UNIVERSITA' DEGLI STUDI DI CAGLIARI
DIPARTIMENTO DI INGEGNERIA ELETTRICA ED ELETTRONICA

Algoritmi per la regolarizzazione e per il calcolo della SVD

Implementazione e 'benchmark' comparativi

Samuele Lilliu
n.matricola 36156
Ingegneria Elettronica Laurea Specialistica
8/3/2007

Corso di Calcolo Numerico 2
Prof. G. Rodriguez
Minicorso di Algebra Lineare Numerica
Instructor: Claude Brezinski
Title: *Numerical linear algebra: tools and methods*
Credits: 2 credit AA
Organizer: *G. Rodriguez e A. Arico'*

Indice dettagliato

1. Introduzione

1.1. La necessità di regolarizzare

Si consideri il problema di minimi quadrati:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2 \tag{1.1}$$

Sia A mal condizionata e $\mathbf{b} = \mathrm{A}\bar{\mathbf{x}} + \boldsymbol{e}$, dove $\bar{\mathbf{x}}$ è la soluzione esatta ed $\boldsymbol{e}$ una piccola perturbazione. La soluzione $\mathbf{x}_{LSQ}$ di (1.1) è molto sensibile alle perturbazioni dei dati e può essere priva di valore.
La difficoltà maggiore nella ricerca della soluzione ordinaria ai minimi quadrati $\mathbf{x}_{LSQ}$ è che la sua norma è molto più grande della norma della soluzione esatta, per cui si potrebbe imporre un ulteriore vincolo alla (1.1):

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2 \text{ tale che } \|\mathbf{x}\|_2 \leq \alpha \tag{1.2}$$

La soluzione calcolata $\mathbf{x}_\alpha$ dipende in modo non lineare da α. Una scelta accurata (e non ovvia) di questo parametro conduce a $\mathbf{x}_\alpha \approx \bar{\mathbf{x}}$. Nei problemi discreti mal posti si incontrano tre difficoltà:

1. Il numero di condizionamento di A è grande;
2. Rimpiazzare A con una matrice ben condizionata ricavata da A non conduce necessariamente a soluzioni utili;
3. Occorre stabilire delle regole per fissare vincoli addizionali.

L'obiettivo della teoria della regolarizzazione numerica è quello di fornire metodi efficaci e numericamente stabili per introdurre opportuni vincoli che conducano a soluzioni utili stabilizzate, e di fornire metodi robusti per la scelta dei parametri per questi vincoli in modo che la soluzione regolarizzata sia una buona approssimazione della soluzione desiderata.

1.2. REGULARIZATION TOOLS e calcolo della SVD

In questo lavoro si farà uso delle funzioni contenute nel REGULARIZATION TOOLS V. 3.1 sviluppato da C.Hansen. Questo pacchetto fornisce degli strumenti per l'analisi di problemi mal posti discreti, la visualizzazione delle loro proprietà, una semplice generazione di problemi test. È possibile sperimentare diverse strategie di regolarizzazione, confrontarle, e trarre conclusioni da questi esperimenti.
Il primo passo del processo di regolarizzazione di un problema consiste, come si vedrà, nel calcolo dei valori singolari ottenuto mediante la decomposizione a valori singolari (SVD) della matrice dei coefficienti. Per questo motivo saranno mostrati diversi approcci al calcolo di questo utilissimo strumento.

2. SVD[1]

La *decomposizione a valori singolari* (SVD) consente di fattorizzare una matrice complessa rettangolare. Il teorema spettrale afferma che le matrici normali possono essere unitariamente diagonalizzate usando gli autovettori. La SVD può essere vista come una generalizzazione del teorema spettrale per matrici arbitrarie, non necessariamente quadrate. Detto in altro modo, qualsiasi matrice può essere ridotta in forma diagonale[2] tramite una pre- e post- moltiplicazione per una matrice unitaria.
Sia $A \in \mathbb{C}^{m\times n}$, allora esistono due matrici unitarie $U \in \mathbb{C}^{m\times m}$ e $V \in \mathbb{C}^{n\times n}$ tali che:

$$U^*AV = \Sigma = \mathrm{diag}\big(\sigma_1, \dots, \sigma_p, 0, \dots, 0\big) \in \mathbb{C}^{m\times n} \quad \text{con} \quad p = \mathrm{rank}(A) \tag{2.1}$$

inoltre $\sigma_1 > \cdots > \sigma_p > 0$. La formula (2.1) è detta SVD di A e i numeri σ_i (oppure $\sigma_i(\mathrm{A})$) sono detti *valori singolari* di A (1).

2.1. Valori singolari, vettori singolari, relazioni

È possibile dare una definizione alternativa della (2.1). Un numero reale positivo σ_i è detto *valore singolare* di A se e solo se esistono i vettori di lunghezza unitaria $\mathbf{u}_i \in \mathbb{C}^m$ e $\mathbf{v}_i \in \mathbb{C}^n$ tali che:

$$\begin{aligned} A\mathbf{v}_i &= \sigma_i \mathbf{u}_i \\ A^*\mathbf{u}_i &= \sigma_i \mathbf{v}_i \end{aligned} \tag{2.2}$$

La (2.2) e la (2.1) coincidono, infatti:

$$AV = U\Sigma$$
$$A\mathbf{v}_i = \sigma_i \mathbf{u}_i \quad \text{per} \quad i = 1 \dots m \tag{2.3}$$

$$U^*A = \Sigma V^*$$
$$A^*\mathbf{u}_i = \sigma_i \mathbf{v}_i \quad \text{per} \quad i = 1 \dots n \tag{2.4}$$

I vettori $\mathbf{u}, \mathbf{v}$ sono detti *vettore singolare sinistro* e *vettore singolare destro* per σ. In ogni decomposizione a valori singolare, gli elementi della diagonale di Σ sono i valori singolari di A. Le colonne di U e V contengono i vettori singolari sinistro e destro. Perciò:

1. È sempre possibile trovare una base unitaria per $\mathbb{C}^m$ costituita da U;
2. È sempre possibile trovare una base unitaria per $\mathbb{C}^n$ costituita da V;

Un valore singolare per il quale possiamo trovare due vettori singolari sinistro (o destro) linearmente indipendenti è detto *degenere.* Quindi la SVD è unica se tutti i valori singolari sono *non degeneri.*

2.2. Relazioni per *A* quadrata

È possibile riscrivere le precedenti relazioni e trovarne di nuove. Supponiamo che A sia quadrata e reale:

$$A = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i {\mathbf{v}_i}^T \tag{2.5}$$

[1] La trattazione che segue fa riferimento a (11)(12)(13)(14)(15)(16)(17).
[2] Matrice diagonale rettangolare.

$$A^*\mathbf{u}_i = \sum_{i=1}^{n} \sigma_i \mathbf{v}_i \mathbf{u}_i^T \tag{2.6}$$

$$A^{-1} = (U\Sigma V^*)^{-1} = V\Sigma^{-1}U^* = \sum_{i=1}^{n} \left(\frac{1}{\sigma_i}\right) \mathbf{v}_i \mathbf{u}_i^T \tag{2.7}$$

2.2.1. Relazioni con gli autovalori (A hermitiana, semidefinita positiva)

La SVD è veramente generale nel senso che può essere applicata a qualsiasi matrice rettangolare. La decomposizione in autovalori invece può essere applicata solo a certe matrici quadrate. Si può mostrare che le due decomposizioni sono legate.

Nel caso in cui A è una matrice *hermitiana semidefinita positiva*[3], allora i valori singolari e i vettori singolari coincidono con gli autovalori e gli autovettori di A:

$$A = V\Lambda V^* \tag{2.8}$$

Data una SVD di A si ha:

$$\begin{aligned} A^*A &= V\Sigma U^* U\Sigma V^* = V\Sigma^2 V^* \\ AA^* &= U\Sigma V^* V\Sigma U^* = U\Sigma^2 U^* \end{aligned} \tag{2.9}$$

Il lato destro di queste relazioni descrive la decomposizione per autovalori del lato sinistro. Di conseguenza i valori singolari non nulli di A sono uguali agli autovalori non nulli sia di A^*A che di AA^*. Inoltre, le colonne di U sono autovettori di AA^*, mentre le colonne di V sono autovettori di AA^*.

$$\sigma_i(A) = \sqrt{\lambda_i(A^*A)} \tag{2.10}$$

La (2.10) indica che se $A \in \mathbb{C}^{m\times n}$ è (quadrata ed) hermitiana con autovalori $\lambda_1, \dots, \lambda_n$ allora i valori singolari di A coincidono con i moduli degli autovalori di A. Questo perché $AA^* = A^2, \sigma_i(A) = |\lambda_i|$ per $i = 1, \dots, n$.

Per quanto concerne il rango, se

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0 \tag{2.11}$$

Allora il rango di A è r.

2.3. Dimostrazione dell'esistenza

Per dimostrare l'esistenza della SVD si possono impiegare due argomenti. Qua si usa solo l'argomento di algebra lineare (l'altro è l'approccio variazionale).

Sia $A \in \mathbb{C}^{m\times n}$, A^*A è semidefinita positiva ed hermitiana. Grazie al teorema dello spettro, esiste una matrice unitaria $V \in \mathbb{C}^{n\times n}$ tale che:

$$V^*A^*AV = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix} \tag{2.12}$$

Dove D è diagonale e definita positiva. Partizionando V opportunamente[4] possiamo scrivere:

[3] Tutti i suoi autovalori sono reali e non negativi

[4] Il numero delle righe di V_1 è uguale al numero delle righe di D e così via.

$$\begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix} A^* A [V_1 \quad V_2] = \begin{bmatrix} V_1^* A^* A V_1 & V_1^* A^* A V_2 \\ V_2^* A^* A V_1 & V_2^* A^* A V_2 \end{bmatrix} = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix} \tag{2.13}$$

Perciò $V_1^* A^* A V_1 = D$ ed $AV_2 = 0$. Sia inoltre:

$$U_1 = AV_1 D^{-\frac{1}{2}} \tag{2.14}$$

Allora

$$U_1 D^{\frac{1}{2}} V_1^* = AV_1 V_1^* = A \tag{2.15}$$

Si badi che U_1 e V_1 non sono unitarie in generale. Sia U_2 tale che:

$$U = [U_1 \quad U_2] \tag{2.16}$$

Risulta:

$$U_1 U_1^* = I_p \quad e \quad U_1 U_1^* + U_2 U_2^* = I_n \tag{2.17}$$

Sia unitario. Si definisce:

$$\Sigma = \begin{bmatrix} \begin{bmatrix} D^{\frac{1}{2}} & 0 \\ 0 & 0 \end{bmatrix} \\ 0 \end{bmatrix} \tag{2.18}$$

A questo punto:

$$[U_1 \quad U_2] \begin{bmatrix} \begin{bmatrix} D^{\frac{1}{2}} & 0 \\ 0 & 0 \end{bmatrix} \\ 0 \end{bmatrix} [V_1 \quad V_2]^* = [U_1 \quad U_2] \begin{bmatrix} D^{\frac{1}{2}} V_1^* \\ 0 \end{bmatrix} = U_1 D^{\frac{1}{2}} V_1^* = A \tag{2.19}$$

Quindi:

$$A = U \Sigma V^* \tag{2.20}$$

2.4. Applicazioni della SVD

2.4.1. Pseudoinversa[5]

La SVD può essere usata per il calcolo della *pseudoinversa* di una matrice. La pseudoinversa $A^\dagger$ di una matrice A rettangolare è la generalizzazione della matrice inversa. Questa matrice è impiegata tipicamente per il calcolo della soluzione 'best fit' (ai minimi quadrati) di un sistema di equazioni lineari. Tale matrice è definita e unica per ogni matrice. Si calcola usando la SVD.

Definizione

La pseudoinversa $A^\dagger$ di una matrice A è una matrice unica che soddisfa i seguenti criteri:

1. $AA^\dagger A = A$;
2. $A^\dagger AA^\dagger = A^\dagger$, ($A^\dagger$ è un'inversa debole per il semigruppo moltiplicativo);

[5] La trattazione sulla pseudoinversa fa riferimento a (18) (19)(20) (21).

