Alessandro Fanti, Alessandro Murgia dottorandi XXIII ciclo

Introduzione

Questa tesina illustra la fattorizzazione QR con le matrici di Householder e l'impiego di tale fattorizzazione nella risoluzione di problemi ai minimi quadrati. Nella tesina verrano presentati i diversi algoritmi implementati prestando particolare attenzione alla visualizzazione degli errori commessi da ciascuno. Nella parte finale della tesina illustriamo il metodo della TSVD.

Teoria

Dato un sistema lineare A**x** = **b** esistono diversi algoritmi con il quali determinare il vettore x (soluzione del sistema). Nel caso in cui A sia una matrice quadrata non singolare, si può ricorrere alla fattorizzazione QR per la risoluzione del sistema.
Data la matrice *A m x n* si può riscrivere questa come:
$A=QR$

con Q matrice m × m ortogonale per la quale vale che $Q^{\mathrm{T}} Q = QQ^{\mathrm{T}} = I$
e R matrice m x n triangolare superiore.
La fattorizzazione QR ha il vantaggio di consentire la trasformazione di un sistema lineare in un sistema triangolare superiore ad esso equivalente e dotato dello stesso numero di condizionamento. Con una fattorizzazione LU potremo avere invece un condizionamento del sistema finale che può aumentare in maniera significativa. Per questa proprietà la fattorizzazione QR è preferibile nella risoluzione di sistemi lineari malcondizionati.
Il sistema Ax=b può quindi essere riscritto tramite fattorizzazione QR come:

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

dalla prima equazione abbiamo **c**= $Q^{\mathrm{T}}\boldsymbol{b}$ mentre il secondo sistema può essere risolto per sostituzione all'indietro.
La fattorizzazione QR viene comunemente adoperata anche quando si devono risolvere problemi ai minimi quadrati:

$$\min_{x\in\Re}\|Ax-b\|_2 \tag{1}$$

Sostituendo ad A la sua fattorizzazione QR il problema (1) si trasforma come segue:

$$||A\mathbf{x}\text{-}\mathbf{b}||^2 = ||QR\mathbf{x}\text{-}\mathbf{b}||^2 = ||Q(R\mathbf{x}\text{-}Q^T\,\mathbf{b})||^2 = ||R\mathbf{x}\text{-}\mathbf{c}||^2 \tag{2}$$

Dove **c**= Q^{T}**b**. La matrice R può essere riscritta come $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ con R_1 matrice triangolare superiore quadrata. Suddividendo il vettore **c** come:

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \qquad C_1 \in \Re^n,\ C_2 \in \Re^{m-n}$$

possiamo riscrivere la (2) come:

$$||A\mathbf{x}\text{-}\mathbf{b}||^2 = \ ||R\mathbf{x}\text{-}\mathbf{c}||^2 = \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right\|^2 = ||R_1\mathbf{x}\text{-}\mathbf{c}_1||^2 + ||\mathbf{c}_2||^2 \tag{3}$$

e nel caso in cui $\det(R_1)\neq 0$ abbiamo che il problema (1) può essere riscritto come:

$$\min_{x\in\Re}\|Ax-b\|_2 = \|c_2\| \tag{3}$$

Un altro contesto nel quale sono utili le fattorizzazioni QR sono le decomposizioni matriciali SVD.
La SVD (*Singular Value Decomposition*) consente di esprimere una matrice $A \in \Re^{mxn}$ con $m \geq n$ in una forma fattorizzata del tipo:

$$A = U\Sigma V^* = \sum_{i=1}^{n} u_i \sigma_i \upsilon_i^* \tag{4}$$

Dove U è una matrice unitaria m x n, Σ è una matrice diagonale n x n con elementi non negativi, V^*è l'aggiunta di una matrice V unitaria.
L'algoritmo prevede la decomposizione della matrice A tramite un metodo iterativo. La decomposizione QR viene richiamata più volte durante i diversi passi d'iterazione.

Talvolta il malcondizionamento della matrice A non consente l'utilizzo dell'SVD. In questi casi, a patto che la matrice A non sia troppo "grande[1]" si può ricorrere alla TSVD (Truncated SVD).
L'applicazione della TSVD richiede anticipatamente l'utilizzo della SVD, ed è proprio per questo motivo che scaturiscono alcune limitazioni nelle dimensioni delle matrici su cui è possibile applicare l'algoritmo.
La TSVD non considera tutti i valori singolari scaturiti dalla SVD, ma solo i primi k (ossia i valori più grandi). I restanti valori singolari vengono invece posti a zero, in questa maniera si cerca di ridurre il problema della crescita del numero di condizionamento della matrice A scomposta.
Per stimare il valore ottimale di k (parametro di troncamento) è possibile calcolare tutte le TSVD (con k variabile), poi procedere con il calcolo dell'errore (in norma 2) della soluzione rispetto alla soluzione vera del problema. Il k migliore sarà quello nel quale otteniamo l'errore minimo. Questo modo di procedere ovviamente non può essere adoperato nei casi reali, dato che non conosciamo le soluzioni del sistema.

