Università degli studi di Cagliari
Corso di laurea Ingegneria Chimica

Esperimenti Numerici sulla fattorizzazione SVD e metodi di regolarizzazione TSVD - L-curve

Seminario di Algebra Lineare Numerica

Professore: Giuseppe Rodriguez

Studenti : Frau Fabio (43440) Porcu Davide(43398)

Anno di corso : 2012/2013

FATTORIZZAZIONE SVD:

Data una qualsiasi matrice $A^{m\ x\ n}$ ad elementi reali, la decomposizione ai valori singolari di A è ua terna di matrici U,∑,V, tali che:

- U ha dimensioni m *x m,* V ha dimensioni *n x n* e sono entrambe matrici ortogonali;
- ∑ è una matrice diagonale di dimensioni *m x n* ad elementi reali corrispondenti ai valori singolari, tanti quanti la dimensione minima della matrice sono maggiori o uguali a zero , gli eventuali posti in diagonale rimanenti sono zeri
- $A = U * \sum * V^T$

Gli elementi σ_i della diagonale hanno inoltre la proprietà di essere ordinati in modo decrescente ed essi sono sempre non negativi.

Un'altra proprietà interessante da notare è che i vettori dati dalle colonne di U e V sono delle basi ortonormali rispettivamente per R^m e per R^n.

La fattorizzazione SVD permette di estendere concetti quali l'inversa anche al caso di matrice rettangolari tramite la pseudoinversa, i valori singolari sono intimamente legati al rango e al condizionamento di una matrice.

Si può dimostrare che i valori singolari di una matrice sono determinanti univocamente e si calcolano nel seguente modo :

data una matrice arbitraria $A^{m\ x\ n}$, la matrice $A^T * A$ è una matrice simmetricasemidefinita positiva , perciò i suoi autovalori sono sempre non negativi $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n \geq 0$ pertanto posso anche esprimerli nella forma $\lambda_k = \sigma_k^2$ in cui i$\sigma_k \ \geq 0$ sono appunto i valori singolari.

La SVD è una fattorizzazione rankrevealing, in quanto il numero di valori singolari diversi da zero corrisponde al rango.

In analisi numerica a causa di inevitabili errori e delle limitatezze intrinseche dell'aritmetica di macchina potrebbe capitare che un numero molto piccolo venga interpretato come uno zero, o il caso opposto in cui durante un calcolo nel quale si dovrebbe ottenere uno zero, invece emerge un numero piccolissimo, ciò causa problemi nello stabilire il rango esatto della matrice.

Le matrici U e V si calcolano invece nel seguente modo:

Le colonne dalla matrice U sono gli autovettori della matrice $A * A^T$, mentre le colonne di V gli autovettori della matrice $A^T * A$.

Applicazionedella SVD per la risoluzionedi sistemi lineari:

Per semplicità ci limitiamo all'analisi di sistemi quadrati a rango pieno ma la SVD si applica anche a sistemi rettangolari e a rango non pieno.

Sia dato un sistema lineare quadrato : $Ax = b$

Usando la SVD il sistema diventa : $U\sum V^T x = b$

Ora moltiplicando a sinistra ambo i membri per $(U\sum V^T)^{-1}$ che equivale a $V\sum^{+}U^T$ ottengo

$$x = V\sum^{+} U^T b$$

Dall'analisi dei precedenti prodotti matriciali si può esprimere x nel seguente modo:

$$\sum_{i=1}^{n} \frac{(u_i^T\ b)v_i}{\sigma_i}$$

Significa che la soluzione x è la combinazione lineare tra i vettori singolari v_i moltiplicati per le proiezioni ortogonali degli u_i su b divise per i valori singolari corrispondenti.

Un modo equivalente è quello di scindere il sistema sistemi più semplici più semplici nel seguente modo:

$$\begin{cases} c = U^T b \\ z = V^T x \\ \sum z = c \end{cases}$$

L'ultimo sistema è diagonale perciò la sua risoluzione è immediata e restituisce il vettore z da usare come termine noto del secondo la cui soluzione x=Vz è anch'essa facilmente calcolabile per l'ortogonalità di V, così come il calcolo del nuovo termine noto c per l'ortogonalità di U.

TSVD:

La truncated SVD calcola la soluzione x troncando al sommatoria al termine k-esimo.

Un'altra interpretazione è quella di usare non la $\sum^{+}$ matrice diagonale in cui ogni elemento diagonale è l'inverso dei valori singolari, ma una matrice in cui i primi k elementi in diagonali sono invariati mentre da k+1 in poi al posto degli inversi dei valori singolari vengono posti degli zeri.

La TSVD permette un approssimazione di una matrice tramite un'altra di rango inferiore, questo può essere utile in diversi da ambiti dalla compressione dei dati alla regolarizzazione delle soluzioni come nel caso che vogliamo esaminare.

La TSVD si comporta come un filtro che taglia i vettori e i valori singolari dal k+1-esimo compreso in poi eliminando qualunque rumore fosse in essi contenuto, poiché i valori singolari decrescono molto rapidamente, saranno gli ultimi che risentiranno di più dell'errore poiché di ordini di grandezza confrontabili o ancora peggio l'errore è più grande del valore singolare stesso.

Il problema sorge nella scelta del parametro k a cui tagliare, esistono vari metodi per la stima di questo parametro noi utilizzeremo la L-curve un metodo euristico ("che favorisce la scoperta") che si avvale di alcune grandezze misurabili quali il residuo e la norma della soluzione.

La L-curve come dice il nome è una curva dal tipico andamento ad L in certe condizioni, essa è disegnata come grafico in scala logaritmica della norma due della soluzione in funzione del residuo $||Ax - b||_2$.

Si sceglie il parametro k come quello situato in corrispondenza dello spigolo della L.