3. $\left(AA^{\dagger}\right)^{*} = AA^{\dagger}$, ($AA^{\dagger}$ è Hermitiana)
4. $\left(A^{\dagger}A\right)^{*} = A^{\dagger}A$, ($A^{\dagger}A$ è Hermitiana)

Un modo alternativo per definire la pseudoinversa è tramite il limite:

$$A^{\dagger} = \lim_{\delta \to 0}(A^{*}A + \delta I)^{-1}A^{*} = \lim_{\delta \to 0} A^{*}(AA^{*} + \delta I)^{-1} \tag{2.21}$$

Questi limiti esistono anche se $(A^{*}A)^{-1}$ e $(AA^{*})^{-1}$ non esistono.

Proprietà

Alcune proprietà

1. $\left(A^{\dagger}\right)^{\dagger} = A$;
2. La pseudoinversa commuta con la trasposta e la coniugata:
   a. $(A^{T})^{\dagger} = \left(A^{\dagger}\right)^{T}$;
   b. $(A^{*})^{\dagger} = \left(A^{\dagger}\right)^{*}$;
3. $(\alpha A)^{\dagger} = \alpha^{-1}(A)^{\dagger}$;
4. $A^{\dagger} = (A^{*}A)^{\dagger}A^{*} = A^{*}(AA^{*})^{\dagger}$

Casi speciali

Se le **colonne** di A sono linearmente indipendenti, allora $A^{*}A$ è invertibile. In questo caso, la formula esplicita è:

$$A^{\dagger} = (A^{*}A)^{-1}A^{*} \tag{2.22}$$

Segue che $A^{\dagger}$ è l'inversa sinistra di A: $A^{\dagger}A = I$. Se le **righe** di A sono linearmente indipendenti, allora AA^{*} è invertibile. In questo caso, la formula esplicita è:

$$A^{\dagger} = A^{*}(AA^{*})^{-1} \tag{2.23}$$

Segue che $A^{\dagger}$ è l'inversa destra di A: $AA^{\dagger} = I$.

Trovare la pseudoinversa di una matrice

Il modo computazionalmente più semplice per ottenere la pseudoinversa è usare la SVD. Se $A = U\Sigma V^{*}$, allora $A^{\dagger} = V\Sigma^{\dagger}U^{*}$. Per una matrice diagonale come Σ, la pseudoinversa si ottene prendendo il recoproco di ogni elemento non nullo sulla diagonale.
Esistono approcci ottimizzati per il calcolo della pseudoinversa di matrici strutturate a blocchi.

Applicazioni

La pseudoinversa fornisce la soluzione ai minimi quadrati per un sistema di equazioni lineari. Dato un sistema sovradeterminato[6] con colonne indipendenti (2):

$$A\mathbf{x} = b \tag{2.24}$$

Si cerca un vettore $\mathbf{x}$:

[6] Ci sono più equazioni (più righe) che incognite (che colonne).

$$\mathbf{x} = \operatorname{argmin}(\|A\mathbf{x} - b\|^2) \tag{2.25}$$

Dove $\|\cdot\|$ è la norma euclidea. La soluzione generale del sistema non omogeneo (2.24) è la somma di una soluzione particolare del sistema (2.24) e del sistema omogeneo $A\mathbf{x} = 0$.

Lemma. *Se* $(AA^*)^{-1}$ *esiste, allora la soluzione* $\mathbf{x}$ *può sempre essere scritta come somma della soluzione pseudoinversa di un sistema non omogeneo e la soluzione di un sistema omogeneo*

$$\mathbf{x} = A^*(AA^*)^{-1}\mathbf{b} + (1 - A^*(AA^*)^{-1}A)\mathbf{y} \tag{2.26}$$

Dim. Sia:

$$A\mathbf{x} = AA^*(AA^*)^{-1}\mathbf{b} + (A\mathbf{y} - AA^*(AA^*)^{-1}A\mathbf{y}) = \\ = \mathbf{b} + (A\mathbf{y} - A\mathbf{y}) = \mathbf{b} \tag{2.27}$$

(**CVD**).
Qua il vettore $\mathbf{y}$ è arbitrario. Inoltre la (2.26) può essere anche scritta come:

$$\mathbf{x} = A^{\dagger}\mathbf{b} + \left(1 - A^{\dagger}A\right)\mathbf{y} \tag{2.28}$$

$A^{\dagger}\mathbf{b}$ è la soluzione pseudoinversa. Nel senso dell'errore dei minimi quadrati, si tratta della migliore approssimazione della soluzione vera. Questo significa che l'elemento di correzione $\left(1 - A^{\dagger}A\right)\mathbf{y}$ ha la norma euclidea minima. Questo termine rappresenta la soluzione del sistema omogeneo $A\mathbf{x} = 0$, perché $\left(1 - A^{\dagger}A\right)$ è la proiezione sul *null space* di A, mentre $A^{\dagger}A = A^*(AA^*)^{-1}A$ è la proiezione nell'immagine di A (lo spazio generato dai vettori colonna di A).

2.4.2. *null space* e rango

Un'altra applicazione della SVD è che fornisce una rappresentazione esplicita dell'*immagine* e del *null space* (null space) di una matrice A. I vettori singolari di destra corrispondenti a valori singolari nulli di A generano il *null space* di A. I vettori singolari di destra corrispondenti ai valori singolari non nulli di A generano l'immagine di A. Di conseguenza, il rango di A coincide con il numero di valori non singolari di A. Inoltre, i ranghi di A, A^*A, AA^* devono essere gli stessi. A^*A ed AA^* hanno gli stessi autovalori non nulli.
Nell'algebra lineare numerica i valori singolari possono essere usati per determinare il *rango effettivo* di una matrice, quando l'errore di arrotondamento conduce a valori singolari piccoli ma non nulli in una matrice *rank deficient.*

2.4.3. Altri esempi

La SVD è applicata ampiamente nello studio di problemi inversi, ed è utile nell'analisi di metodi di regolarizzazione come quello di Tikhonov.

2.5. GSVD

La GSVD della coppia di matrici (A, L) è una generalizzazione della SVD di A nel senso che i valori singolari generalizzati sono le radici quadrate degli autovalori generalizzati delle coppie (A^*A, L^*L).
Per semplificare spostiamoci nel dominio di $\mathbb{R}$ e assumiamo che $A \in \mathbb{R}^{m\times n}$ e che $L \in \mathbb{R}^{p\times n}$ soddisfino a $m \geq n \geq p$ che è sempre il caso connesso ai problemi discreti mal posti. Allora la GSVD è una decomposizione di A ed L nella forma:

$$A = U\begin{bmatrix}\Sigma & 0\\ 0 & I_{n-p}\end{bmatrix}X^{-1}$$
$$L = V[M \quad 0]X^{-1} \tag{2.29}$$

Dove le colonne di $U \in \mathbb{R}^{m\times n}$ e $V \in \mathbb{R}^{p\times p}$ sono ortonormali, $X \in \mathbb{R}^{n\times n}$ è non singolare, Σ ed M sono diagonali $p \times p$: $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_p)$, $M = \mathrm{diag}(\mu_1, \dots, \mu_p)$. Inoltre gli elementi delle matrici diagonali sono non negativi in modo che:

$$0 \le \sigma_1 \le \cdots \le \sigma_p \le 1$$
$$1 \ge \mu_1 \ge \cdots \ge \mu_p > 0 \tag{2.30}$$

E sono normalizzati in modo che:

$$\sigma_i^2 + \mu_i^2 = 1 \quad i = 1, \dots, p \tag{2.31}$$

I *valori singolari generalizzati* γ_i di (A, L) sono definiti dai rapporti

$$\gamma_i = \frac{\sigma_i}{\mu_i} \quad i = 1, \dots, p \tag{2.32}$$

E sono disposti in ordine non decrescente (un ordinamento contrario a quello della SVD).
Nella GSVD le ultime $n - p$ colonne $\mathbf{x}_i$ della matrice X soddisfano a:

$$L\mathbf{x}_i = \mathbf{0}, \quad i = p + 1, \dots, n \tag{2.33}$$

E sono quindi una base per il *null space* $\mathcal{N}(L)$. Si noti che le matrici U, Σ, V della GSVD sono **diverse** rispetto alle matrici con gli stessi simboli usate nella SVD di A. Quando L è la matrice identità, allora U, V della GSVD coincidono con quelle della SVD, e i valori singolari generalizzati coincidono con quelli di A ad eccezione dell'ordinamento.
In generale, non c'è legamene tra valori/vettori singolare generalizzati e ordinari.

3. Algoritmi per il calcolo della SVD

3.1. Introduzione

Nella prima parte di questo capitolo è descritto l'algoritmo per la fattorizzazione QR di matrici in forma di Hessemberg, che verrà usato nell'algoritmo 5 nell'ambito della cosiddetta *fast SVD* per il calcolo degli autovalori di A^TA ovvero AA^T. Seguono gli altri algoritmi implementati per il calcolo della SVD.

3.2. Algoritmi QR[7]

In questa sezione è discussa una tecnica iterativa per l'approssimazione *simultanea* di *tutti* gli autovalori di una matrice A. L'idea di base consiste nel ridurre A tramite opportune trasformazioni di similitudine, in un forma in cui il calcolo degli autovalori è più semplice rispetto al problema di partenza.
L'uso della decomposizione di Schur[8] non è fattibile per $n \geq 5$ (1) e il problema deve essere risolto usando tecniche iterative. Segue il metodo dell'*iterazione* QR mostrato per matrici reali e quadrate.
Sia $A \in \mathbb{R}^{n\times n}$, $Q^{(0)} \in \mathbb{R}^{n\times n}$ ortogonale, $T^{(0)} = \left(Q^{(0)}\right)^T AQ^{(0)}$, per $k = 1,2,\dots$ fino alla convergenza, l'iterazione QR consiste nei seguenti passi:

Determina $Q^{(k)}, R^{(k)}$ tali che:
a. $T^{(k-1)} = Q^{(k)}R^{(k)}$
b. $T^{(k)} = R^{(k)}Q^{(k)}$ (3.1)

Si ottiene:

$$T^{(k)} = \left(Q^{(k)}\right)^T\left(Q^{(k)}R^{(k)}\right)Q^{(k)} = \left(Q^{(k)}\right)^T\left(T^{(k-1)}\right)Q^{(k)} =$$
$$= \left(\prod_{k=0}^{k}\left(Q^{(k)}\right)\right)^T A\left(\prod_{k=0}^{k}\left(Q^{(k)}\right)\right)^{9} \quad (3.2)$$

Dove ogni $T^{(k)}$è ortogonalmente simile ad A. Questo è importante per la stabilità del metodo perché il condizionamento del problema agli autovalori per $T^{(k)}$non è peggio rispetto a quello per A (2).
Se A ha autovalori reali, distinti in modulo, allora $T^{(k)}$è triangolare superiore (con gli autovalori nella diagonale principale). Tuttavia se A è complessa tale limite non può essere una matrice triangolare.
Si noti che la matrice A^TA è hermitiana e semidefinita positiva, quindi per il teorema dello spettro la sua decomposizione QR di A^TA[10] conduce sempre ad una matrice triangolare.

3.3. Metodo QR per matrici in forma di Hessemberg

L'idea consiste nell'iniziare l'iterazione da una matrice $T^{(0)}$ in forma di *Hessemberg superiore,* cioè $t_{ij}^{(0)} = 0$ per $i > j + 1$. Si può mostrare che con questa scelta il calcolo di $T^{(k)}$richiede $O(n^2)$ flops per iterazione.

---

[7] Questo capitolo fa riferimento principalmente a (1).
[8] Trovare U unitaria tale che $T = U^*AU$, con T triangolare superiore, quindi contenente gli autovalori nella diagonale.
[9] $Q^{(0)}Q^{(1)}$ …
[10] O della sua trasposta.

Per raggiungere la massima efficienza e stabilità dell'algoritmo, si impiegano opportune *trasformazioni matriciali.* La riduzione della A in *Hessemberg superiore* è realizzata tramite le matrici di Householder, mentre la fattorizzazione QR di $T^{(k)}$ è ottenuta usando le matrici di Givens.

