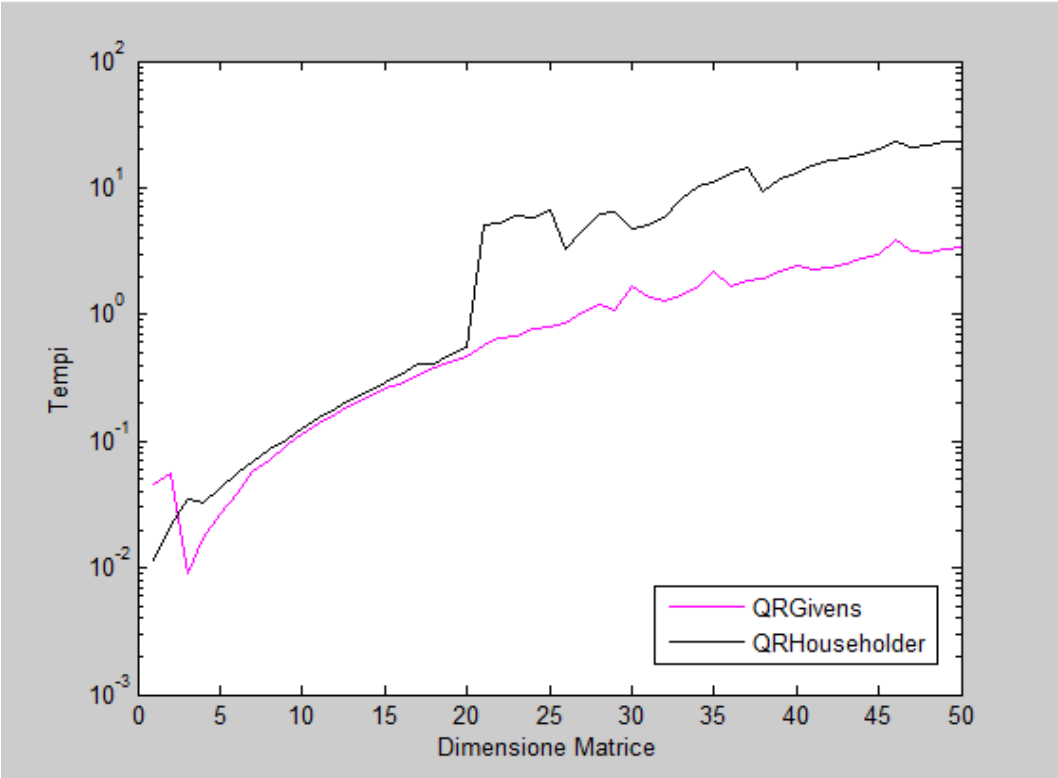
$$Q^{(k)} = G_1(k) G_2(k) = \begin{bmatrix} \bullet & \bullet & 0 \\ \bullet & \bullet & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \bullet & \bullet \\ 0 & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet \end{bmatrix}$$

In tutto dunque si ha un costo pari a $6n^2$ flops.
A questo punto, se proviamo ad applicare le fattorizzazioni QR di Givens e QR di Householder questa volta a una matrice in forma di Hessenberg, abbiamo che la fattorizzazione di Givens è più efficiente, anche al crescere delle dimensioni della matrice.
In matlab la matrice di Hessenberg si ottiene mediante il comando