Descrizione dell'algoritmo:

Il seguente algoritmo offre la possibilità di eseguire degli esperimenti numerici sulla fattorizzazione SVD con regolarizzazione mediante TSVD e stima del parametro K tramite la curva L.
La necessità di un metodo di regolarizzazione sorge in quei casi in cui il problema presenta un forte condizionamento, per cui la propagazione degli errori causa un elevato errore nelle soluzioni calcolate.
L'algoritmo è strutturato in modo che si possa scegliere tra 4 problemi di test tramite uno switch menu, i primi tre problemi sono altamente mal condizionati e il loro condizionamento aumenta molto velocemente all'aumentare della dimensione della matrice, mentre il quarto consiste in una matrice random il cui condizionamento è di molti ordini di grandezza più piccolo e cresce molto più lentamente con le dimensioni.
Abbiamo quindi per questo motivo inserito la possibilità di scegliere la dimensione della matrice tramite un input da tastiera, per verificarela buona riuscita del calcolo, delle soluzioni e della successiva regolarizzazione in funzione delle dimensioni della matrice e quindi del suo condizionamento.
Ogni problem test genera una matrice A quadrata di dimensione 'n' , il vettore soluzione 'xr' viene scelto arbitrariamente , quindi siamo in possesso della soluzione esatta in un ambiente protetto a differenza dei casi reali in cui la soluzione non è ovviamente nota.
Questo è necessario perché per poter fare un confronto proficuo tra i vari algoritmi occorre sapere la soluzione esatta e verificare quanto le soluzioni si discostano da quella reale e quando sono tollerabili o meno o in quali casi coincidono.
Inoltre in questo modo possiamo verificare l'efficienza dell'algoritmo nel calcolo della soluzione e ottenere un dato importante che è l'errore definito come ||(xr-x)||, questa variabile non è misurabile in situazioni reali.
Calcoliamo in seguito il vettore dei termini noti 'btrue' come A*xr, questo vettore è virtualmente esente da errori, se non quelli di memorizzazione del calcolatore comunque molto bassi (la precisione di macchina è nell'ordine di 10^{-16}).
Nelle situazioni reali non si verificano mai errori così piccoli, per rendere l'esperimento più vicino alla realtà inseriamo un rumore gaussiano che può essere scalato sul vettore btrue.
Come vedremo in seguito sporcare il vettore b influenza non solo il calcolo delle soluzioni ma anche la buona riuscita della regolarizzazione che è estremamente sensibile alla presenza degli errori.

```
k=input('scegliil problem test:  1)Hilbert   2)Shaw   3)Baart   4)Random  ')
switch k

case 1
%problem test Hilbert
n=input('dammi la dimensione della matrice di hilbert');
A=hilb(n);
xr=ones(n,1); %posso scrivere il vettore soluzione che desidero lo conservo in
xr mentre x sono le soluzioni al passo k
btrue=A*xr;
```

```
case 2
%problem test Shaw
n=input('dammi la dimensione dellamatrice,deve essere un numero pari');
[A,b,xr]=shaw(n);
btrue=A*xr;

case 3
%problem test Baart
n=input('dammi la dimensione della matrice ');
[A,b,xr] = baart(n);
btrue=A*xr;

case 4
%Problem test Random
n=input('dammi la dimensione della matrice');
A=rand(n);
xr=ones(n,1);
btrue=A*xr;

end

ds=input('dammiil ds di errore');
b=btrue+randn(n,1)*ds;
```

L'esperimento confronta due algoritmi , questi dal punto di vista matematico seguono gli stessi principi e servono entrambi per lo stesso scopo, la differenza è dal punto di vista computazionale, e vedremo poi come uno più di un altro giunge a una soluzione migliore.
Grazie ad un altro switch case siamo in grado di scegliere in partenza quale di questi due algoritmi far partire nel workspace di Matlab.
Ancora prima di selezionare l'algoritmo effettuiamo la fattorizzazione SVD necessaria a entrambi, e da essa estraiamo la diagonale contenente i valori singolari.

```
[U S V]= svd(A);
s=diag(S);
```

Il primo algoritmo sfrutta al meglio l'algebra delle matrici, traducendo ed eseguendo la maggior parte possibile dei calcoli in forma matriciale così da poter sfruttare le librerie BLAS (Basic Linear Algebra Subroutines) scritte in parte in linguaggio C in parte in Fortran e ottimizzate per i vari processori e quando possibile per il calcolo in parallelo, questo approccio consente un guadagno computazionale dal punto di vista del tempo di esecuzione e della propagazione degli errori sia per motivi matematici che intrinseci della macchina.

```
case 1
c=(U')*b;
d=c./s ;

for i=1:n
    x=V(:,1:i)*d(1:i,:);
eta(i,:)=norm(d(1:i,:));
etan(i,:)=norm(x);
    rho(i,:)=norm(c(i+1:end,:));
if rho(i,:)==0;
        rho(i,:)=eps;
end
rhon(i,:)=norm(A*x-b);
err(i,:)=norm(xr-x);
    y(:,i)=x;
end
```

Primo passo dell'algoritmo è il calcolo del vettore c e del vettore d che dovranno essere riusati più volte durante l'esecuzione.
Dove c è il vettore ottenuto come U '*b, mentre d è il vettore ottenuto dividendo le varie componenti di c per i rispettivi valori singolari.
Calcoliamo una serie di TSVD,alla prima iterazione usando un solo vettore singolare, alla k-esima k vettori singolari, sino all'utilizzarli tutti che equivale a calcolare le soluzioni di una SVD, realizzato in pratica con un ciclo for per i=1:n, a ogni iterazione servendosi della matrice V, ricavata mediante la SVD, dei vettori c e d calcola la soluzione i-esima x come prodotto matriciale tra una sottomatrice di V che contiene le i colonne di V e una sottovettore di d che contiene le prime i componenti, questo equivale a fare la combinazione lineare dei primi i vettori colonna contenuti in V con i primi i scalari contenuti in d.
Inseriamo la soluzione i-esima in una matrice chiamata y la cui colonna i-esima contente appunto la soluzione i-esima.
Calcoliamo inoltre la norma della soluzione e il residuo .
Da questo punto in poi quando parleremmo di norma si sotto intende la norma due.
La norma della soluzione i-esima viene calcolata come norma delle prime i-esime componenti del vettore d che rappresenta il vettore c diviso per i valori singolari componente per componente, mentre il residuo i-esimo è la norma del vettore dato dalle componenti di c che non vengono usate nel calcolo della soluzioni quindi dalla i+1 sino all'ultima, abbiamo introdotto un accorgimento per evitare che i rho molto piccoli, oltre la precisione di macchina, vengano considerati zero e in questo tipo di problemi sappiamo che i rho non saranno mai uguali a zero per via del condizionamento, un if all'interno del ciclo che controlla ad ogni passo il rho i-esimo e se questo è uguale a zero rho i-esimo prende eps(precisione di macchina).
Un altro modo per calcolare le norme e i residui ma che non sfrutta appieno le potenzialità della fattorizzazione SVD è il seguente: per quanto riguarda la norma calcolare semplicemente la norma della soluzione i-esima, mentre per il residuo calcolare la norma di A*x-b.
Quest'ultimo approccio crea dei problemi nel calcolo del residuo quando si opera con matrici altamente mal condizionate e con vettori di termini noti molto sporchi, infatti i residui calcolati saranno anch'essi affetti da errori.
Infine calcoliamo l'errore assoluto come norma della differenza tra la soluzione i-esima e quella esatta. L'altro algoritmo, che però si è rivelato poco efficiente nel calcolo delle soluzioni e nel calcolo dei residui(come descritti in precedenza) è illustrato qui.

```
case 2
x=((U(:,1)')*b)* V(:,1)./s(1,:);
y(:,1)=x;
eta(1,:)=norm(x);
rho(1,:)=norm(A*x-b);
err(1,:)=norm(xr-x);

for i=2:n
x=x+(((U(:,i)')*b)* V(:,i)./s(i,:));
y(:,i)=x;
eta(i,:)=norm(x);
rho(i,:)=norm(A*x-b);
err(i,:)=norm(xr-x);

end
```