3.3.1. Matrici di trasformazione di Givens e di Householder

Householder

Per ogni vettore $\mathbf{v} \in \mathbb{R}^n$, è possibile costruire la seguente matrice ortogonale e simmetrica:

$$P = I - 2\mathbf{v}\mathbf{v}^T/\|\mathbf{v}\|_2^2 \tag{3.3}$$

Se $\mathbf{x} \in \mathbb{R}^n$ il vettore $\mathbf{y} = P\mathbf{x}$ è la riflessione di $\mathbf{x}$ rispetto all'iperpiano formato dall'insieme dei vettori ortogonali a $\mathbf{v}$**.** La matrice P e il vettore $\mathbf{v}$ sono detti *matrice di riflessione di Householder* e *vettore di Householder.*

Le matrici di Householder possono essere usate per *azzerare* un blocco di elementi di un vettore $\mathbf{x} \in \mathbb{R}^n$. Se si vogliono azzerare tutti gli elementi di $\mathbf{x}$ tranne l'*m-esimo,* deve essere (2):

$$\mathbf{v} = \mathbf{x} \pm \|\mathbf{x}\|_2 \mathbf{e}_m \tag{3.4}$$

$\mathbf{e}_m$ è l'*m-esimo* vettore della base canonica. Inoltre:

$$\begin{aligned} P\mathbf{x} = \left(I - \frac{2\mathbf{v}\mathbf{v}^T}{\|\mathbf{v}\|_2^2}\right)\mathbf{x} = \mathbf{x} - \frac{2\mathbf{v}\mathbf{v}^T\mathbf{x}}{\|\mathbf{v}\|_2^2} = \mathbf{x} - \frac{2\mathbf{v}^T\mathbf{x}\mathbf{v}^T}{\|\mathbf{v}\|_2^2} \\ = (0, \dots, 0, \pm\|\mathbf{x}\|_2, 0, \dots, 0)^T \end{aligned} \tag{3.5}$$

Se per un certo $k \geq 1$ le prime *k* componenti di $\mathbf{x}$ devono rimanere inalterate, mentre le compoenenti dalla *k+2* devono essere poste a zero, la matrice di Householder $P = P_{(k)}$ assume la forma seguente:

$$P_{(k)} = \begin{bmatrix} I_k & 0 \\ 0 & R_{n-k} \end{bmatrix} \quad R_{n-k} = I_{n-k} - \frac{2\mathbf{w}^{(k)}\left(\mathbf{w}^{(k)}\right)^T}{\|\mathbf{w}^{(k)}\|_2^2} \tag{3.6}$$

Dove R_{n-k} è detta *matrice elementare di Householder* di ordine *n-k* associata alle riflessioni lungo l'iperpiano ortogonale al vettore $\mathbf{w}^{(k)}\mathbb{R}^{n-k}$. Dalla (3.4) si ricava:

$$\mathbf{w}^{(k)} = \mathbf{x}^{(n-k)} \pm \left\|\mathbf{x}^{(n-k)}\right\|_2 \mathbf{e}_1^{(n-k)} \tag{3.7}$$

Le componenti del vettore trasformato $\mathbf{y} = P_{(k)}\mathbf{x}$ sono:

$$\begin{cases} y_j = x_j & j = 1, \dots, k \\ y_j = 0 & j = k+2, \dots, n \\ y_{k+1} = \pm\left\|\mathbf{x}^{n-k}\right\|_2 & j = k+1 \end{cases} \tag{3.8}$$

Questo è il primo passo per una efficiente implementazione dell'iterazione QR con $T^{(0)} = H^{(0)}$.

Givens

Le *matrici elementari di Givens* sono matrici di rotazione ortogonale che consentono di porre a zero in modo selettivo gli elementi di un vettore o una matrice. Per una coppia di indici *i* e *k,* e un dato angolo θ, queste matrici sono definite come:

$$G(i, k, \theta) = I_n - Y \tag{3.9}$$

Dove $Y \in \mathbb{R}^{n\times n}$ è una matrice nulla tranne che per un blocco quadrato costituito dalla matrice elementare di rotazione: $y_{ii} = y_{kk} = 1 - \cos(\theta)$ e $y_{ik} = -\sin(\theta) = -y_{ki}$.
Per un certo vettore $\mathbf{x} \in \mathbb{R}^n$, il prodotto $\mathbf{y} = G(i,k,\theta)^T\mathbf{x}$ equivale ad una rotazione antioraria di un angolo θ nelle coordinate piane (x_i, x_k). Posto $c = \cos(\theta)$ e $s = \sin(\theta)$, segue (confrontare con (2)):

$$\begin{cases} y_j = x_j & j \neq i,k \\ y_j = cx_i - sx_k & j = i \\ y_j = sx_i + cx_k & j = k \end{cases} \tag{3.10}$$

Scegliendo opportunamente i parametri c, s si può annullare y_k oppure y_i.(2).

3.3.2. Riduzione di una matrice in forma di Hessemberg

Una matrice $A \in \mathbb{R}^{n\times n}$ può essere trasformata mediante trasformazioni di similitudine in una matrice di *Hessemberg superiore* con un costo di $O(n^3)$ flops. L'algoritmo è completato in *n-2* passi e la trasformazione di similitudine Q può essere calcolata come prodotto delle matrici $P_{(1)}, \dots, P_{(2)}$. Per questo, la procedura di riduzione di Householder è comunemente nota come *metodo di Householder.*
I *k-passi* costituiscono delle trasformazioni di similitudine di A attraverso la matrice di Householder $P_{(k)}$ il cui obiettivo è quello di porre a zero gli elementi in posizione $k+2, \dots, n$ della *k-esima* colonna di A, per $k = 1, \dots, (n-2)$. Per esempio, nel caso $n = 4$ il processo di riduzione si concluderebbe in 2 passi:

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

Posto $A^{(0)} = A$ il metodo genera una sequenza di matrici $A^{(k)}$ ortogonalmente simili ad A:

$$A^{(k)} = P_{(k)}^T A^{(k-1)} P_{(k)} = \left(P_{(k)} \dots P_{(1)}\right)^T A \left(P_{(k)} \dots P_{(1)}\right) = Q_{(k)}^T A Q_{(k)} \quad k \geq 1 \tag{3.11}$$

Per ogni $k \geq 1$ la matrice $P_{(k)}$ è data dalla (3.6) sostituendo la $\mathbf{x}$ con la *k-esima* sotto-colonna (con componenti dalla $k+1$-esima alla n-esima) della matrice $A^{(k-1)}$. Dalla (3.6) è facile osservare che l'operazione $P_{(k)}^T A^{(k-1)}$ lascia le prime *k* righe di $P_{(k)}$ invariate, mentre $P_{(k)}^T A^{(k-1)} P_{(k)}$ fa la stessa cosa con le prime *k* colonne. Dopo *n-2* passi della riduzione di Householder, si ottiene una matrice $H = A^{(n-2)}$ in forma di Hessemberg superiore.

Osservazione (Caso Simmetrico). Se A è simmetrica, la trasformazione (3.11) mantiene la simmetria. Infatti:

$$\left(A^{(k)}\right)^T = \left(Q_{(k)}^T A Q_{(k)}\right)^T = A^{(k)} \quad \forall k \geq 1 \tag{3.12}$$

In questo modo H **deve essere tridiagonale.** I suoi autovalori possono essere calcolati in modo efficiente tramite il *metodo delle sequenze di Sturm* (1) con un costo di $O(n)$ flops.
Una codifica del metodo di riduzione di Householder è fornita sotto.

houshess : Hessemberg-Householder method (1)

```
function [H,Q] = houshess(A)
n = max(size(A));
Q = eye(n) ;
H = A ;
for k = 1:(n-2)
    [v, beta] = vhouse(H(k+1:n,k));
```

```
    I          = eye(k) ;
    N          = zeros(k,n-k);
    m          = length(v);          % n-k
    R          = eye(m)-beta*v*v' ;  % matrice elementare di Householder
    H(k+1:n,k:n) = R*H(k+1:n,k:n);   % applicata alle righe
    H(1:n,k+1:n) = H(1:n,k+1:n)*R;   % applicata alle colonne
    P = [ I N ; N' R] ;              % matrice di Householder al passo k+1
    Q = Q*P ;
end
```

L'algoritmo **houshess** richiede $O\left(\frac{10n^3}{3}\right)$ flop ed è ben condizionato rispetto agli errori di arrotondamento. Vale la seguente stima:

$$\widehat{H} = Q^T(A+E)Q \quad \|E\|_F \leq cn^2\mathrm{u}\|A\|_F \tag{3.13}$$

Dove u è l'unità di *arrotondamento* (`eps/2`) e $\|\ \|_F$ è la norma di Frobenius. L'accuratezza di una procedura di trasformazione può essere misurata calcolando:

$$\|H - Q^T H_k Q\|_F \tag{3.14}$$

3.3.3. Fattorizzazione QR usando Householder

Rispetto all'algoritmo **houshess** sono sufficienti poche modifiche, evidenziate in rosso. Chiaramente l'applicazione di `R` alle colonne deve essere eliminata. Maggiori dettagli si trovano in (2).

housholder : Fattorizzazione QR di HouseHolder (1)

```
function [Q,H] = housholder(A)     % A = Q*H
n = max(size(A));
Q = eye(n) ;
H = A ;
for k = 1:(n-1)
    [v, beta] = vhouse(H(k:n,k));
    I          = eye(k-1) ;
    m          = length(v);          % n-k
    N          = zeros(k-1,m);
    R          = eye(m)-beta*v*v' ;  % matrice elementare di Householder
    H(k:n,k:n) = R*H(k:n,k:n);       % applicata alle righe
    P = [ I N ; N' R] ;              % matrice di Householder al passo k+1
    Q = Q*P ;
end
```

3.3.4. Algoritmo QR per matrici in forma di Hessemberg

Mostriamo ora come implementare in modo efficiente il passo generico di una iterazione QR, partendo da una matrice $T^{(0)} = H(0)$ in forma di Hessemberg superiore. Per ogni $k \geq 1$ la prima fase consiste nel calcolo della fattorizzazione QR di $H^{(k-1)}$ tramite $n-1$ rotazioni di Givens:

$$\left(Q^{(k)}\right)^T H^{(k-1)} = \left(G_{n-1}^{(k)}\right)^T \dots \left(G_1^{(k)}\right)^T H^{(k-1)} = R^{(k)} \tag{3.15}$$

Dove, per ogni $j = 1, \dots, n-1, G_j^{(k)} = G\left(j, j+1, \theta_j\right)^{(k)}$ è, per ogni $k \geq 1$, la *j-esima* matrice di rotazione di Givens nelle quali gli θ_j sono scelti in modo che gli elementi $(j+1, j)$ della $\left(G_j^{(k)}\right)^T \dots \left(G_1^{(k)}\right)^T H^{(k-1)}$ siano posti a zero. Il prodotto (3.15) ha un costo di $O(3n^2)$ flops.

Il passo successivo consiste nel completare la trasformazione ortogonale di similitudine:

$$H^{(k)} = R^{(k)}Q^{(k)} = R^{(k)}\left(G_1^{(k)} \dots G_{n-1}^{(k)}\right) \tag{3.16}$$

La matrice ortogonale $Q^{(k)}$ è una matrice di *Hessemberg superiore.* Quindi, per esempio, per $n = 3$:

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

Anche la (3.16) richiede $O(3n^2)$ flops, quindi in tutto sono $O(6n^2)$. In conclusione la fattorizzazione QR con matrici elementari di Givens su una matrice iniziale di Hessemberg conduce ad una riduzione delle operazioni di **un ordine di grandezza** rispetto alla corrispondente fattorizzazione di Gram-Schmidt (1).

3.3.5. Iterazione di base QR a partire dalla forma di Hessemberg superiore

L'implementazione dell'iterazione di base QR genera la decomposizione reale di Schur di una matrice A.
Il programma che segue usa **`houshess`** per ridurre A in forma di Hessemberg superiore; quindi ogni passo della fattorizzazione QR (3.1) è condotto usando le rotazioni di Givens. L'efficienza dell'algoritmo è assicurata da una pre- e post- moltiplicazione con delle matrici di Givens come spiegato in seguito, e tramite la costruzione di matrici $Q^{(k)} = \left(G_{(1)}^{(k)}\right) \dots \left(G_{(n-1)}^{(k)}\right)$ (**`prodgiv`**) con un costo di $O(n^2 - 2)$ flop e senza effettuare un moltiplicazione esplicita (2).
Come già discusso per la stabilità della iterazione QR, rispetto alla propagazione dell'errore di arrotondamento, si può mostrare che la forma di Schur $\hat{T}$ calcolata è ortogonalmente simile ad una matrice 'vicina' ad A cioè:

$$\hat{T} = Q^T(A + E)Q \quad \|E\|_2 \leq \mathrm{u}\|A\|_2 \tag{3.17}$$

Il programma che segue, dopo `niter` iterazioni della procedura QR fornisce le matrici T, Q, R. Le ipotesi per la convergenza del metodo sono menzionate in (2).