La fattorizzazione QR può essere implementata in diverse maniere, come ad esempio attraverso l'utilizzo delle matrici di Householder[2].
Nella sezione successiva descriveremo l'algoritmo per la fattorizzazione QR di Householder.

Algoritmi

In questa sezione sono illustrati gli algoritmi adoperati per la fattorizzazione QR.

Funzione qr_householder

```
function [Q,R]=qr_householder(A)
[n,m]=size(A);
Anew=A;
Q=eye(n);
for j=1:m
    A_orlata=Anew(j:n,j:m);
    H=h_householder(A_orlata(:,1));
    H=de_orla(H,n,m);
    Q=Q*H;
    Anew=H*Anew;
end
```

[1] In genere la matrice non supera la dimensione di 1000 x 1000. Matrici più grandi risultano richiedere un onere computazionale eccessivo.

2 Le matrici di Householder sono simmetriche e ortogonali è possono essere riscritte come:

$$H = I - 2ww^T, \quad w \in \Re^n, \quad \|w\| = 1$$

```
R=Anew;
```

La funzione prende in ingresso la matrice A restituendone la sua fattorizzazione QR. Nel ciclo la matrice A viene orlata, da quest'ultima si estrae il primo vettore colonna, e dal vettore colonna si calcola la matrice H di Householeder. Alla matrice H vengono poi aggiunte una serie di righe e colonne[3] che generano una nuova matrice di dimensione m x n.
Le matrici Q ed R vengono costruite iterativamente all'interno del ciclo.

Funzione h_ householder

```
function  H=h_householder(x)
[n,m]=size(x);
e1=eye(n,1);   % versore colonna di lunghezza n
I=eye(n);

sigma=norm(x);
x11=x(1,1);
k=-sign(x11)*sigma;    % k>0
lambda=sqrt(2*sigma*(sigma+abs(x11)));
w=(x-k*e1)/lambda;
H=I-2*w*(w');
```

Questa funziona restituisce la matrice H di Householder

Funzione de_orla

```
function A=de_orla(B,n,m)
[r,c]=size(B);
A=B;
deltaRiga=n-r;
for j=1:deltaRiga
    [r,c]=size(A);
    e1=eye((r+1),1);
    row0=zeros(c,1)'; % vettore riga nullo
    A=[row0;A];
    A=[e1,A];
end
```

La funzione "de-orla" prende la matrice in ingresso B, di dimensioni r x c, e vi aggiunge k righe e k colonne, dove k è la differenza tra il numero di righe della matrice A (di cui si sta eseguendo la fattorizzazione) e il numero di righe della matrice H ottenuta dal metodo di h_householder. Le k colonne aggiunte hanno come caratteristica quello di essere dei versori (indipendenti), i cui elementi non nulli sono in posizione (1,1), (2,2), ..., (k,k).

Test algoritmi

Per testare l'algoritmo implementato e verificare il ruolo svolto dagli errori approssimazione, abbiamo risolto diversi problemi ai minimi quadrati ricorrendo alla fattorizzazione QR. Le matrici A sulle quali si è eseguita la fattorizzazione sono state:

- matrici generate in maniera random,
- matrici di Pascal
- matrici di Hilbert.

Per la risoluzione del problema ai minimi quadrati applichiamo la funzione "minq", che prende in ingresso la matrice A e il vettore b della (1) e restituisce il vettore soluzione x e la misura del

[3] Il tipo di righe e colonne inserite sono descritte successivamente nella funzione de_orla

residuo. L'algoritmo, tramite la fattorizzazione QR dai noi implementata, esegue quanto esposto nei passaggi descritti nella (2)

```
function [x,residuo]=minq(A,b)
[n,m]=size(A);
[q1,q2]=size(b);
[Q,R]= qr_householder(A);

c=Q'*b;

if n>m
    R1=R(1:m,1:m);
    c1=c(1:m);
    c2=c((m+1):n);
    d=det(R1);
    if d~=0
        x=R1\c1;
        residuo=norm(c2);
    end
end

if n==m
   x=R\c;
   residuo=0;
end
```

Seguono ora una serie di test nel quale risolviamo problemi di tipo (1) impiegando la funzione minq. In ogni test adoperiamo una matrice A differente e verifichiamo cosa questo comporti negli errori generati.