Le soluzioni qui vengo calcolate come $\sum_{i=1}^{k}(u_i^T\ b)v_i/\sigma_i$,a ogni passo calcoliamo il termine i-esimo della soluzione e li sommiamo a quelli precedenti, conserviamo la soluzione i-esima nella solita matrice y.

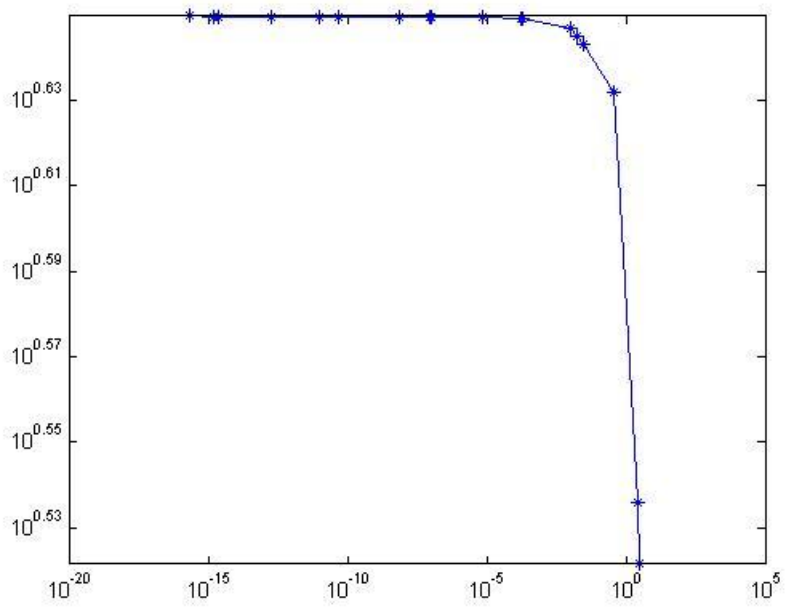
Infine facciamo il plot dei vari grafici:
il primo è il grafico (figure 1) Residui-Norme in scala logaritmica, (quelli ricavati da noi) le due variabili osservabili anche nella pratica a differenza dell'errore che è possibile misurare solo in ambiente protetto.
Il grafico ha la caratteristica di avere la forma a " L" in certe condizioni, e lo spigolo individua una stima del valore a cui troncare la SVD per ottenere una buona soluzione.
Il secondo grafico (figure 2) è analogo al primo solo che è stato ottenuto con il comando L-curve di reg tools.
Il terzo (figure 3) è il grafico in semi scala logaritmica dell'errore assoluto 'err'.
Nell'ultimo infine disegniamo 4 curve che rappresentano le componenti della soluzione . La prima curva è la soluzione esatta da noi imposta, la seconda è la soluzione con errore minore quindi la miglior soluzione che si può sperare di ricavare, la terza è la soluzione calcolata a partire dalla curva L disegnata da noi,mentre l'ultima è quella associata alla L curve di reg tools.

```
figure(1)
loglog(rho,eta,'-*')


figure(2)
[reg_corner,rhol,etal,reg_param] =l_curve(U,s,b,'tsvd');

[errmkmin] = min(err);
corner_nostro=corner(rho,eta);
corner_regtools=corner(rhol,etal);


figure(3)
semilogy(1:n,err,'-o')

figure(4)
plot([xr y(:,kmin) y(:,corner_nostro) y(:,corner_regtools)])
legend('esatta','migliore','nostra','regtools')

fprintf('dimensione matrice:    n=%d\n',n)
fprintf('ds errore introdotto:  ds=%.2e\n',ds)
fprintf('condizionamento A:    cond=%.2e, \n',t)
fprintf('errore minimo:         k=%d, errore=%.2e\n',kmin,err(kmin))
fprintf('curva-L (nostra):      k=%d,
errore=%.2e\n',corner_nostro,err(corner_nostro))
fprintf('curva-L (reg tools):   k=%d,
errore=%.2e\n',corner_regtools,err(corner_regtools))
```

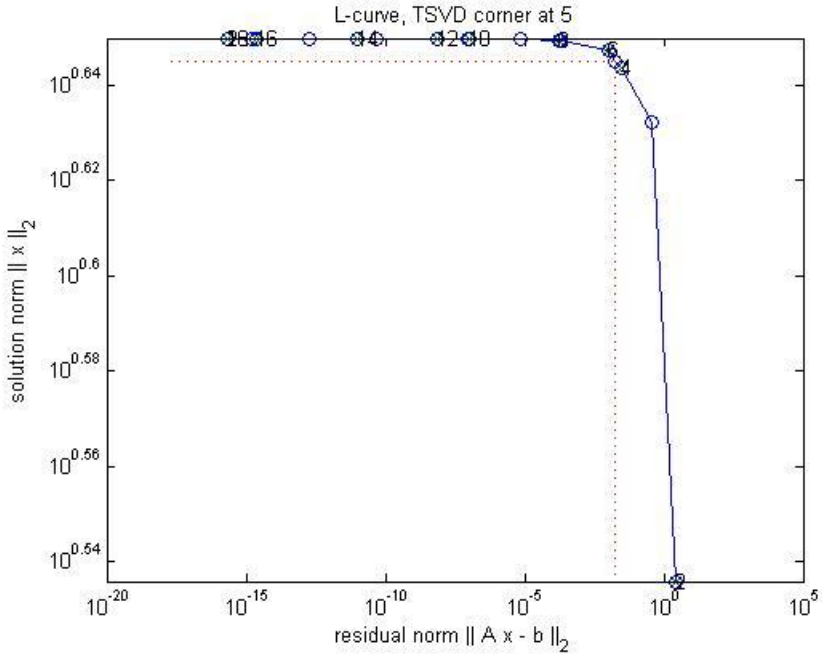
Esperimenti numerici:

Di seguito proponiamo alcuni esperimenti numerici ottenuti con l'algoritmo precedentemente descritto, e utilizzando il problem test shaw variando di volta in volta il 'ds' di errore.