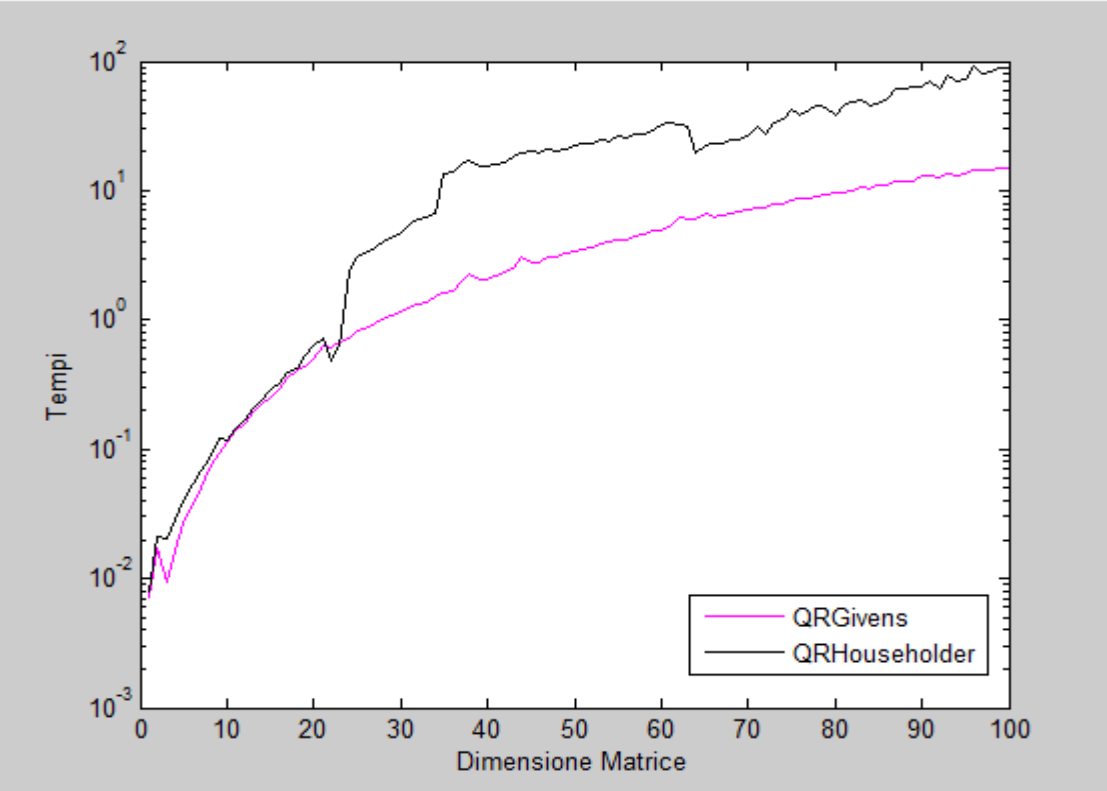
```
hess(A);
```

Facendo una valutazione sui tempi di elaborazione, che mettono in risalto le differenze computazionali tra gli algoritmi, otteniamo i seguenti risultati al variare di n:





Come si nota, l'algoritmo di Givens è eccezionalmente più stabile, i tempi sono nettamente inferiori rispetto all'algoritmo di Householder al crescere della dimensione n della matrice in forma di Hessenberg.



1.7 ALGORTIMO QR CON SHIFT

Se gli autovalori della matrice A anche se sono tutti distinti tra loro non sono ben separati, è possibile che la convergenza della matrice T potrebbe risultare molto lenta. Per questo motivo è utile implementare la tecnica dello shift.
Esistono diverse tecniche di shift, tuttavia in questa tesina verrà affrontata la tecnica dello shift singolo, che consente dia accelerare la convergenza rispetto al metodo di base quando A presenta autovalori vicini in modulo.
Il metodo QR con shift può essere schematizzato nel modo seguente:

1. Si determinano $Q^{(k)}$, $R^{(k)}$ tali che
   $Q^{(k)}R^{(k)}= T^{(k-1)}- \mu I$ (fattorizzazione QR)
2. Si pone $T^{(k)} = Q^{(k)}R^{(k)}+ \mu I$

dove $\mu \in \mathbb{R}$ viene chiamato shift, k=1,2,...fino alla convergenza e $T^{(0)}=\left(Q^{(0)}\right)^T AQ^{(0)}$ è una matrice in forma di Hessenberg superiore.

La matrice T(k) generata è simile alla matrice A di partenza per ogni $k \geqslant 1$.

Supponiamo che μ abbia un valore fissato. Ordiniamo gli autovalori di A nel modo seguente:

$$|\lambda_1 - \mu| \geqslant |\lambda_2 - \mu| \geqslant \ldots \geqslant |\lambda_n - \mu|.$$

Si può dimostrare che per $1 < j \leqslant n$, l'elemento sottodiagonale $t^{(k)}_{j,j-1}$ va a zero proporzionalmente al rapporto

$$|(\lambda_j - \mu)/(\lambda_{j-1} - \mu)|^k.$$

Inoltre se si sceglie μ in modo tale

$$|\lambda_n - \mu| \leqslant |\lambda_i - \mu|,$$

con i = 1,...n-1, allora l'elemento t(k)n,n-1 generato dallo schema sopra riportato tende rapidamente a zero al crescere di k.
Il metodo QR con shift si ottiene scegliendo

$$\mu = t^{(k)}_{n,n}$$

Se la matrice A è in forma di Hessenberg si attua una strategia di deflazione, ovvero quando si azzerano gli elementi della matrice ciò che otterremo è che gli elementi $t^{(k)}_{n,n}$ sono un'approssimazione degli autovalori $\lambda_{n.}$. Il metodo QR quindi continuerà sulla matrice $T^{(k)}$ di dimensioni n-1. Tale metodo verrà reiterato finché non si

otterranno tutti gli autovalori di A.

Un metodo per la costruzione dello shift con matlab è il seguente:

```
%QR con singolo shift

function[T,iter]=qrshift(A,toll,itmax)
n=max(size(A));
iter=0
[T,Q]=houshess(A);
for k=n:1:2
  I=eye(k);
while abs(T(k,k-1))>toll*(abs(T(k,k))+abs(T(k-1,k-1)))
iter=iter+1;
if(iter>itmax),
return
end
mu=T(k,k);
[Q,R,c,s]=qrgivens(T(1:k,1:k)-mu*I);
T(1:k,1:k)=R*Q+mu*I;
end
T(k,k-1)=0;
end
```