`hessqr` : Metodo Hessemberg-QR (1)

```
function [T,Q,R]=hessqr(A,niter)
n=max(size(A)); %
[T,Qhess]=houshess(A);
for j=1:niter
    [Q,R,c,s]= qrgivens(T);
    T=R;
    for k=1:n-1,
    T=gacol(T,c(k),s(k),1,k+1,k,k+1);
    end
end
```

`qrgivens` : Fattorizzazione QR con rotazioni di Givens (1)

```
function [Q,R,c,s]= qrgivens(H)
[m,n]=size(H); %
for k=1:n-1
    [c(k),s(k)]=givcos(H(k,k),H(k+1,k));
    H=garow(H,c(k),s(k),k,k+1,k,n);
end
    R=H;
    Q=prodgiv(c,s,n);
```

prodgiv : Costruzione della matrice $Q^{(k)}$

```
function Q=prodgiv(c,s,n)
n1=n-1; %
n2=n-2;
Q=eye(n);

Q(n1,n1)=c(n1);
Q(n,n)=c(n1);
Q(n1,n)=s(n1);
Q(n,n1)=-s(n1);

for k=n2:-1:1,
k1=k+1;
Q(k,k)=c(k);
Q(k1,k)=-s(k);
q=Q(k1,k1:n);
Q(k,k1:n)=s(k)*q;
Q(k1,k1:n)=c(k)*q;
end
```

3.3.6. Implementazione delle matrici di trasformazione

Nella (3.7) conviene scegliere il segno meno, ottenendo $\mathbf{w}^{(k)} = \mathbf{x}^{(n-k)} - \left\|\mathbf{x}^{(n-k)}\right\|_2 \mathbf{e}_1^{(n-k)}$ [11] in modo che $R_{n-k}\mathbf{x}^{(n-k)}$ sia un multiplo positivo di $\mathbf{e}_1^{(n-k)}$. Se x_{k+1} è positivo, per evitare cancellazioni numeriche, il calcolo deve essere razionalizzato come segue:

$$w_1^{(k)} = \frac{\left(x_{k+1}^2 - \left\|\mathbf{x}^{(n-k)}\right\|_2^2\right)}{\left(x_{k+1} + \left\|\mathbf{x}^{(n-k)}\right\|_2\right)} = \frac{-\sum_{j=(k+2)}^{n} x_j^2}{\left(x_{k+1} + \left\|\mathbf{x}^{(n-k)}\right\|_2\right)} \tag{3.18}$$

La costruzione del vettore di Householder è realizzata da **vhouse** che prende in ingresso un vettore $\mathbf{p} \in \mathbb{R}^{n-k}$ (quello che prima era il vettore $\mathbf{x}^{(n-k)}$) e ritorna un vettore $\mathbf{q} \in \mathbb{R}^{n-k}$ (il vettore $\mathbf{w}^{(k)}$ di Householder), con un costo di $O(n)$ flop.

Se $M \in \mathbb{R}^{m\times m}$ è la generica matrice alla quale è applicata la matrice P di Householder (dove I è la matrice identità di ordine m e $\mathbf{v} \in \mathbb{R}^m$) mettendo $\mathbf{w} = M^T\mathbf{v}$ allora

$$PM = M - \beta\mathbf{v}\mathbf{w}^T \quad \beta = 2/\|\mathbf{v}\|_2^2 \tag{3.19}$$

Il prodotto PM è costituito da un prodotto matrice per vettore ($\mathbf{w} = M^T\mathbf{v}$) più un prodotto esterno vettore per vettore ($\mathbf{v}\mathbf{w}^T$). L'intero costo computazionale del prodotto PM è pari a $O\big(2(m^2+m)\big)$. Simili considerazioni possono essere fatte nel caso del prodotto MP. Definendo $\mathbf{w} = M^T\mathbf{v}$ otteniamo:

$$MP = M - \beta\mathbf{w}\mathbf{v}^T \tag{3.20}$$

Si noti che queste ultime due formule non richiedono esplicitamente la costruzione della matrice P. Questo riduce il costo computazionale a $O(m^2)$ flops, mentre il calcolo del prodotto PM senza tener conto della struttura speciale di P necessiterebbe $O(m^3)$ flops.

---

[11] Citazione da (1):

In the definition (5.42) it is convenient to choose the minus sign, obtaining $\mathrm{w}^{(k)} = \mathrm{x}^{(n-k)} - \|\mathrm{x}^{(n-k)}\|_2 \mathrm{e}_1^{(n-k)}$, in such a way that the vector $\mathrm{R}_{n-k}\mathrm{x}^{(n-k)}$ is a positive multiple of $\mathrm{e}_1^{(n-k)}$. If x_{k+1} is positive, in order to avoid numerical cancellations, the computation can be rationalized as follows

vhouse : Costruzione del vettore di Householder (1)

```
function [v,beta]=vhouse(x)
n=length(x);
x=x/norm(x);
s=x(2:n)'*x(2:n);
v=[1; x(2:n)];

if (s==0)
    beta=0; %
else
    mu=sqrt(x(1)^2+s);
if (x(1) <= 0)
    v(1)=x(1)-mu;
else
    v(1)=-s/(x(1)+mu);
end
beta=2*v(1)^2/(s+v(1)^2);
v=v/v(1);
end
```

Relativamente alle matrici di rotazione di Givens, per il calcolo di c ed s, si considerino i e k e si assuma che la *k-esima* componente di un dato vettore $\mathbf{x} \in \mathbb{R}^n$ debba essere posta a zero. Essendo $r = \sqrt{x_i^2 + x_k^2}$ si ha:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{3.21}$$

L'esecuzione del programma **givcos** richiede solo 5 flops, più la valutazione della radice quadrata. Come notato nel caso della matrice di Householder, anche per le rotazioni di Givens non abbiamo bisogno di calcolare esplicitamente la matrice $G(i,k,\theta)$ per ottenere il prodotto con una data matrice $M \in \mathbb{R}^{m\times m}$. Per questo obiettivo si usano i due programmi **garow** e **gacol** entrambi al costo di $O(6m)$ flops. Osservando la struttura della matrice $G(i,k,\theta)$ è chiaro che il primo algoritmo modifica solo le righe *i* e *k* di M, mentre il secondo cambia solo le colonne *i* e *k* di M.
Concludiamo notando che il calcolo del vettore di Householder $\mathbf{v}$ e di (c,s) sono operazioni *ben condizionate* rispetto agli errori di arrotondamento. La soluzione del sistema (3.21) è implementata in **givcos**.

givcos : Calcolo del seno e coseno di Givens (1)

```
function [c,s]=givcos(xi, xk)
if (xk==0)
    c=1;
    s=0;
else
if abs(xk) > abs(xi)
    t=-xi/xk;
    s=1/sqrt(1+t^2);
    c=s*t;
else
    t=-xk/xi;
    c=1/sqrt(1+t^2);
    s=c*t;
end
end
```

I programmi **garow** e **gacol** calcolano $G(i,k,\theta)^T M$ e $MG(i,k,\theta)$rispettivamente. I parametri di ingresso sono (c,s). In **garow** gli indici *i* e *k* identificano le righe della matrice M che sono soggette

all'aggiornamento $M \leftarrow G(i,k,\theta)^T M$, mentre *j1* e *j2* sono gli indici delle colonne coinvolte nel calcolo. Similmente in **gacol** *i* e *k* identificano le colonne soggette all'aggiornamento $M \leftarrow MG(i,k,\theta)$, mentre *j1* e *j2* sono gli indici delle righe coinvolte nel calcolo.

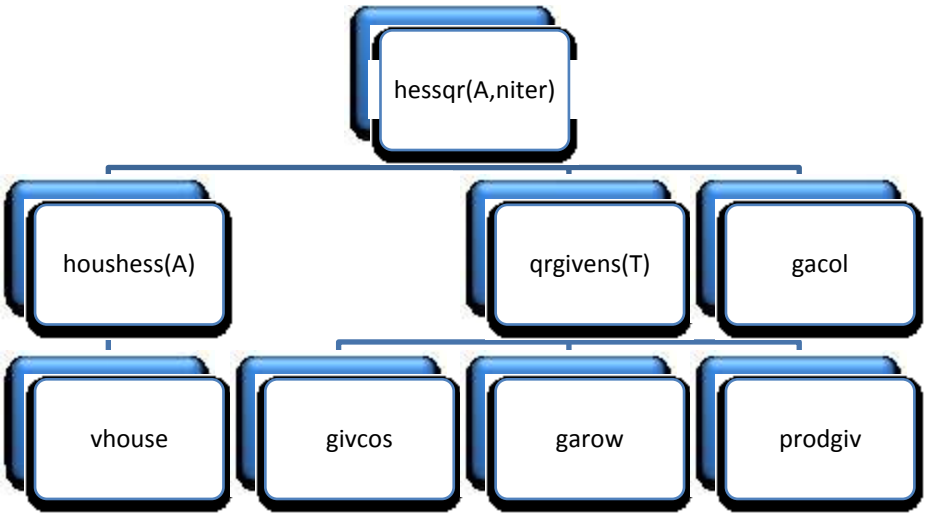
garow : Prodotto $G(i,k,\theta)^T M$ (1)

```
function [M]=garow(M,c,s,i,k,j1,j2)

for j=j1:j2
    t1=M(i,j);
    t2=M(k,j);
    M(i,j)=c*t1-s*t2;
    M(k,j)=s*t1+c*t2;
end
```

gacol : Prodotto $MG(i,k,\theta)$

```
function [M]=gacol(M,c,s,j1,j2,i,k)
for j=j1:j2
    t1=M(j,i);
    t2=M(j,k);
    M(j,i)=c*t1-s*t2;
    M(j,k)=s*t1+c*t2;
end
```



3-1 - Schema delle funzioni usate

3.4. Calcolo della SVD

Segue l'esposizione degli algoritmi implementati per il calcolo della SVD.

3.4.1. SVD_ver1 – QR con Householder – id.1

Si consideri la solita espressione della SVD:

$$A = U\Sigma V^* \tag{3.22}$$

L'idea è quella di usare la decomposizione QR per ottenere U sulla sinistra e di nuovo la QR per ottenere V^* sulla destra. Questo procedimento rende A triangolare inferiore e quindi triangolare superiore in modo alternativo. Alla fine A diventa sia triangolare superiore che triangolare inferiore, cioè diagonale con i valori singolari sulla diagonale. Maggiori dettagli sono forniti nel paragrafo successivo. Al primo passo si avrà:

$$A_{0,1} = IAI = U_0\Sigma_0 V_0^* = U_0 Q_{0,1} R_{0,1} V_0^* = U_1 R_{0,1} V_0^* = U_1 \Sigma_{0,1} V_0^* \tag{3.23}$$

$$A_1 = A_{0,2} = U_1\Sigma_{0,1}V_0^* = U_1 Q_{0,2} R_{0,2} V_0^* = U_1 \Sigma_{0,2}^* Q_{0,2}^* V_0^* = U_1 \Sigma_1 R_{0,2}^* V_1^* = U_1 \Sigma_1 V_1^* Q_{0,2} \tag{3.24}$$

La matrice $\Sigma_{0,1}$ ha dimensioni $m \times n$, mentre $R_{0,1,1}$ è una $n \times n$:

$$\Sigma_{0,1} = R_{0,1} = \begin{bmatrix} R_{0,1,1} \\ 0 \end{bmatrix}$$

Chiaramente Σ_1 ha ancora dimensioni $m \times n$. Il criterio di stop è basato sulle due condizioni:

$$k \geq N \tag{3.25}$$

$$e_k = \frac{\|T(\Sigma_k)\|}{\|D(\Sigma_k)\|} < \tau \tag{3.26}$$

Dove $T(\Sigma_k) = \Sigma_k - D(\Sigma_k)$ è la matrice $m \times n$ contenente la parte triangolare superiore di Σ_k che tende a zero per $k \to \infty$, mentre $D(\Sigma_k)$ è la matrice $m \times n$ contenente la diagonale di Σ_k che per $k \to \infty$ tende a Σ che nella diagonale contiene i valori singolari. Segue il programma per il calcolo della SVD (una versione modificata di (3))

svd_v1 : Fornisce la SVD mediante fattorizzazione QR

```
function [U,S,V] = svd_v1(X,tol)
 if ~exist('tol','var')
   tol=eps*1024; %Fisso una tolleranza sull'errore se non è fornita
end

[M N]=size(X);
loopmax=100*max(size(X));
loopcount=0;

U=eye(M); % A = IAI
S=X';
V=eye(N);

Err=realmax;
while (Err>tol && loopcount<loopmax)
    [Q,S]=qr(S'); U=U*Q;
    [Q,S]=qr(S'); V=V*Q;
    e=triu(S,1);
    E=norm(e(:));
    F=norm(diag(S));
    Err=E/F;
    loopcount=loopcount+1;
end
%Stabilisce i segni in S
SS=diag(S);
S=zeros(M,N);
for n=1:length(SS)
    SSN=SS(n);
    S(n,n)=abs(SSN);
```

```
    if SSN<0
       U(:,n)=-U(:,n);
    end
end
```