Con un errore pari a zero, ovvero supponendo il caso irreale di non comettere errori nelle misurazioni sperimentali l'algoritmo, il primo, quello che si serve meglio delle BLAS, si comporta in questo modo:



L-curve, grafico rho vs eta, ricavati con il nostro algoritmo.
Si può notare che per questo problema la curva risulta capovolta ma presenta comunque un corner preciso, difficile da individuare a occhio sul grafico a causa degli addensamenti di punti in prossimità di questo, ma misurabile con la funzione 'corner' di reg-tools



Questo è il grafico L-curve roh vs eta, con rho ed eta ricavati con la funzione 'corner' di reg tools, il grafico è pressoché identifico al precedente in quanto i roh e gli eta coincidono o sono estremamente simili.

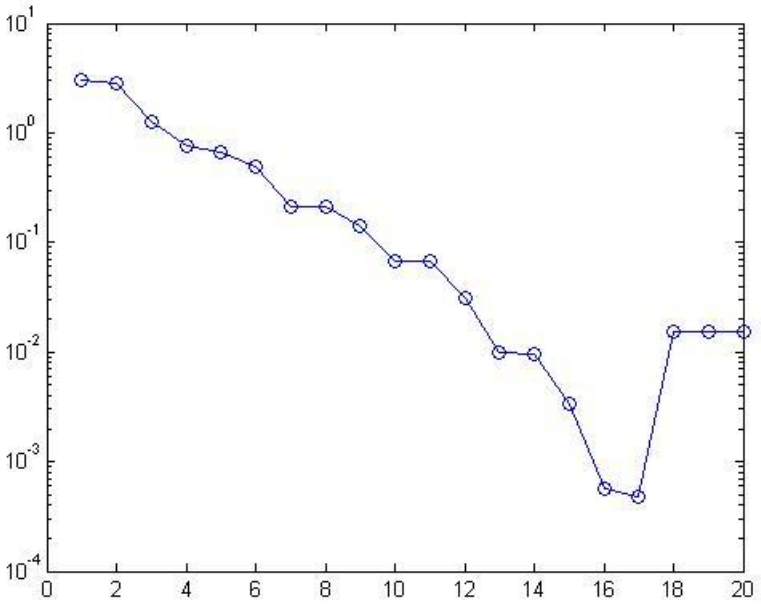


Grafico semilogaritmico dell'errore al variare del numero di vettori singolari usati.

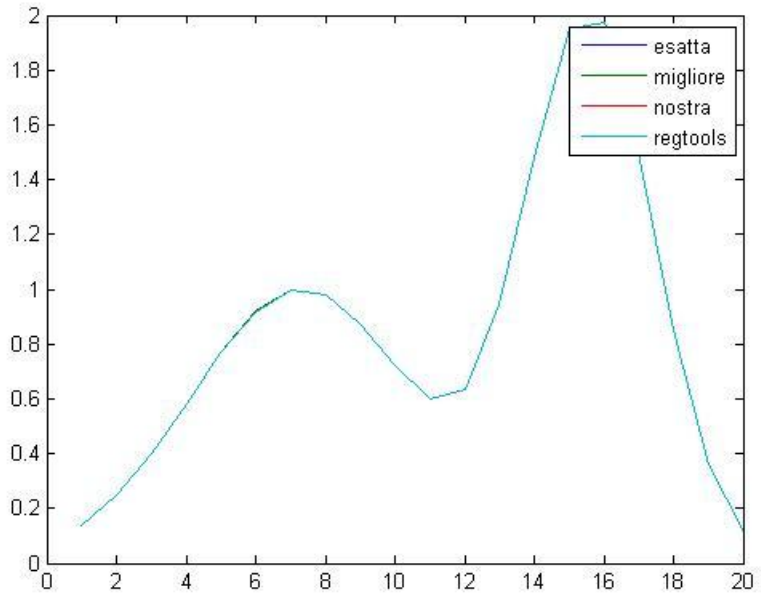


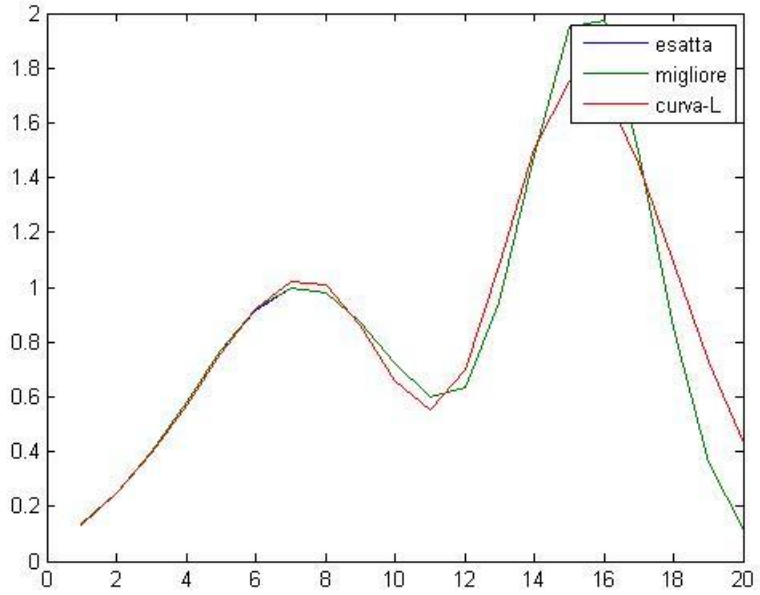
Grafico delle soluzioni, ogni curva rappresenta una diversa soluzione stimando il valore ottimale di troncamento k usando L-curve diverse e confrontandole con quella esatta e la migliore che presenta l'errore minimo, in tale caso essendo l'errore sulle varie soluzioni sufficientemente piccolo e vicino esse appaiono coincidenti.