Caso 1:
matrice random A di dimensione 2m x m. L'errore commesso dall'algoritmo viene calcolato (e plottato) al crescere delle dimensioni di A.

```
i=0;
for m=6:100
    i = i +1;
    A=rand(2*m,m);
    b=A*ones(m,1);
    x=A\b;
    Xr{ i }=x;                  %Xr è l'array contenente i vettori x soluzione Matlab
    [x1,residuo]=minq(A,b);
    Xf{ i }=x1;                 %Xf è l'array contenente i vettori x1 soluzione del nostro algoritmo
    nr(i,1)=norm(x1-ones(m,1));     %  nr è un vettore contenente la norma gli errori del nostro
                                    % algoritmo[4]
    nm(i,1)=norm(x-ones(m,1));      %  nm è un vettore contenente la norma degli errori commessi
                                    % dall'algoritmo di Matlab
end
legend('residui');
semilogy([nr,nm])
legend('nr','nm');
```

[4] L'errore che commette il nostro algoritmo è la differenza fra la soluzione nostra e la soluzione reale

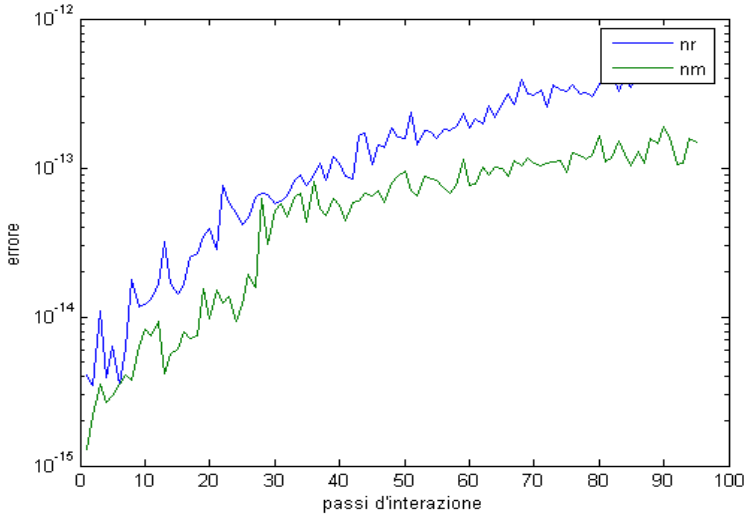

Figura 1 Andamento dell'errore al crescere della dimensione della matrice random A (caso 1)

Come possiamo vedere al crescere della dimensione della matrice A, aumenta l'errore dall'algoritmo minq e di quello di Matlab (anche se l'errore dell'algoritmo di Matlab risulta essere meglio).

Caso 2:

A è una matrice di Hilbert di dimensione m x m/2. L'errore commesso dall'algoritmo viene calcolato (e plottato) al crescere delle dimensioni di A.

```
i=0;
for m=6:25
    i=i+1;
    H=hilb(m);
    n=floor(m/2);
    A=H(1:m,1:n);
    b=A*ones(n,1);
    x=A\b;
    Xr{i}=x;
    [x1,residuo]=minq(A,b);
    Xf{i}=x1;
    res(i,1)=residuo;
    nrh(i,1)=norm(x1-ones(n,1));
    nmh(i,1)=norm(x-ones(n,1));
end
semilogy(res)
semilogy([nrh,nmh])
legend('nrh','nmh');
```

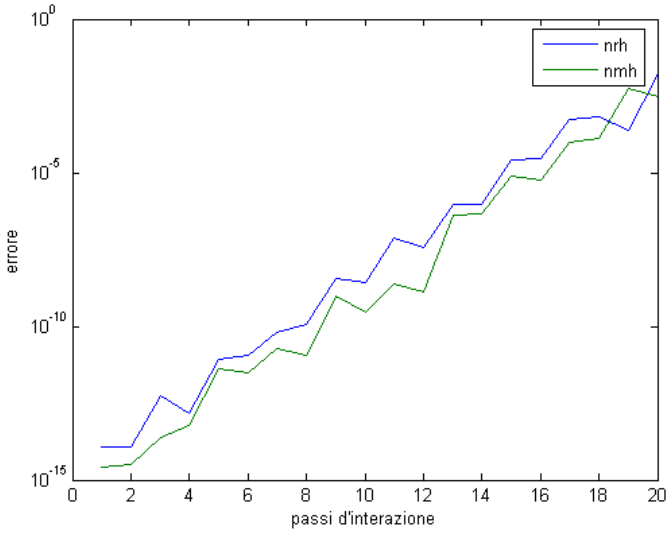

Figura 2 Andamento dell'errore al crescere della dimensione della matrice di Hilbert (caso 2)