3.4.2. SVD_ver1 – QR di MATLAB– id.2

Rispetto all'algoritmo precedente usa la fattorizzazione QR di MATLAB

svd_v1_matlab : Fornisce la SVD mediante fattorizzazione QR

```
function [U,S,V] = svd_v1_matlab(X,tol)
…
while (Err>tol && loopcount<loopmax)
    [Q,S]=qr(S'); U=U*Q;
    [Q,S]=qr(S'); V=V*Q;
…
```

3.4.3. SVD_ver2 – algoritmo di Golub-Kahan-Reinsch - id. 3

L'algoritmo di Golub-Kahan-Reinsch per il calcolo della SVD di una matrice $A \in \mathbb{R}^{m\times n}$ con $m \geq n$ consiste di due fasi, una diretta e una iterativa. La parte iterativa coincide con SVD_ver1. Nella prima fase A è trasformata in una matrice trapezioidale superiore della forma:

$$\mathcal{U}^T A \mathcal{V} = \begin{pmatrix} B \\ 0 \end{pmatrix} \tag{3.27}$$

Dove $\mathcal{U}^T, \mathcal{V}$ sono due matrici ortogonali e $B \in \mathbb{R}^{n\times n}$ è bidiagonale superiore. Le matrici $\mathcal{U}, \mathcal{V}$ sono generate usando $n + m - 3$ matrici di Householder come mostrato in (1). All'*(m-1)-esimo* passo si avrà:

$$\mathcal{U} = \mathcal{U}_1\mathcal{U}_1 \dots \mathcal{U}_{m-1} \quad \mathcal{V} = \mathcal{V}_1\mathcal{V}_2 \dots \mathcal{V}_{n-2} \tag{3.28}$$

Queste operazioni sono implementate nel programma **bidiag** (4) .

bidiag: Bidiagonalizzazione

```
function [UU,b,VV] = bidiag(a,mode)
if exist('mode','var')
   if mode~='imag'
      mode='real';
   end
else
   mode='real';
end

[r,c]=size(a);
n=min(r,c);
UU=eye(r);
VV=eye(c);

for k=1:n

%zero a col
if k<=r
  x=a(k:end,k);
  v=x;
%what if x is a zero vector or has x(1)=0?
  if x(1)==0,
     x(1)=eps*eps;
  end
```

```
  v(1)=v(1)+sign(x(1))*norm(x);

  U=eye(length(v))-2/(v'*v)*(v*v');
  z=zeros(k-1,r-k+1);
  U=[eye(k-1)  z  ;
       z.'     U];
if mode=='real'
  if ~isreal(v(1))
     phi=atan2(imag(v(1)),real(v(1)));
     U(k,:)=exp(-i*phi)*U(k,:);
  end
end
  UU=U*UU;
  a=U*a;
end

%zero a row
if k<c
  x=a(k,k+1:end);
  v=x.';

%what if x is a zero vector or has x(1)=0?
  if x(1)==0,
     x(1)=eps*eps;
  end
  v(1)=v(1)+sign(x(1))*norm(x);

  V=conj(eye(length(v))-2/(v'*v)*(v*v'));
  z=zeros(k,c-k);
  V=[eye(k)  z  ;
       z.'   V];
if mode=='real'
  if ~isreal(v(1))
     phi=atan2(imag(v(1)),real(v(1)));
     V(:,k+1)=exp(-i*phi)*V(:,k+1);
  end
end
  VV=VV*V;
  a=a*V;
end

end

UU=UU';
if mode=='real'
   a=real(a);
   q=sign(diag(a));
   if r==1,q=q(1:r,1:r);end
   if c==1,q=q(1:c,1:c);end
   q(q>=0)=1;
   q=diag(q);
   if r<=c
      UU=UU*q;
       a=q*a;
   else
      VV=q*VV;
       a=a*q;
   end
end

b=triu(tril((a),1));
```

```
if nargout<=1
   UU=b;
end
```

Nella seconda fase, la matrice B è ridotta in una matrice Σ usando le iterazioni QR. Ad essere precisi, si costruisce una sequenza di matrici bidiagonali superiori B^k in modo che, quando $k \to \infty$, gli elementi che si trovano fuori dalla diagonale tendono a zero quadraticamente e gli elementi della diagonale tendono ai valori singolari σ_i di A. Alla fine il processo genera due matrici ortogonali $\mathcal{W}$ e $\mathcal{Z}$ tali che:

$$\mathcal{W}^T B \mathcal{Z} = \Sigma = diag(\sigma_i, \dots, \sigma_n) \tag{3.29}$$

La SVD di A è quindi data da:

$$U^T A V = \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} \tag{3.30}$$

Con $U = \mathcal{U} diag(\mathcal{W}, I_{m-n})$ ed $V = \mathcal{V}\mathcal{Z}$. Il costo computazionale di questa procedura è di $O\left(2m^2n + 4mn^2 + \frac{9}{2}n^3\right)$ flop, che si riduce a $O(2m^2n - 2n^3)$ flop se si calcolano solo i valori singolari. Questo problema è meno costoso di quello che impiega A^TA per la ricerca della SVD. Per quanto riguarda la stabilità della procedura, si può mostrare che i valori singolari ottenuti sono i valori singolari di $A + \sigma A$ con:

$$\|\delta A\|_2 \leq C_{mn} \mathrm{u} \|A\|_2 \tag{3.31}$$

Dove C_{mn} è una costante che dipende da n, m e da `eps`. L'algoritmo è quindi:

SVD_v2 :

```
function [U,S,V] = svd_v2(X)

[UU,b,VV] = bidiag(X,'imag');
[M N] = size(b);
n = max([M N]);
m = min([M N]);
B = b(1:n,1:n);
[W,S,Z] = svd_v1_matlab(B);
U = UU*W;
V = VV*Z;
```

3.4.4. SVD_ver3 - Fast SVD

L'idea per questo algoritmo è stata dedotta da (5). La principale differenza tra la versione originale della SVD e la fast SVD è che quest'ultima è sostanzialmente più veloce per matrici 'grasse' ($n > m$) e 'magre' ($m > n$). In questo caso la decomposizione di A può essere accelerata prima calcolando la SVD di AA^T ($n > m$) oppure A^TA ($m > n$), piuttosto che su A. La seconda differenza è che la fast SVD ritorna solo i valori singolari positivi (quindi la dimensione di Σ coincide sempre con il rango di A). Si noti che i vettori singolari calcolati con la fast SVD possono differire nel segno da quelli calcolati con la svd. Se la matrice è 'grassa' ($n > m$) allora:

$$AA^T = U\Sigma^2 U^T \tag{3.32}$$

La matrice AA^T ha solo dimensione $n \times n$, quindi è più rapido decomporla rispetto alla decomposizione su A. Essendo $A = U\Sigma V^T$:

$$V = A'U\Sigma^{-1}$$
$$U = AV\Sigma^{-1} \tag{3.33}$$

Si noti che $\Sigma^{-1} = diag\left(\frac{1}{d_1}, \dots, \frac{1}{d_n}\right)$ se $\Sigma = diag(d_1, \dots, d_n)$. Se la matrice è 'magra' ($m > n$) allora:

$$A^T A = V\Sigma^2 V^T \tag{3.34}$$

La matrice $A^T A$ ha solo dimensione $m \times m$, quindi è più rapido decomporla rispetto alla decomposizione su A. Essendo $A = U\Sigma V^T$:

$$U = AV\Sigma^{-1}$$
$$V = A^T U{\Sigma^{-1}}^T \tag{3.35}$$

Id. 5 – QR con Householder

SVD_v3:

```
function [U,S,V] = svd_v3(X)

[M N]=size(X);

if M > N  % la matrice è magra
[V SS V] = svd_v1(X'*X);
S = sqrt(SS) ;
S_inv = S^-1 ;
U = X*V*S_inv ;
V = X'*U*S_inv';
S = [ sqrt(SS) ; zeros(M-N,N)];
else      % la matrice è grassa
[U SS U] = svd_v1(X*X');
S = sqrt(SS) ;
S_inv = S^-1;
S = [ S zeros(M,N-M)];
V = X'*U*S_inv ;
U = X*V*S_inv ;
end
```

Id. 6, 8, 9 – Calcolo degli autovalori di $A^T A$

È una variante dell'algoritmo precedente in cui si calcolano gli autovalori di $A^T A$ oppure AA^T usando l'algoritmo descritto in 3.3.4.

SVD_v3_hess:

```
function [U,S,V] = svd_v3_hess(X,iter)

[M N]=size(X);
if M > N  % la matrice è magra
[T Q R] = hessqr(X'*X,iter);
S = sqrt(diag(T));
else      % la matrice è grassa
[T Q R] = hessqr(X*X',iter);
S = sqrt(diag(T));
end
```

3.4.5. SVD in MATLAB

L'algoritmo `svd` usa le *routine* di LAPACK. La subroutine di LAPACK, DGESVD, rappresenta l'approccio tipico per il calcolo della SVD. Se una matrice ha più righe che colonne occorre una decomposizione QR. Il fattore R è ridotto ad una matrice bidiagonale. I valori singolari desiderati e i vettori si trovano realizzando una iterazione bidiagonale QR, usando la routine di LAPACK, DBDSQR.
La GNU Scientific Library fornisce dei modi alternativi per il calcolo della SVD: attraverso l'algoritmo di Golub-Reinsch, attraverso l'algoritmo di Golub-Reinsch modificato (più veloce per matrici con $m > n$), o tramite l'ortogonalizzazione di Jacobi da un lato. Segue un tabella con i principali usi della funzione `svd`:

`[U,S,V] = svd(X)`	I valori singolari nella diagonale di S sono in ordine decrescente
`[U,S,V] = svd(X,0)`	Produce una decomposizione 'economica': se $m > n$ sono calcolate solo le prime n colonne di U ed S è $n \times n$.
`[U,S,V] = svd(X,'econ')`	Produce una decomposizione 'economica': se $m \geq n$ equivale alla precedente, se $m < n$ sono calcolate solo le prime m colonne di V ed S è $m \times m$.
`s = svd(X)`	Fornisce un vettore contenente i valori singolari.

3.4.6. SVD del REGULARIZATION TOOL

E' calcolata tramite la *routine* `csvd` e al suo interno usa `svd`.

4. Problemi discreti mal posti

In questo capitolo è fornita una breve introduzione ai problemi discreti mal posti. Sono discussi alcuni metodi di regolarizzazione numerica l'uso della SVD nella regolarizzazione, la condizione discreta di Picard, la curva L, necessari per l'analisi di problemi discreti mal posti.

4.1. Problemi discreti mal posti

Un *problema è ben posto* (Hadamard) se:

1. La soluzione esiste;
2. La soluzione è unica;
3. La soluzione dipende con continuità dai dati, in una opportuna topologia.

In campo ingegneristico esistono molti problemi *mal posti.* I problemi continui devono essere discretizzati per ottenere una soluzione numerica. Mentre in termini di analisi funzionale questi problemi sono tipicamente continui, quando risolti con una precisione finita potrebbero soffrire di instabilità numerica o errori nei dati.
Anche se un problema è ben posto, può essere ancora mal condizionato. Se un problema è ben posto, è possibile ottenere una soluzione corretta usando un algoritmo stabile. Se è mal posto è necessario riformularlo prima di trattarlo numericamente. Tipicamente è necessario fare delle assunzioni aggiuntive, come la *smoothness* della soluzione o la limitatezza della sua norma. Questo processo è noto come *regolarizzazione* (6).
Il classico esempio di un problema mal posto è l'equazione integrale di Fredholm del primo tipo con un nucleo integrabile al quadrato (in L^2). Si tratta di un problema dimensionalmente infinito estremamente sensibile alle perturbazioni ad alta frequenza. Alcuni problemi discreti a dimensione finita hanno comportamenti simili e in tale caso si parla di *problemi discreti mal posti.* Si consideri il problema del sistema lineare di equazioni:

$$A\mathbf{x} = \boldsymbol{b}, \qquad A \in \mathbb{R}^{n\times n} \tag{4.1}$$

E il problema ai minimi quadrati:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2 \ , \ \ A \in \mathbb{R}^{m\times n}, \ \ m > n \tag{4.2}$$

Si dice che questi sono problemi discreti mal posti se sono soddisfatte le due condizioni[12]:

1. I valori singolari di A decadono gradualmente a zero;
2. Il rapporto tra il più grande e il più piccolo valore singolare non nullo è grande.