Dati esperimento:

dimensione matrice: n=20
ds errore introdotto: ds=0.00e+00
condizionamento A: cond=1.03e+16,
errore minimo: k=17, errore=4.74e-04
curva-L (nostra): k=16, errore=5.67e-04
curva-L (reg tools): k=16, errore=5.67e-04

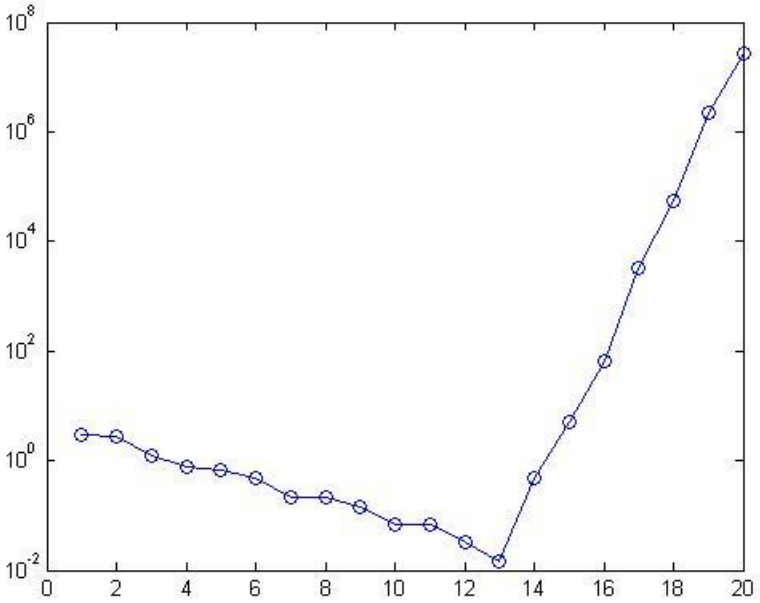
Dai grafici si può notare che in questo caso si giunge quasi alla soluzione migliore ovvero quella reale. Abbiamo così un'ottimma approssimazione della soluzione . Le cose cambiano moltissimo se si usa il secondo algoritmo che non riesce a stimare il parametro ottimale di troncamento,lo si può notare dal grafico delle soluzioni qui sotto riportato.

errore minimo: k=17, errore=4.93e-04
curva-L (nostra): k=5, errore=6.57e-01
curva-L: k=17, errore=4.93e-04

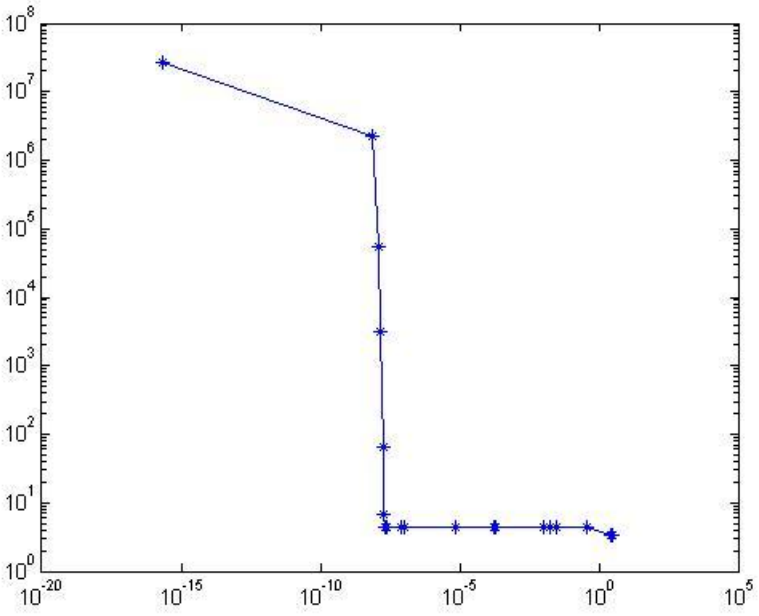


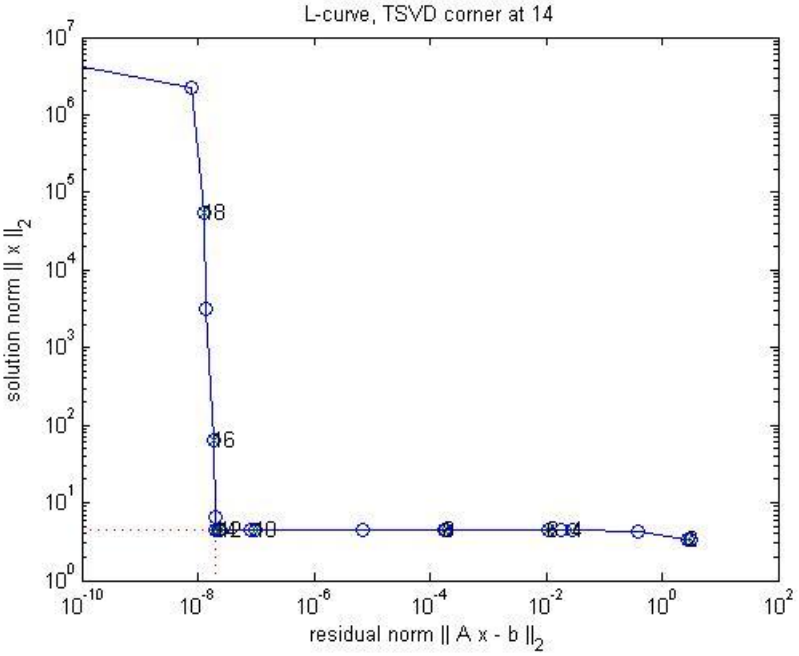
La soluzione migliore non viene opportunamente evidenziata in quanto coincide quasi perfettamente con quella reale.

Osserviamo ora cosa accade alle soluzioni se introduciamo un errore anche se piccolo.

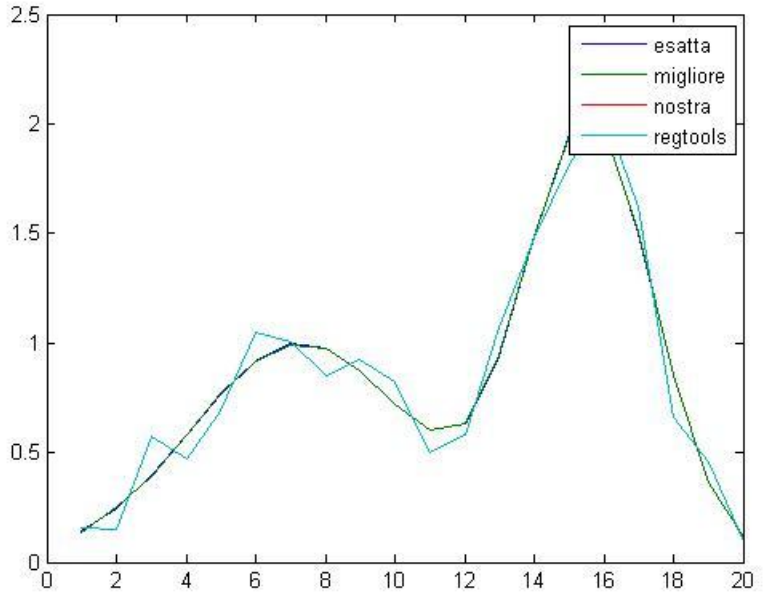


Notiamo subito che all'aumentare del ds di errore inserito il parametro ottimale di troncamento si sposta verso sinistra, questo perché il peso dell'errore diventa influente su un numero maggiore di vettori singolari (gli ultimi), evidenziato dalla veloce crescita dell'errore per i vettori successivi al k ottimale.