Possiamo vedere ancora una volta come al crescere della matrice A di Hilbert, aumenta l'errore dell'algoritmo minq come del resto accade anche per l'algoritmo di Matlab (che comunque si comporta meglio).

Caso 3:

A è una matrice di Pascal di dimensione m x m. L'errore commesso dall'algoritmo viene calcolato (e plottato) al crescere delle dimensioni di A.

```
t=0;
for m=6:14
      t=t+1;
      A=pascal(m);
      b=A*ones(m,1);
      x=A\b;
      Xr{t}=x;
       [x1,residuo]=minq(A,b);
      Xf{t}=x1;
      res(t,1)=residuo;
      nrp(t,1)=norm(x1-ones(m,1));
      nmp(t,1)=norm(x-ones(m,1));
end
semilogy(res)
semilogy([nrp,nmp])
legend('nrp','nmp');
```

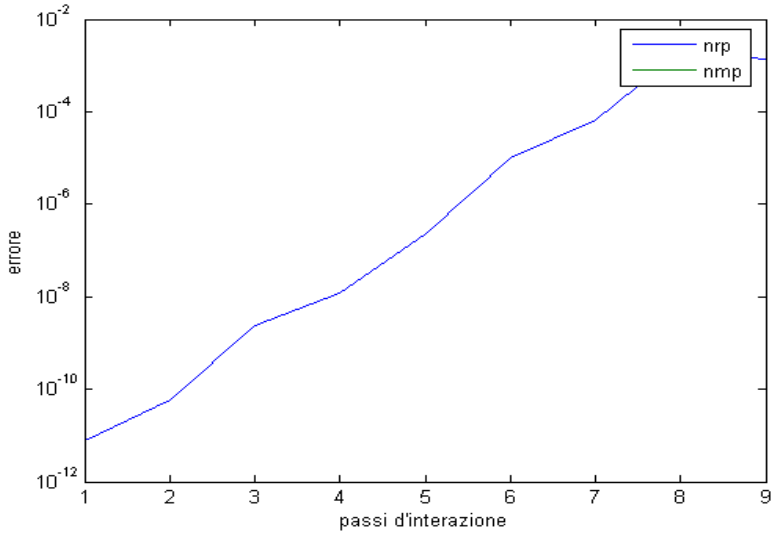

Figura 3 Andamento dell'errore al crescere della dimensione della matrice di Pascal (caso 3)

In quest'ultimo grafico l'errore commesso dall'algoritmo minq è lo stesso che commette Matlab, possiamo infatti vedere che le due curve sono sovrapposte.

L'effetto degli errori commessi dall'algoritmo minq, per la risoluzione del problema dei minimi quadrati, può anche essere esibito attraverso un confronto con la funzione polyfit di Matlab.

```
Funzione test seno

clear all
N=50;
n=9;
t=linspace(-1,1,N+1)';
b=sin(pi*t)+.2*randn(N+1,1); %la funzione campionata
x=linspace(-1,1,201)';
f=sin(pi*x);                 %la funziona reale
a=polyfit(t,b,n);
p=polyval(a,x);              %p è il polinomio che approssima la funzione reale
plot(x,p,x,f,'--',t,b,'o')
legend('p_n(x)','sin(\pix)',4)

%Matrice di Vandermonde
n=9;
X2(:,n+1) = ones(length(t),1,class(t));
for j = n:-1:1
   X2(:,j) = t.*X2(:,j+1);
end

[xt,residuo]=minq(X2,b);
p1=polyval(xt,x);       % p1 è il polinomio che approssima la funzione reale
                        % basandosi sui risultati di minq
figure(2)
plot(x,p1,x,f,'--',t,b,'o')
legend('p_n(x)','sin(\pix)',4)
```

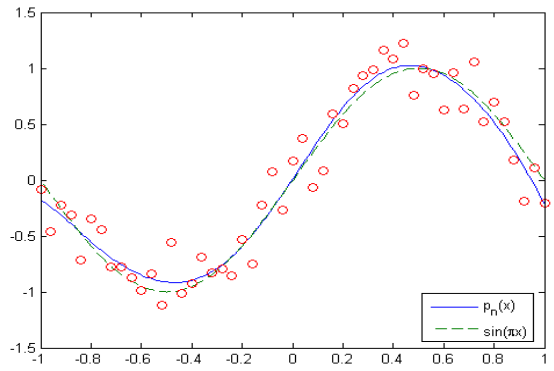

Figura 4 Andamento della curva interpolante il seno calcolata tramite polyfit.