Il secondo criterio implica che la matrice A è mal condizionata, cioè che la soluzione è molto sensibile alle perturbazioni. Il primo indica che 'nelle vicinanze' non ci sono problemi con una matrice dei coefficienti ben condizionata e con un rango numerico ben determinato.

[12] Intuitivamente comprensibili se si fa riferimento agli autovalori.

Tipicamente discretizzando problemi mal posti si ottengono sistemi lineari e problemi lineari ai minimi quadrati mal posti.
Un aspetto interessante dei problemi mal posti è che il cattivo condizionamento del problema non esclude la possibilità di calcolare una soluzione approssimata significativa. Tuttavia, il cattivo condizionamento implica che i metodi standard dell'algebra lineare numerica, come LU, Cholesky, o fattorizzazione QR non possono essere usati direttamente per il calcolo di (4.1)-(4.2) e che bisogna ricorrere a tecniche di regolarizzazione.

4.2. Metodi di regolarizzazione

La difficoltà principale con i problemi discreti mal posti (4.1)-(4.2) è che questi sono essenzialmente indeterminati a causa del *cluster* di piccoli valori singolari di A. Perciò è necessario incorporare ulteriori informazioni o vincoli sulla soluzione desiderata per stabilizzare il problema e per isolare una soluzione utile e stabile. Questo è lo scopo della regolarizzazione.
Tipicamente si richiede che una norma 2 (o un'opportuna seminorma) della soluzione sia piccola. È anche possibile includere una stima della soluzione $\mathbf{x_0}$ nel vincolo. Il vincolo è:

$$\min \Omega(\mathbf{x}) \quad \text{con} \quad \Omega(\mathbf{x}) = \|L(\mathbf{x} - \mathbf{x_0})\|_2 \tag{4.3}$$

La matrice L può essere:

1. Tipicamente la matrice identità I_n;
2. Una approssimazione discreta $p \times n$ dell'operatore di derivazione *(n-p)-esimo,* in questo caso L è una matrice a bande con rango pieno di riga
3. Per le altre espressioni si veda (7).

Tramite il vincolo Ω è possibile controllare la *smoothness* della soluzione regolarizzata. Introdotto il vincolo $\Omega(\mathbf{x})$, occorre abbandonare la richiesta di una soluzione esatta in (4.1), per cercare una soluzione che non sia troppo lontana da quella desiderata (la soluzione sconosciuta del problema imperturbato) e che fornisca un compromesso tra la minimizzazione della (semi)norma (si veda 7.1) $\Omega(\mathbf{x})$ e la minimizzazione della norma del residuo $\|A\mathbf{x} - \mathbf{b}\|_2$. La stessa idea può essere applicata al metodo dei minimi quadrati (4.2).
La più nota forma di regolarizzazione è la *regolarizzazione di Tikhonov.* Si definisce la soluzione regolarizzata $\mathbf{x}_\lambda$ come quella che minimizza la seguente combinazione pesata della norma del residuo e del vincolo:

$$\mathbf{x}_\lambda = \operatorname{argmin}\{\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda^2 \|L(\mathbf{x} - \mathbf{x_0})\|_2^2\} \tag{4.4}$$

Dove il *parametro di regolarizzazione* λ controlla il peso dato alla minimizzazione del vincolo rispetto alla minimizzazione della norma del residuo:

- Un grande λ (una grande quantità di regolarizzazione) favorisce una soluzione con una piccola seminorma al costo di una grande norma del residuo;
- Un piccolo λ (cioè una piccola quantità di regolarizzazione) ha l'effetto opposto.

Inoltre λ controlla la sensibilità della soluzione regolarizzata $\mathbf{x}_\lambda$ rispetto alle perturbazioni in A e $\mathbf{b}$, con un fattore di proporzionalità λ^{-1}. Ci sono due condizioni per l'applicazione del metodo di Tikhonov (4.4):

- Gli errori su $\mathbf{b}$ devono essere equi (*unbiased*) ;
- La loro matrice di covarianza deve essere proporzionale alla matrice identità.

Se quest'ultima condizione non è verificata occorre riscalare il problema o usare una versione regolarizzata del modello lineare generale di Gauss-Markov(7). Oltre alla regolarizzazione di Tikhonov, ci sono molti altri metodi di regolarizzazione, ciascuno adatto ad un particolare problema o ad un particolare calcolatore.

4.3. SVD ed SVD generalizzata (GSVD)

Gli strumenti per l'analisi di problemi mal posti sono la *decomposizione a valori singolari* (SVD) di A e la sua generalizzazione rispetto a due matrici, la *SVD generalizzata* (GSVD) della coppia (A, L). La SVD rivela tutte le difficoltà associate al cattivo condizionamento di una matrice, mentre la GSVD di (A, L) consente di esaminare il problema della regolarizzazione considerando sia la matrice dei coefficienti che la matrice di regolarizzazione L, come in (4.4).

4.3.1. Proprietà della SVD per problemi mal posti

In relazione ai problemi mal posti, ci sono due caratteristiche importanti della SVD di A:

- I valori singolari σ_i decadono gradualmente a zero senza nessun particolare salto nello spettro. Un incremento nelle dimensioni di A incrementerà il numero dei valori singolari piccoli.
- I vettori singolari sinistro e destro $\mathbf{u}_i, \mathbf{v}_i$ tendono ad avere più cambiamenti di segno nei loro elementi all'aumentare dell'indice i, cioè al decrescere dei σ_i.

Sebbene queste caratteristiche si trovino in molti problemi discreti connessi ad applicazioni pratica, sono difficili da dimostrare in generale. Per vedere come la SVD dia indicazioni sul cattivo condizionamento di A, si consideri le seguenti relazioni:

$$\begin{aligned} A\mathbf{v}_i &= \sigma_i \mathbf{u}_i \\ \|A\mathbf{v}_i\|_2 &= \sigma_i \end{aligned} \tag{4.5}$$

Un piccolo valore singolare σ_i, con $\|A\|_2 = \sigma_1$, indica che esiste una certa combinazione lineare delle colonne di A, caratterizzata dagli elementi del vettore singolare destro $\mathbf{v}_i$tale che $\|A\mathbf{v}_i\|_2 = \sigma_i$ sia piccolo. In altri termini, uno o più σ_i piccoli implicano che A è quasi *rank deficient,* e che i vettori $\mathbf{v}_i$ associati ai piccoli σ_i sono numericamente vettori nulli di A. Concludiamo quindi che la matrice A in un problema discreto mal posto è sempre fortemente mal condizionata.
La SVD fornisce anche indicazioni sullo *smoothing effect* tipicamente associato ad un kernel integrabile al quadrato. Si noti che quando σ_i decrescono, i vettori singolari $\mathbf{u}_i, \mathbf{v}_i$ oscillano sempre di più. Si consideri ora una mappatura $A\mathbf{x}$ di un vettore arbitrario $\mathbf{x}$. Usando la SVD otteniamo[13]:

$$A\mathbf{x} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i (\mathbf{v}_i{}^T \mathbf{x}) = \sum_{i=1}^{n} \sigma_i (\mathbf{v}_i{}^T \mathbf{x}) \mathbf{u}_i \tag{4.6}$$

Questa espressione mostra che a causa della moltiplicazione per σ_i le componenti di $\mathbf{x}$ ad alta frequenza sono più smorzate in $A\mathbf{x}$ delle componenti a bassa frequenza. Inoltre, il problema inverso, cioè il calcolo della $\mathbf{x}$ a partire da $A\mathbf{x} = \mathbf{b}$ oppure $\min\|A\mathbf{x} - \mathbf{b}\|_2$, dovrebbe avere l'effetto opposto: amplificare le oscillazioni ad alta frequenza di $\mathbf{b}$.

[13] $(\mathbf{v}_i{}^T \mathbf{x})$ è uno scalare.

Osservazione. Usando il seguente codice si può osservare il degrado del condizionamento della matrice A man mano che un *cluster* di valori singolari tende a zero.

Test_reg

```
sv = 1: 16 ;
for j = 1 : 16
  sv = [sv(1:9) sv(10:16)/10^(j-1)] ;
  n = length(sv);
  sv = sort(sv, 'descend');
  A = csgen(sv); % genera una matrice complessa simmetrica con i val.sing. sv
  condizionamento(j)=(cond(A));
  [U S V] = svd(A);
end
```

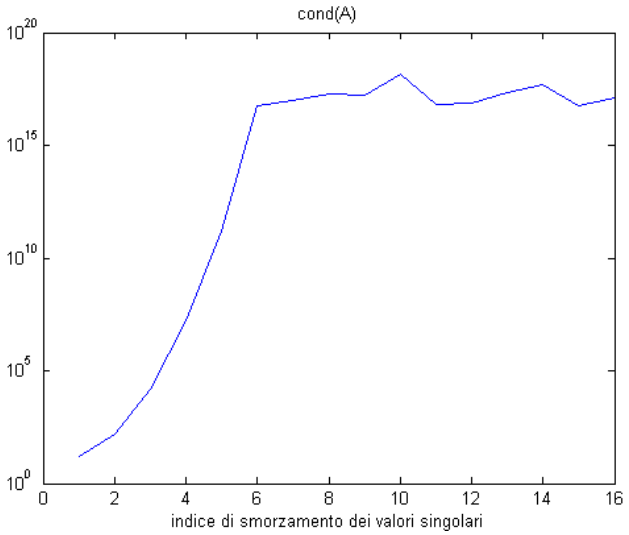


Figura 4-1 Effetto di un *cluster* di valori singolari tendenti a zero sul condizionamento.

4.3.2. Proprietà della GSVD per problemi mal posti

Per problemi discreti mal posti è possibile dire qualcosa sul legame SVD-GSVD perché L è tipicamente una matrice ben condizionata. In questo caso si può mostrare che anche X è ben condizionata. Perciò, la matrice diagonale Σ deve mostrare il cattivo condizionamento di A, e siccome $\gamma_i = \sigma_i\left(1-\sigma_i^2\right)^{1/2} \approx \sigma_i$ per piccoli σ_i i valori singolari generalizzati devono decadere gradualmente a zero come fanno i valori singolari ordinari. Inoltre le proprietà oscillatorie (l'aumento dei cambiamenti di segno) dei vettori singolari destri si trasporta alle colonne di X della GSVD (si veda 2.5): più piccoli sono i γ_i e più variazioni di segno ci saranno in $\mathbf{x}_i$.

Un esempio immediato di uso della GSVD nella analisi della regolarizzazione di Tikhonov. Siano E ed $\mathbf{e}$ le perturbazioni di A e $\mathbf{b}$, e sia $\bar{\mathbf{x}}_\lambda$ la soluzione esatta del problema imperturbato; allora l'errore relativo nella soluzione perturbata $\mathbf{x}_\lambda$ soddisfa a:

$$\frac{\|\mathbf{x}_\lambda - \bar{\mathbf{x}}_\lambda\|_2}{\|\bar{\mathbf{x}}_\lambda\|_2} \leq \frac{\|A\|_2\|X\|_2\lambda^{-1}}{1-\|E\|_2\|X\|_2\lambda^{-1}} \cdot \left(1 + \text{cond(X)}\frac{\|E\|_2}{\|A\|_2} + \frac{\|\mathbf{e}\|_2}{\|\mathbf{b}_\lambda\|_2} + \|E\|_2\|X\|_2\lambda^{-1}\frac{\|\mathbf{r}_\lambda\|_2}{\|\mathbf{b}_\lambda\|_2}\right) \qquad (4.7)$$

Dove $\mathbf{b}_\lambda = A\mathbf{x}_\lambda$ e $\mathbf{r}_\lambda = \mathbf{b} - \mathbf{b}_\lambda$. La conclusione importante che per tutti i ragionevoli λ, il legame tra la perturbazione e l'errore nella soluzione è proporzionale a λ^{-1} e alla norma della matrice X. Inoltre si può dimostrare che $\|X\|_2$ è approssimativamente proporzionale all'inverso del più piccolo valore singolare di L, cioè $\left\|L^\dagger\right\|_2$. Quindi, oltre a controllare la *smoothness* della soluzione regolarizzata, λ ed L controllano anche la sensibilità alle perturbazioni in A e $\mathbf{b}$.