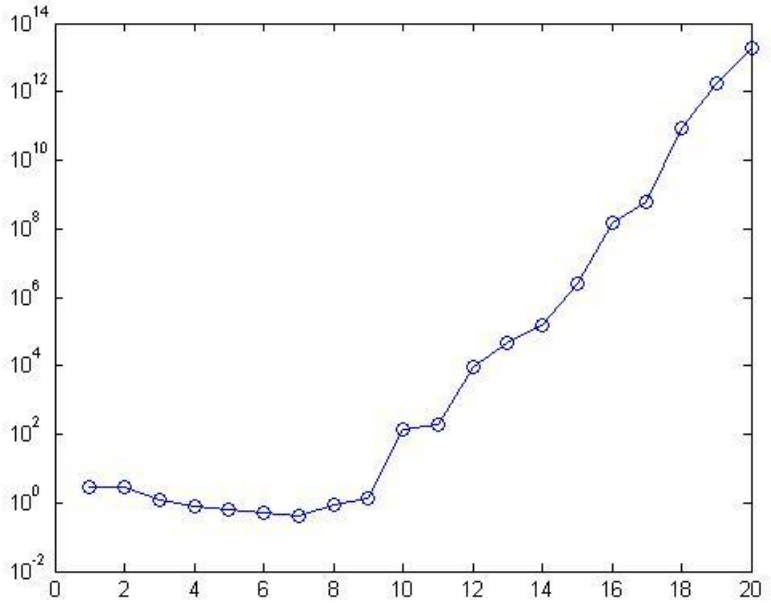
In questo caso la L-curve ha il suo caratteristico aspetto ad L, determinato dal fatto che è un metodo sensibile agli errori, l'assenza di questi o una presenza errori troppo grandi ne limita l'affidabilità nella stima del parametro k, e ne distorcono la forma.



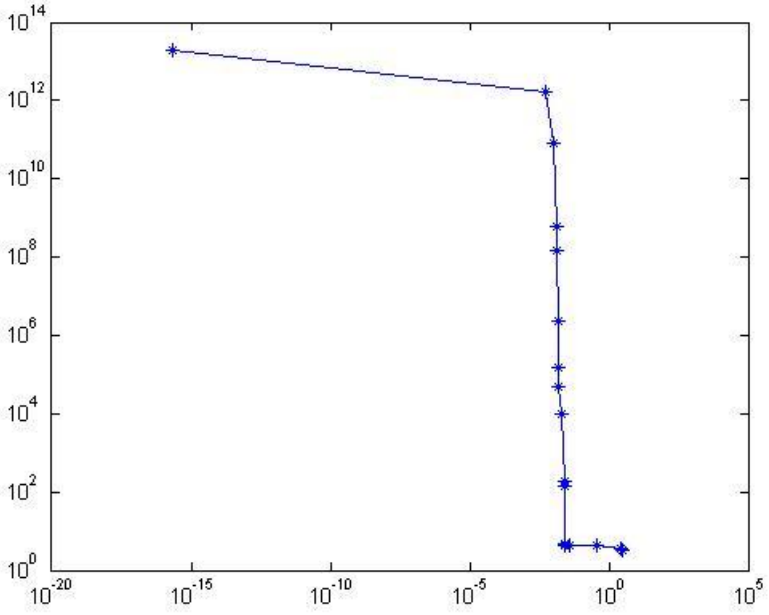
Le soluzioni calcolate tramite la nostra curva e reg tools coincidono, anche se non sono le migliori che si possono ottenere.

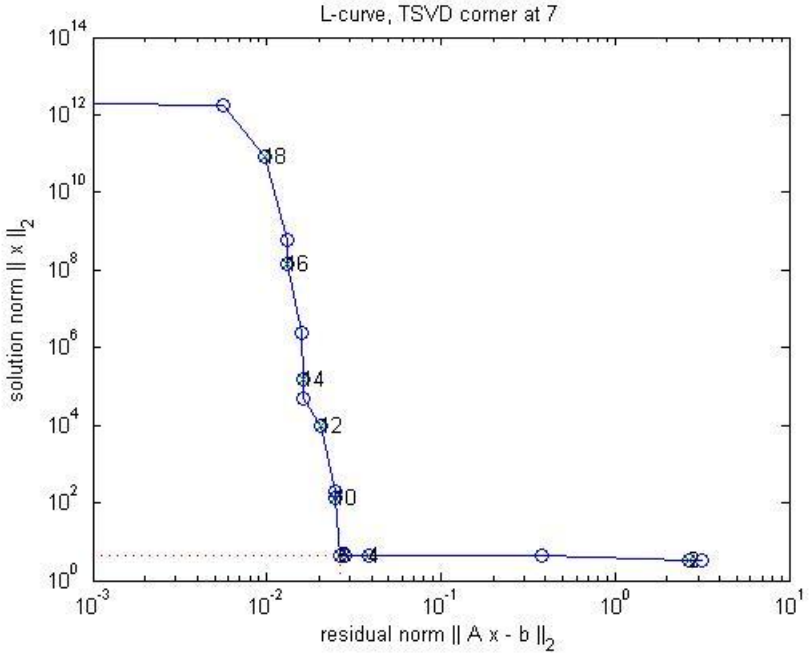
La curva migliore ottenibile segue molto bene la soluzione reale, ma anche le soluzioni calcolate seguono bene per un certo tratto la soluzione esatta, mentre in quello iniziale vi oscillano leggermente intorno.

dimensione matrice: n=20
ds errore introdotto: ds=1.00e-08
condizionamento A: cond=1.03e+16,
errore minimo: k=13, errore=1.44e-02
curva-L (nostra): k=14, errore=4.67e-01
curva-L (reg tools): k=14, errore=4.67e-01