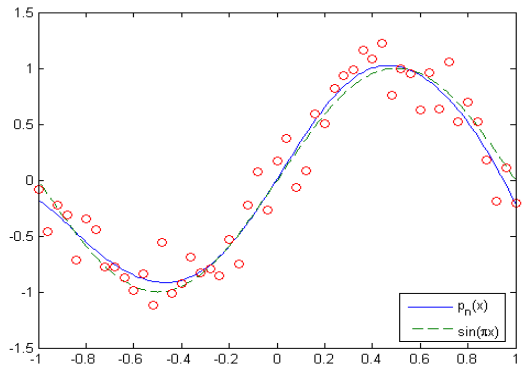

Figura 5 Andamento della curva interpolante il seno calcolata tramite minq.

Le stesse analisi sono poi state ripetute andando ad interpolare un coseno.

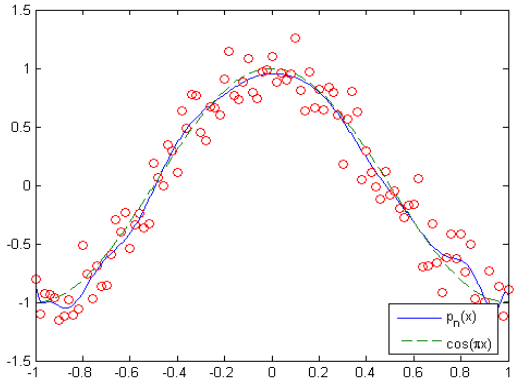

Figura 6 Andamento della curva interpolante il coseno calcolato tramite polyfit.

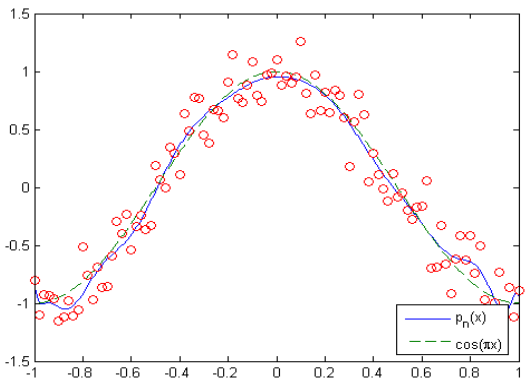

Figura 7 Andamento della curva interpolante il coseno calcolato tramite minq.

Confrontando la figura 4 con 5 e la 6 con la 7 possiamo verificare come polyfit e minq determinino risultati analoghi.

Test SVD

Passiamo ora ad analizzare l'errore che commette Matlab nella fattorizzazione SVD di matrice. Per verificare l'andamento dell'errore applichiamo la SVD a una matrice A generata in maniera random.

```
i=0;
for m=6:100
    i=i+1;
    A=rand(2*m,m);
    [U,S,V]=svd(A);
    Errore_U(i,1)= norm(nansum(U'*U-eye(2*m)), 'fro')/norm(1);
    Errore_V(i,1)= norm(nansum(V'*V-eye(m)), 'fro')/norm(1);
    Errore_decm(i,1)= norm(nansum(A-U*S*V'),'fro')/norm(A,'fro');
end

semilogy([Errore_U,Errore_V,Errore_decm])
legend('Errore_U','Errore_V','Errore_d_e_c_m_p_ _ _q_r');
```

L'algoritmo genera matrici rettangolari random A di dimensione crescente. Ad ogni matrice applica la scomposizione SVD, poi per le matrici U, V e A calcola l'errore commesso.
L'insieme degli errori viene infine plottatato e i suoi risultati possono essere visualizzati nella figura 8.