4.4. Condizione discreta di Picard e Fattore di filtro

Nelle applicazioni reali, l'RHS è sempre contaminata da diversi tipi di errori e può essere scritta come:

$$\mathbf{b} = \bar{\mathbf{b}} + \mathbf{e} \tag{4.8}$$

Dove $\mathbf{e}$ sono gli errori e $\bar{\mathbf{b}}$ è l'RHS imperturbata. Se volessimo calcolare una soluzione $\mathbf{x}_{\text{reg}}$ regolarizzata usando $\mathbf{b}$, la $\bar{\mathbf{b}}$ dovrebbe soddisfare la:

Condizione di Picard discreta. L'RHS imperturbata $\bar{\mathbf{b}}$ di un problema discreto mal posto con matrice di regolarizzazione L soddisfa alla condizione di Picard discreta se i coefficienti di Fourier $\left|\mathbf{u}_i^T\mathbf{b}\right|$ hanno un decadimento medio verso lo zero più rapido di quello dei valori singolari generalizzati γ_i.

Si consideri il sistema lineare (4.1) e il problema ai minimi quadrati (4.2) e si assuma per semplicità che A non abbia valori singolari esattamente nulli. Usando la SVD, si mostra che la soluzione di entrambi i sistemi è data dalla stessa equazione:

$$\mathbf{x}_{\text{LSQ}} = A^{-1}\mathbf{b} = \sum_{i=1}^{n}\left(\frac{1}{\sigma_i}\right)\mathbf{v}_i\mathbf{u}_i^T\,\mathbf{b} = \sum_{i=1}^{n}\left(\frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\right)\mathbf{v}_i \tag{4.9}$$

Questa relazione illustra chiaramente le difficoltà che si incontrano con la soluzione standard di (4.1)-(4.2). Siccome i coefficienti di Fourier $\left|\mathbf{u}_i^T\mathbf{b}\right|$ corrispondenti ai piccoli valori di σ_i non decadono così rapidamente come i valori singolari, la soluzione $\mathbf{x}_{\text{LSQ}}$ è dominata dai termini nella somma che corrispondono ai valori più piccoli di σ_i. Di conseguenza la $\mathbf{x}_{\text{LSQ}}$ ha molte variazioni di segno e quindi sembra completamente casuale.

Perciò lo scopo della regolarizzazione è quello di smorzare o filtrare i contributi alla soluzione corrispondenti ai piccoli valori singolari generalizzati. Quindi, si richiederà che un metodo di regolarizzazione produca una soluzione regolarizzata che, a partire da una stima della soluzione nulla $\tilde{\mathbf{x}} = \mathbf{0}$, possa esser scritta come:

$$\begin{aligned}\mathbf{x}_{\text{reg}} &= \sum_{i=1}^{n} f_i\left(\frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\right)\mathbf{v}_i && \text{se } L = I_n\\ \mathbf{x}_{\text{reg}} &= \sum_{i=1}^{p} f_i\left(\frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\right)\mathbf{x}_i + \sum_{i=p+1}^{n}\left(\mathbf{u}_i^T\mathbf{b}\right)\mathbf{x}_i && \text{se } L \neq I_n\end{aligned} \tag{4.10}$$

I numeri f_i sono detti *fattori di filtro* per il particolare metodo di regolarizzazione. Questi devono tendere a zero quando i σ_i decrescono, in modo che i contributi $\left(\mathbf{u}_i^T\mathbf{b}/\sigma_i\right)\mathbf{x}_i$ alla soluzione finale siano effettivamente filtrati dai piccoli σ_i. La differenza tra i vari metodi di regolarizzazione risiede essenzialmente nel modo in cui questi fattori di filtro f_i sono definiti.
Per la regolarizzazione di Tikhonov, fondamentale nella teoria della regolarizzazione, i fattori di filtro, sono:

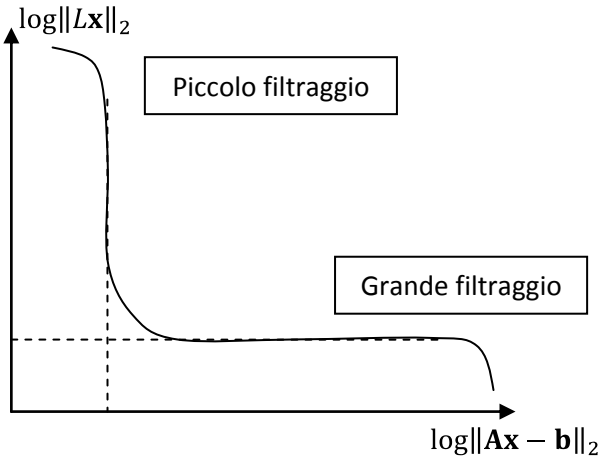
$$\begin{aligned}f_i &= \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2} && \text{se } L = I_n \text{ , valido per } \sigma_i < \lambda\\ f_i &= \frac{\gamma_i^2}{\gamma_i^2 + \lambda^2} && \text{se } L \neq I_n, \text{valido per } \gamma_i < \lambda\end{aligned} \tag{4.11}$$

In particolare, questo mostra che i problemi mal posti non sono regolarizzati dal metodo di Tikhonov se $\sigma_1 < \lambda$ e $\gamma_p < \lambda$.

I fattori di filtro per i vari metodi di regolarizzazione possono essere calcolati tramite un routine `fil_fac` del pacchetto, mentre la routine `picard` fornisce il grafico di quantità importanti come σ_i, $|\mathbf{u}_i\mathbf{b}|$, e $\left(\mathbf{u}_i^T\mathbf{b}/\sigma_i\right)$ se $L = I_n$ oppure γ_i, $|\mathbf{u}_i\mathbf{b}|$, e $\left(\mathbf{u}_i^T\mathbf{b}/\gamma_i\right)$ se $L \neq I_n$.

4.5. La curva L

Forse lo strumento grafico più conveniente per l'analisi di problemi discreti mal posti è la cosiddetta *curva L,* cioè un grafico – per tutti i parametri di regolarizzazione validi – della (semi)norma $\|L\mathbf{x}_{\text{reg}}\|$della soluzione regolarizzata rispetto alla norma del residuo $\|A\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2$. In questo modo la curva L mostra chiaramente il compromesso tra la minimizzazione di queste due quantità.



Per problemi discreti mal posti risulta che la curva L, quando rappresentata in scala *log-log,* ha talvolta questa forma (da cui il nome) con uno spigolo che separa la parte verticale da quella orizzontale.
Notiamo che se $\bar{\mathbf{x}}$ è la soluzione esatta, non regolarizzata corrispondente all'esatta RHS $\bar{\mathbf{b}}$, allora l'errore $\mathbf{x}_{\text{reg}} - \bar{\mathbf{x}}$ è costituito da due componenti:

- un errore di perturbazione dovuto all'errore $\mathbf{e}$ nell'RHS $\mathbf{b}$**;**
- un errore di regolarizzazione dovuto alla regolarizzazione della componente libera da errori $\bar{\mathbf{b}}$ nell'RHS.

La porzione verticale della curva L corrisponde alle soluzioni per cui $\|L\mathbf{x}_{\text{reg}}\|$ è molto sensibile alle variazioni del parametro di regolarizzazione perché l'errore di perturbazione $\mathbf{e}$ domina su $\mathbf{x}_{\text{reg}}$ e perché $\mathbf{e}$ non soddisfa la condizione discreta di Picard.
La porzione orizzontale della curva L corrisponde alle soluzioni in cui la norma del risiduo $\|A\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2$ è più sensibile al parametro di regolarizzazione perché $\mathbf{x}_{\text{reg}}$ è dominato dall'errore di regolarizzazione – finchè $\bar{\mathbf{b}}$ soddisfa la condizione di Picard.
Possiamo dimostrare questo tramite le relazioni per la soluzione regolarizzata $\mathbf{x}_{\text{reg}}$ in termini di fattori di filtro. Per una regolarizzazione nella forma generale ($L \neq I_n$) l'equazione (4.10) conduce alla seguente espressione per l'errore in $\mathbf{x}_{\text{reg}}$:

$$\mathbf{x}_{\text{reg}} - \bar{\mathbf{x}} = \left(\sum_{i=1}^{p} f_i \left(\frac{\mathbf{u}_i^T\mathbf{e}}{\sigma_i}\right)\mathbf{x}_i + \sum_{i=p+1}^{n} (\mathbf{u}_i^T\mathbf{e})\mathbf{x}_i\right) + \sum_{i=1}^{p} (f_i - 1)\left(\frac{\mathbf{u}_i^T\bar{\mathbf{b}}}{\sigma_i}\right)\mathbf{x}_i \qquad (4.12)$$

Qui, il termine tra parentesi è l'*errore di perturbazione* dovuto alla perturbazione $\mathbf{e}$**,** e il secondo termine è l'*errore di regolarizzazione.* Quando si introduce una piccola regolarizzazione, la maggior parte dei fattori di

filtro sono approssimativamente 1 e l'errore $\mathbf{x}_{\text{reg}} - \bar{\mathbf{x}}$ è dominato dall'errore di perturbazione. Dall'altro lato, con una grande regolarizzazione la maggior parte dei fattori di filtro sono piccoli $f_i \ll 1$, ed $\mathbf{x}_{\text{reg}} - \bar{\mathbf{x}}$ è dominato dall'errore di regolarizzazione.

Si può dimostrare che se $\bar{\mathbf{b}}$ soddisfa alla condizione di Picard, allora la parte orizzontale della curva corrisponde alle soluzioni in cui gli errori di regolarizzazione dominano – cioè in cui è introdotto la maggior parte del filtraggio che la soluzione risulta veramente *smooth* e quindi $\|L\mathbf{x}_{\text{reg}}\|_2$ cambia poco rispetto al parametro di regolarizzazione. Di contro, la parte verticale della curva L corrisponde alle soluzioni che sono dominate dall'errore di perturbazione e dovute alla divisione per i piccoli σ_i; è chiaro che $\|L\mathbf{x}_{\text{reg}}\|_2$ varia drammaticamente con il parametro di regolarizzazione mentre, simultaneamente la norma del residuo non cambia tanto. Inoltre si può mostrare che la scala *log-log* enfatizza i diversi aspetti delle parti orizzontali e verticali. In questo modo, la curva L mostra chiaramente il compromesso tra la minimizzazione della norma del residuo e il vincolo.

La curva L è continua quando il parametro di regolarizzazione è continuo, come nella regolarizzazione di Tikhonov. Per altri metodi di regolarizzazione, con un parametro di regolarizzazione discreto, come nella TSVD, è possibile rappresentare la curva L solo per un numero finito di punti.

La curva L per la regolarizzazione di Tikhonov divide il primo quadrante in due regioni. È impossibile costruire qualsiasi soluzione che corrisponda ad un punto al di sotto della curva L di Tikhonov; qualsiasi soluzione regolarizzata deve giacere sulla o al di sopra di questa curva. La soluzione calcolata tramite la regolarizzazione di Tikhonov è quindi ottima nel senso che per una data norma del residuo non può esistere una soluzione con una seminorma più piccola di quella della soluzione di Tikhonov – e viceversa. Una conseguenza di questo fatto è che è possibile confrontare altri metodi di regolarizzazione con la regolarizzazione di Tikhonov ispezionando quando vicine sono le altre curve L dei metodi alternativi alla curva L di Tikhonov. Se $\bar{\mathbf{b}}$ soddisfa alla condizione discreta di Picard, allora le due curve L sono vicine e le soluzioni calcolate tramite i due metodi di regolarizzazione sono simili.

Per una data RHS $\mathbf{b} = \bar{\mathbf{b}} + \mathbf{e}$, c'è un parametro ottimo di regolarizzazione che bilancia l'errore di perturbazione e l'errore di regolarizzazione in $\mathbf{x}_{\text{reg}}$. Una caratteristica essenziale della curva L è che questo parametro ottimo di regolarizzazione non è lontano dal parametro di regolarizzazione corrispondente all'angolo della curva L. In altre parole, localizzando l'angolo della curva L è possibile calcolare una approssimazione del parametro ottimo di regolarizzazione e quindi, calcolare una soluzione regolarizzata con un buon bilanciamento tra i due tipi di errori.