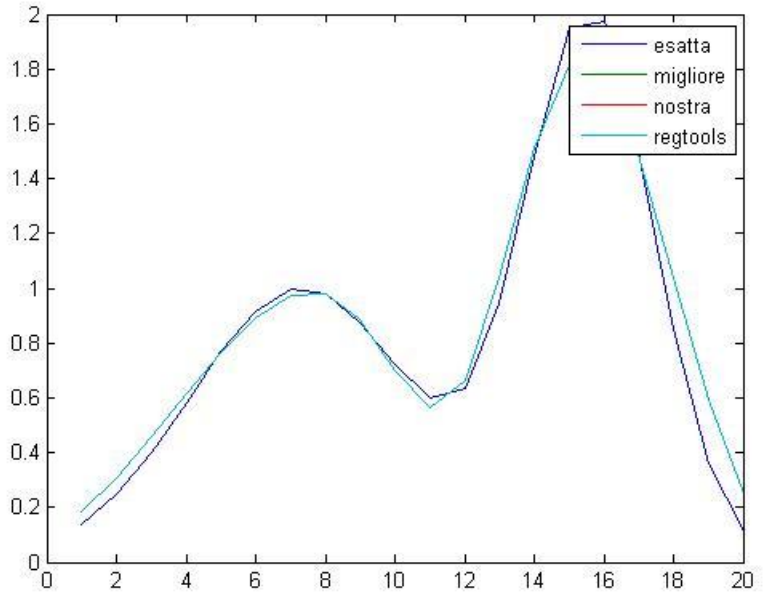


Notiamo un ulteriore spostamento verso sinistra del parametro di troncamento come precedentemente descritto





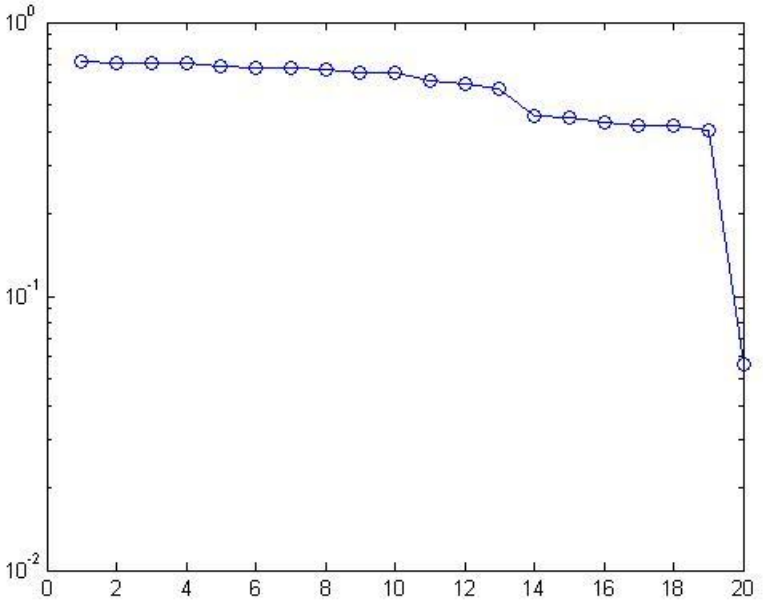
L'andamento della curva è quello tipico, questo è importante poiché ci troviamo nell'ordine di grandezza degli errori che si verificano nei casi pratici, perciò il metodo è effettivamente utile e usufruibile nei casi pratici che necessitano di regolarizzazione offrendoci un certo grado di sicurezza.



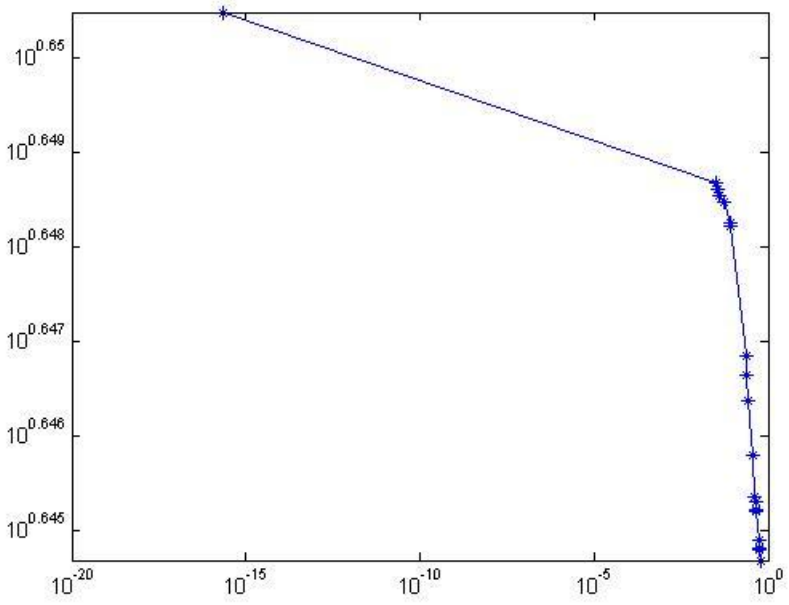
In questo caso le soluzioni calcolate coincidono entrambe con la soluzione migliore ottenibile ed entrambe hanno un errore che può essere accettabile per molti casi pratici.
Ulteriore conferma della sensibilità agli errori da parte del metodo è il fatto che paradossalmente con un errore di 10^-2 si sia stimata la soluzione migliore , mentre con errori più piccoli non si è mai raggiunta.

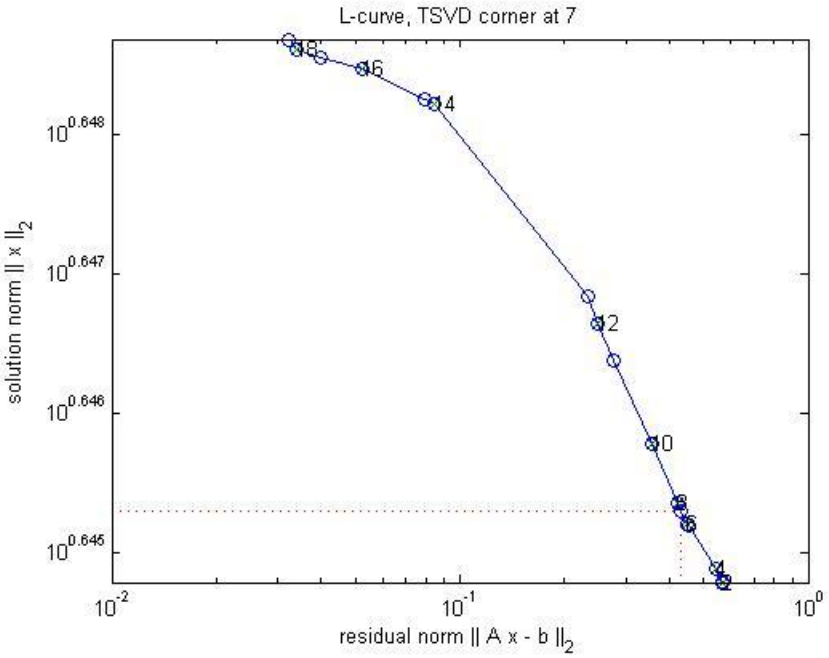
dimensione matrice: n=20
ds errore introdotto: ds=1.00e-02
condizionamento A: cond=1.03e+16,
errore minimo: k=7, errore=4.26e-01
curva-L (nostra): k=7, errore=4.26e-01
curva-L (reg tools): k=7, errore=4.26e-01