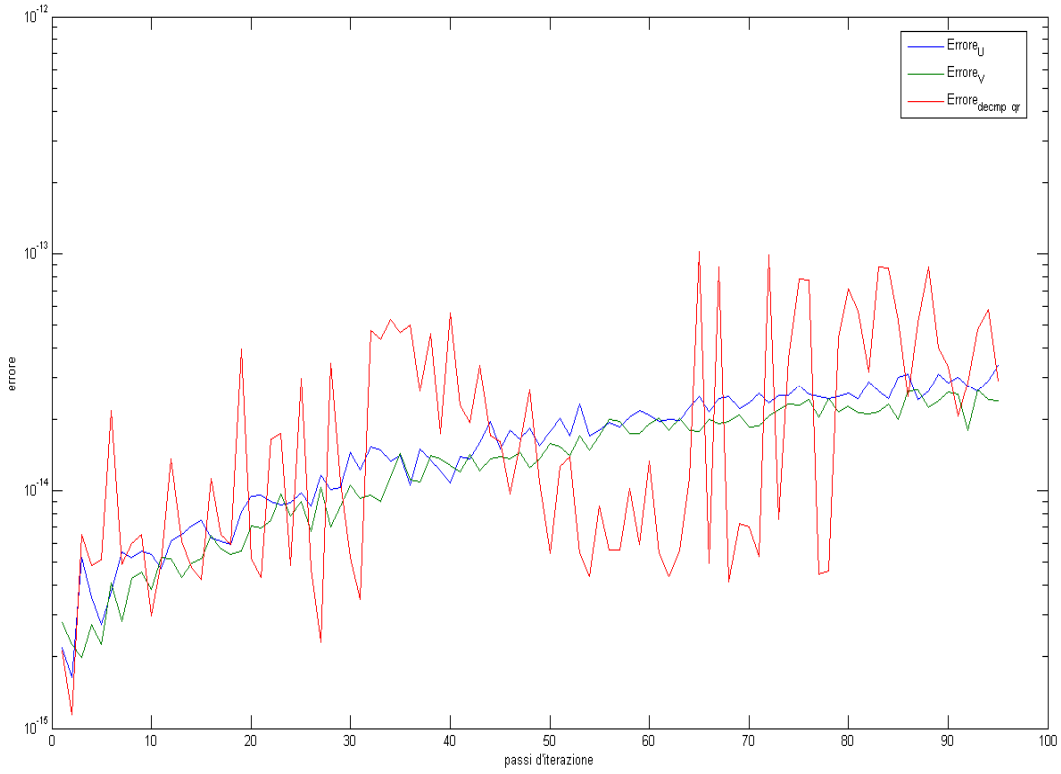

Figura 8 Andamento dell'errore nel calcolo dell'errore delle matrici U, V e A

Come possiamo vedere al crescere della dimensione della matrice A aumenta l'errore commesso dall'algoritmo SVD.

Test TSVD

Nel caso la matrice A presenti un forte malcondizionamento, la sua fattorizzazione SVD può essere sostituita dalla TSVD. Naturalmente anche la TSVD non è immune dall'introduzione d'errori durante la fattorizzazione. In questa sezione verificheremo come gli errori commessi siano dipendenti dal parametro di troncamento k adoperato nell'algoritmo.

Funziona TSVD

```
function [xk] = trunk_svd(U,s,V,b,k)
[p_righe,p_colonne] = size(V);

beta = U(:,1:p_colonne)'*b;
x_bs = beta./s;
for is=1:k
  xk(:,is) = V(:,1:is)*x_bs(1:is); %vettore soluzioni
end
```

Questo algoritmo calcola le soluzioni del problema ai minimi quadrati applicando la TSVD ad una generica matrice A. Prende in ingresso le matrici U e V ottenute della fattorizzazione (4); della matrice diagonale S viene presa la diagonale principale e passata in ingresso come vettore s. Gli altri 2 ingressi sono il parametro k dell'algoritmo, e il vettore dei termini noti b.

Per ogni passo d'iterazione viene calcolato (e memorizzato) il vettore delle soluzioni xk.
Per testare l'utilità della TSVD, la impieghiamo per la fattarizzazione di una matrice quadrata di Hilbert. Il seguente algoritmo permette di verificare il comportamento della TSVD applicata a matrici di Hilbert di diversa dimensione al variare del parametro di troncamento k.

```
%-- primo listato
A=hilb(100);
k=100;              % numero passi d'iterazione
[m,n]=size(A);
[U,S,V]=svd(A);
s=diag(S);
b=A*ones(m,1);
x1=V*((U'*b)./s);

xk = trunk_svd(U,s,V,b,k); %restituisce la matrice con le soluzioni

for i=1:k
    err2(i)=norm(xk(:,i)-ones(m,1));
    condA(i)=s(1)/s(i);       %indice di malcondizionamento
end

semilogy(err2)
legend('err')
figure(2)
semilogy(condA)

k ottimale K=15

%-- secondo listato
m=300;
A=pascal(m);
A=A(1:m,1:((floor(m/2))-1));
[m,n]=size(A);
[U,S,V]=svd(A);
s=diag(S);
b=A*ones(n,1);
xa=(U'*b);
xa=xa(1:n,1);
x1=V*(xa./s);

xk = trunk_svd(U,s,V,b,k); %restituisce la matrice con le soluzioni

for i=1:k
    err2(i)=norm(xk(:,i)-ones(n,1));
    condA(i)=s(1)/s(i);       %indice di malcondizionamento
end

semilogy(err2)
legend('err')
figure(2)
semilogy(condA)


k ottimale K=10
```

Nel primo listato dell'algoritmo prendiamo una matrice di Hilbert (di dimensione[5] 100 x 100), ne determiniamo la fattorizzazione SVD, e poi calcoliamo la soluzioni x1 del sistema Ax=b.

[5] Le problematiche associate al malcondizionamento della matrice A possono già essere osservate su matrici di Hilbert di dimensioni maggiori di 20.

Successivamente calcoliamo la fattorizzazione TSVD con range di variazione del parametro di troncamento che va da 1 a k. La soluzioni xk fornite dall'algoritmo vengono confrontata con il vettore dei termini noti b e il vettore delle soluzioni x2 per determinare l'errore commesso. A conclusione del primo listato plottiamo i risultati ottenuti in scala semilogaritmica.
Il secondo listato è analogo al precedente e si distingue per prendere in esame una matrice di Pascal, per il resto valgono le stesse considerazioni precedenti.
Il blocco di test della TSVD è stato eseguito più volte modificando la dimensione della matrice A e il valore k. I risultati più significativi sono stati riportati di seguito.

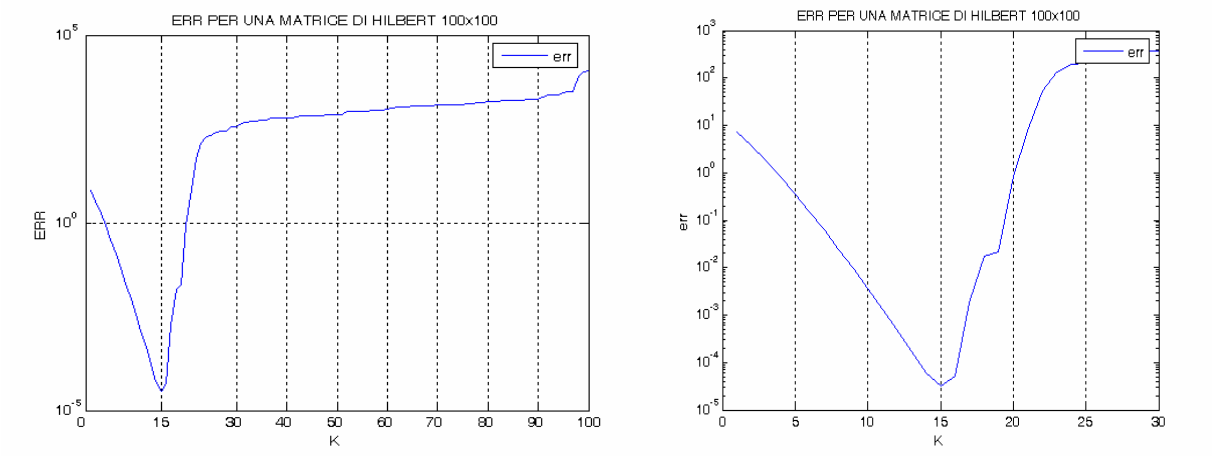

Figura 9 Andamento dell'errore al variare del parametro K nella matrice di Hilbert 100x100

Come possiamo vedere nella figura 9, l'errore nel calcolo della soluzione del sistema Ax=b tende a decrescere all'aumentare del valore k sino a raggiungere il suo valore minimo per k=15. Tale minimo ovviamente non vale per qualsiasi scomposizione TSVD, nella figura 10 infatti possiamo notare come il valore minimo di k, per la matrice di Pascal presa in esame, valga 10.
In entrambi i casi possiamo notare come risulti vantaggioso (per la riduzione dell'errore) prendere in esame i primi valori singolari della matrice A. Poi superata una certa soglia, cioè andando a considerare i restanti valori singolari, la tendenza si inverte e l'errore aumenti.