Ora eseguiamo lo stesso esperimento su un problema ben condizionato costituito da una semplice matrice random con errori nell'ordine di 10^{-2}



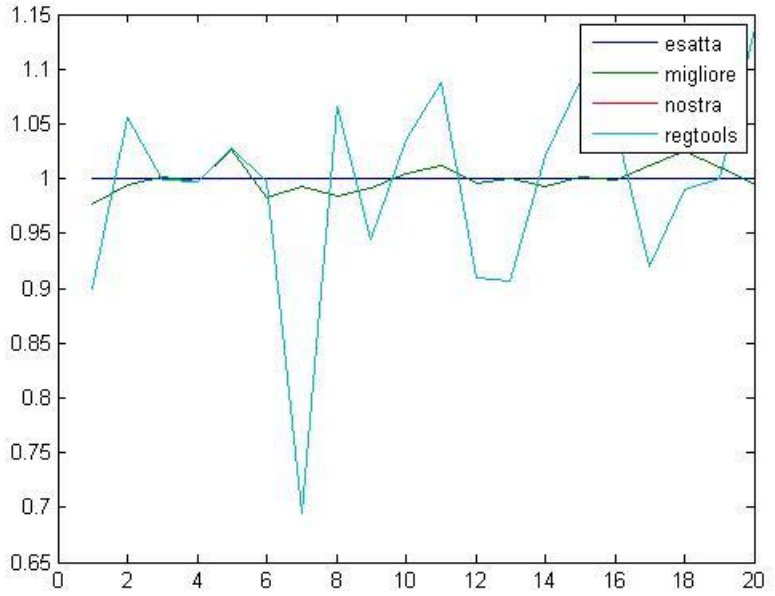
Nel caso di una matrice con condizionamento basso l'errore minimo si ha utilizzando tutti i vettori singolari, perciò non vi è bisogno di metodi di regolarizzazione.





La curva nonostante la presenza di un errore di 10^{-2} perde il suo andamento ad L a causa del basso condizionamento della matrice A che limita la propagazione degli errori.

Il calcolo del corner in questo caso perde di significato poiché sappiamo che per problemi ben condizionati non serve regolarizzare la soluzione perciò il parametro stimato k non offrirà sempre una soluzione accettabile.



Le soluzioni calcolate nonostante tutto sono molto buone, fanno molte oscillazioni ma con un ampiezza molto piccola perciò sono più che accettabili, nonostante non raggiungano la soluzione migliore.

dimensione matrice: n=20
ds errore introdotto: ds=1.00e-02
condizionamento A: cond=1.27e+02,
errore minimo: k=20, errore=5.63e-02
curva-L (nostra): k=18, errore=4.21e-01
curva-L (reg tools): k=18, errore=4.21e-01

Programmacompleto

```
 clear all

k=input('scegliil problem test:  1)Hilbert   2)Shaw   3)Baart   4)Random  ')
switch k

case 1
%problem test Hilbert
n=input('dammi la dimensione della matrice di hilbert');
A=hilb(n);
xr=ones(n,1);
btrue=A*xr;

case 2
%problem test Shaw
n=input('dammi la dimensione della matrice, deve essere un numero pari');
[A,b,xr]=shaw(n);
btrue=A*xr;

case 3
%problem test Baart
 n=input('dammi la dimensione della matrice ');
[A,b,xr] = baart(n);
btrue=A*xr;

case 4
%Problem test Random
n=input('dammi la dimensione della matrice');
A=rand(n);
xr=ones(n,1);
btrue=A*xr;

end

ds=input('dammiil ds di errore');
b=btrue+randn(n,1)*ds;

t=cond(A);

[U S V]= svd(A);
s=diag(S);


k=input('scegli l algoritmo 1) ottimizzato 2)non ottimizzato')
switch k

case 1
c=(U')*b;
d=c./s ;
for i=1:n
x=V(:,1:i)*d(1:i,:);
eta(i,:)=norm(d(1:i,:));
etan(i,:)=norm(x);
rho(i,:)=norm(c(i+1:end,:));
if rho(i,:)==0;
rho(i,:)=eps;
end
```

```
rhon(i,:)=norm(A*x-b);
err(i,:)=norm(xr-x);
y(:,i)=x;


end

case 2
x=((U(:,1)')*b)* V(:,1)./s(1,:);
y(:,1)=x;
       eta(1,:)=norm(x);
       rho(1,:)=norm(A*x-b);
       err(1,:)=norm(xr-x);

for i=2:n
            x=x+(((U(:,i)')*b)* V(:,i)./s(i,:));
            y(:,i)=x;
            eta(i,:)=norm(x);
            rho(i,:)=norm(A*x-b);
            err(i,:)=norm(xr-x);

end

end

figure(1)
loglog(rho,eta,'-*')


figure(2)
[reg_corner,rhol,etal,reg_param] =l_curve(U,s,b,'tsvd');

[errmkmin] = min(err);
corner_nostro=corner(rho,eta);
corner_regtools=corner(rhol,etal);


figure(3)
semilogy(1:n,err,'-o')

figure(4)
plot([xr y(:,kmin) y(:,corner_nostro) y(:,corner_regtools)])
legend('esatta','migliore','nostra','regtools')

fprintf('dimensione matrice:    n=%d\n',n)
fprintf('ds errore introdotto:  ds=%.2e\n',ds)
fprintf('condizionamento A:    cond=%.2e, \n',t)
fprintf('errore minimo:         k=%d, errore=%.2e\n',kmin,err(kmin))
fprintf('curva-L (nostra):      k=%d,
errore=%.2e\n',corner_nostro,err(corner_nostro))
fprintf('curva-L (reg tools):   k=%d,
errore=%.2e\n',corner_regtools,err(corner_regtools))
```