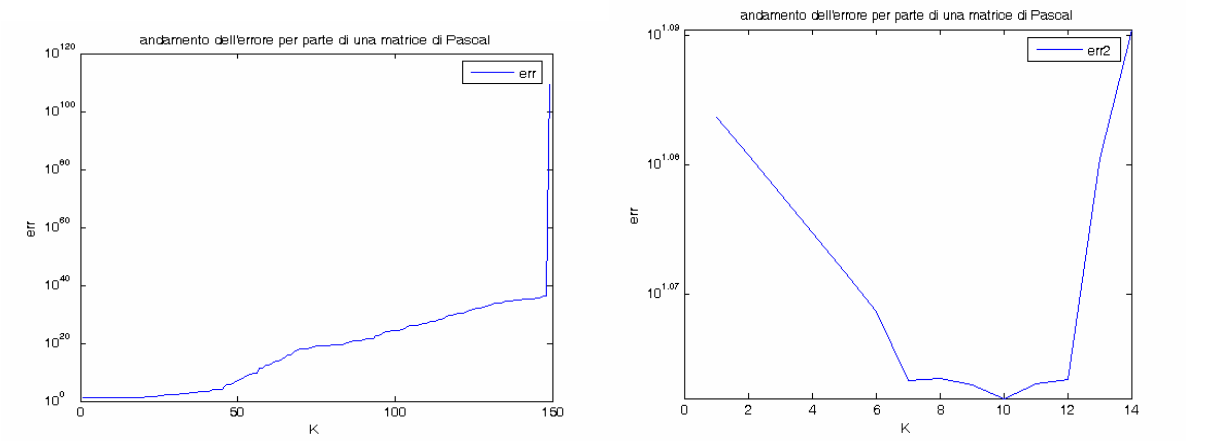

Figura 10 Andamento dell'errore al variare del parametro K nella matrice di Pascal 300x149

Possiamo osservare questo fenomeno anche analizzando la crescita del malcondizionamento della matrice in funzione del parametro k nelle figure 11 e 12 successive.

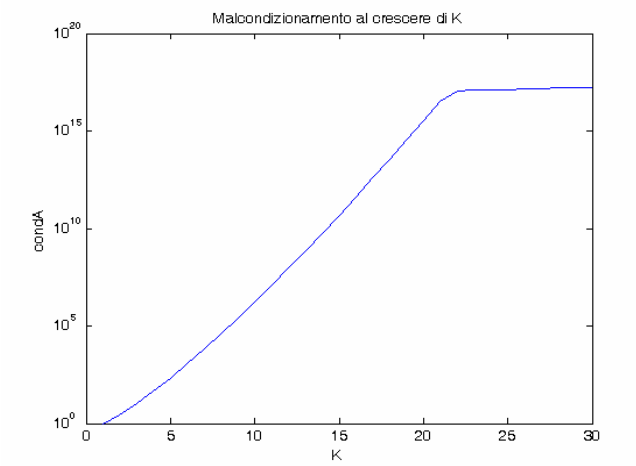

Figura 11 Crescita del malcondizionamento nella matrice di Hilbert 100x100

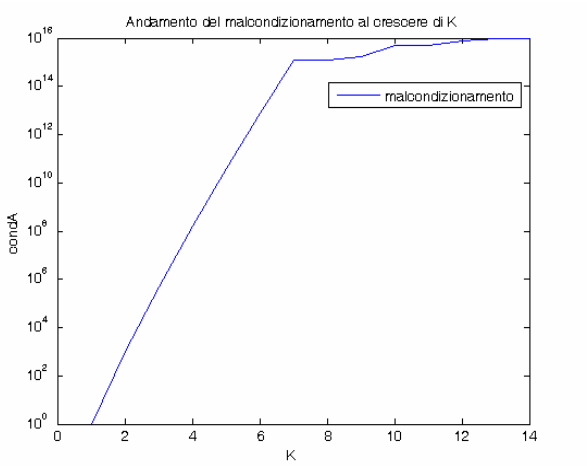

Figura 12 Crescita del malcondizionamento nella matrice di Pascal 300x149

La corretta scelta del parametro k deve bilanciare la crescita del valore del condizionamento e contemporaneamente render il minore possibile l'errore nella soluzione calcolata.

Considerazioni finali

Quando si ha che fare con sistemi lineari malcondizionati l'utilizzo della TSVD è molto utile per trovare una soluzione che sia avvicina maggiormente a quella reale anche quando si è in presenza di un elevato rumore sui termini noti. Lo svantaggio di questa tecnica è l'onere computazionale quando si ha a che fare con sistemi lineari di elevato ordine.