Nel pacchetto di regolarizzazione, la routine `l_curve` produce un grafico log-log della curva L e – se richiesto – trova l'angolo e identifica il corrispondente parametro di regolarizzazione. Dato un insieme discreto di valori di $\|A\mathbf{x}_{\text{reg}} - \mathbf{b}\|_2$ e $\|L\mathbf{x}_{\text{reg}}\|_2$, la routine `plot_lc` grafica la curva L corrispondente, mentre la routine `l_corner` individua l'angolo della curva L.

4.6. Trasformazione nella forma standard

Un problema di regolarizzazione con vincolo $\Omega(\mathbf{x}) = \|L(\mathbf{x} - \mathbf{x_0})\|$ è detta essere in *forma standard* se la matrice L è la matrice identità I_n. In molte applicazioni, la regolarizzazione in forma standard non è la scelta migliore, e si dovrebbe usare $L \neq I_n$ nel vincolo $\Omega(\mathbf{x})$. La scelta opportuna della matrice L dipende dall'applicazione particolare.

Tuttavia dal punto di vista numerico è più semplice trattare problemi in forma standard, fondamentalmente perché solo una matrice, A, è coinvolta invece che le due matrici A ed L. Perciò, è conveniente essere in grado di trasformare un dato problema di regolarizzazione in forma generale in uno equivalente in forma standard usando metodi numericamente stabili. Per esempio, si vorrebbe un metodo

numericamente stabile per la regolarizzazione di Tikhonov per trasformare il problema in forma generale (4.4) nel seguente problema in forma standard:

$$\min\left\{\left\|\bar{A}\bar{\mathbf{x}}-\bar{\mathbf{b}}\right\|_2^2+\lambda^2\|L(\bar{\mathbf{x}}-\bar{\mathbf{x}}_{\mathbf{0}})\|_2^2\right\} \tag{4.13}$$

Dove $\bar{A},\bar{\mathbf{x}},\bar{\mathbf{b}},\bar{\mathbf{x}}_{\mathbf{0}}$ dipendono dalle quantità originali $A,\mathbf{x},\mathbf{b},\mathbf{x}_{\mathbf{0}}$. Inoltre si vorrebbe uno schema numericamente stabile per trasformare la soluzione $\bar{\mathbf{x}}_\lambda$ nella forma generale $\mathbf{x}_\lambda$. Inoltre è preferibile una trasformazione che conduca ad una semplice relazione tra la SVD di $\bar{A}$ e la GSVD di (A,L).
Nel semplice caso in cui L è quadrata e invertibile, la trasformazione è ovvia:

$$\begin{aligned}\bar{A}&=AL^{-1}\\ \bar{\mathbf{b}}&=\mathbf{b}\\ \bar{\mathbf{x}}_{\mathbf{0}}&=\mathbf{L}\mathbf{x}_{\mathbf{0}}\\ \mathbf{x}_\lambda&=L^{-1}\bar{\mathbf{x}}_\lambda\end{aligned} \tag{4.14}$$

Nella maggior parte delle applicazioni, tuttavia, la matrice L non è quadrata, e la trasformazione diventa più complicata di una semplice inversione. Occorre distinguere tra metodi di regolarizzazione diretti e iterativi. Per i metodi diretti occorre calcolare $\bar{A}$ esplicitamente attraverso metodi standard come la fattorizzazione QR. Per metodi iterativi, d'altra parte occorre calcolare solo il prodotto $\bar{A}\bar{\mathbf{x}}$ in modo efficiente. Per maggiori dettagli sulle trasformazioni per metodi diretti e iterativi si veda (7).

4.7. Metodi di regolarizzazione diretti

In questa e nella prossima sezione segue un elenco dei metodi di regolarizzazione per il trattamento numerico di problemi discreti mal posti compresi nel REGULARIZATION TOOLS. Questa sezione si focalizza su metodi diretti, in cui la soluzione è ricavata tramite un calcolo diretto (che potrebbe comprendere anche una procedura iterativa per il calcolo delle radici), mentre i metodi di regolarizzazione che sono intrinsecamente iterativi sono trattati nella sezione successiva.

4.7.1. Regolarizzazione di Tikhonov

Il metodo di Tikhonov è un metodo diretto perché la soluzione regolarizzata $\mathbf{x}_\lambda$ definita in(4.4) è la soluzione al seguente problema ai minimi quadrati:

$$\mathbf{x}_\lambda=\operatorname{argmin}\left\|\begin{pmatrix}A\\ \lambda L\end{pmatrix}\mathbf{x}-\begin{pmatrix}\mathbf{b}\\ \lambda L\mathbf{x}_0\end{pmatrix}\right\|_2 \tag{4.15}$$

Ed è facile vedere che $\mathbf{x}_\lambda$ è unica se i *null space*[14] di A ed L banalmente si intersecano (come accade in pratica di solito (?)). L'algoritmo più efficiente per il trattamento numerico del metodo di Tikhonov per una matrice di regolarizzazione generale L è costituito da 3 passi:

1. Il problema è trasformato in forma standard;
2. La matrice $\bar{A}$ è trasformata in una bidiagonale superiore $p\times p$ $\bar{B}$ tramite:

$$\bar{A}=\bar{U}\bar{B}\bar{V}^T \tag{4.16}$$

3. Infine il problema è risolto per $\bar{V}^T\bar{\mathbf{x}}_\lambda$ e la soluzione è ritrasformata.

[14] Spazio nullo.

Nel pacchetto di regolarizzazione si usa un altro approccio per la risoluzione della (4.15) usando i fattori di filtro della GSVD esplicitamente (o la SVD, se $L = I_n$). Questo approccio, implementato nella routine `tikhonov`, è più adatto a Matlab.

4.7.2. TSVD

Il metodo precedente (ma anche altri menzionati in (7)) aggira il cattivo condizionamento di A introducendo un nuovo problema (4.15) con una matrice dei coefficienti ben condizionata a rango pieno $\begin{pmatrix} A \\ \lambda L \end{pmatrix}$. Un modo diverso per trattare il cattivo condizionamento di A è quello di derivare un nuovo problema con una matrice dei coefficienti *rank deficient* ben condizionata. Un risultato fondamentale sulle matrici *rank deficient,* che può essere derivato dalla SVD di A, è che la più vicina approssimazione A_k di rango k di A - misurata in norma 2 – si ottiene troncando l'espansione SVD al *k-esimo* termine, cioè:

$$A_k = \sum_{i=1}^{k} \mathbf{u_i} \sigma_i \mathbf{v_i^T} \quad k \leq n \tag{4.17}$$

Il metodo della SVD troncata (TSVD) è basato su questa osservazione per la risoluzione di:

$$\min\|\mathbf{x}\|_2 \quad \text{tale che} \quad \min\|A_k\mathbf{x} - \mathbf{b}\|_2 \text{[15]} \tag{4.18}$$

La soluzione a questo problema è data da:

$$\mathbf{x_k} = \sum_{i=0}^{k} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i \tag{4.19}$$

La TSVD può essere calcolata tramite `tsvd`.

4.8. Metodi di regolarizzazione iterativi

Questa sezione descrive i metodi di regolarizzazione iterativi compresi nel REGULARIZATION TOOLS. Queste *routine* costituiscono dei modelli di implementazione; le implementazioni reali dovrebbero incorporare il trattamento di qualsiasi sparsità e/o struttura della matrice A.

4.8.1. Algoritmo del gradiente coniugato

L'algoritmo del gradiente coniugato (CG) è un metodo per la risoluzione di sistemi di equazioni sparsi con matrice dei coefficienti simmetrica definita positiva. Quando l'algoritmo CG è applicato alle equazioni normali non regolarizzate $A^TA\mathbf{x} = A^T\mathbf{b}$ (implementato in modo che A^TA non sia eseguito) le componenti a bassa frequenza della soluzione tendono a convergere più rapidamente di quelle ad alta frequenza. Perciò, il processo CG ha un intrinseco effetto regolarizzante quando il numero delle iterazioni gioca il ruolo del parametro di regolarizzazione. Il *k-esimo* passo del processo CG ha la forma:

[15] $\min\|L\mathbf{x}\|_2 \quad \text{tale che} \quad \min\|A_k\mathbf{x} - \mathbf{b}\|_2$ è riferito alla MTSVD, si veda (7).

$$\begin{aligned} \beta_k &\leftarrow \frac{\left\|\mathbf{q}^{k-1}\right\|_2^2}{\left\|\mathbf{q}^{k-2}\right\|_2^2} \\ \mathbf{p}^k &\leftarrow \mathbf{q}^{k-1} + \beta_k \mathbf{p}^{k-1} \\ \alpha_k &\leftarrow \frac{\left\|\mathbf{q}^{k-1}\right\|_2^2}{\left\|A^T A\mathbf{p}^k\right\|_2^2} \\ \mathbf{x}^k &\leftarrow \mathbf{x}^{k-1} + \alpha_k \mathbf{p}^k \\ \mathbf{q}^k &\leftarrow \mathbf{q}^{k-1} + \alpha_k A^T A\mathbf{p}^k \end{aligned} \tag{4.20}$$

Dove $\mathbf{x}^k$ è l'approssimazione di $\mathbf{x}$ dopo *k* iterazioni, mentre $\mathbf{p}^k$ e $\mathbf{q}^k$ sono due vettori di iterazione ausiliari di lunghezza *n*. L'effetto regolarizzante del metodo CG è descritto in dettaglio in (7).
La soluzione $\mathbf{x}^k$ dopo *k* passi CG può essere definita come:

$$\min\|Ax - b\|_2 \quad \text{tale che} \quad \mathbf{x} \in \mathcal{K}_k(A^T A, A^T \mathbf{b}) \tag{4.21}$$

Dove $\mathcal{K}_k(A^T A, A^T \mathbf{b})$ è il sottospazio di Krylov(2) associato alle equazioni normali. Perciò vediamo che il CG rimpiazza il vincolo $\Omega(\mathbf{x}) = \|\mathbf{x}\|_2$ con il vincolo $\mathbf{x} \in \mathcal{K}_k(A^T A, A^T \mathbf{b})$. Anche la migliore implementazione dell'algoritmo CG con equazioni normali soffre di una certa inaccuratezza dovuta all'uso implicito del prodotto matriciale $A^T A$. Per questo sono stati proposti altri metodi (7).
Si sono diversi modi per implementare l'algoritmo CG per le equazioni normali in modo numericamente stabile. Nel Regularization Tool è possibile usare l'algoritmo CG tramite la funzione `cgls`.

4.9. Metodi per la scelta del Parametro di Regolarizzazione

Un buon parametro di regolarizzazione dovrebbe fornire un buon compromesso tra l'errore di perturbazione e l'errore di regolarizzazione nella soluzione regolarizzata. I metodi per il calcolo del parametro di regolarizzazione possono essere suddivisi in due categorie a seconda delle assunzioni fatte su $\|\mathbf{e}\|_2$, che è la norma della perturbazione dell'RHS. Le due classi possono essere caratterizzate come segue:

1. Metodi basati sulla conoscenza, o su una buona stima di $\|\mathbf{e}\|_2$;
2. Metodi che non richiedono $\|\mathbf{e}\|_2$, ma che invece cercano di estrarre le informazioni necessarie dalla RHS fornita.

Di seguito sono esposti solo i metodi del secondo tipo.

Criterio della curva L (l-curve, l-corner)

Questo criterio è già stato discusso in relazione alla introduzione sulla curva L. Per un parametro di regolarizzazione continuo λ calcoliamo la curvatura della curva:

$$(\log\|A\mathbf{x} - \mathbf{b}\|_2, \log\|L\mathbf{x}_\lambda\|_2) \tag{4.22}$$

(Con λ come suo parametro) e cerchiamo il punto con la massima curvatura, che definiamo angolo della curva L. Quando il parametro di regolarizzazione è discreto si approssima la curva discreta in scala *log-log* con una *spline* in 2D, si calcola il punto sulla *spline* con la massima curvatura, e si definisce l'angolo della curva L discreta come quel punto che è più vicino all'angolo della curva *spline*. Questo criterio è implementato in due routines `l_curve` ed `l_corner`.

Cross-validazione generalizzata (GCV)

La GCV è basata sull'idea che se un elemento arbitrario b_i dell'RHS $\mathbf{b}$ è omesso, allora la corrispondente soluzione regolarizzata dovrebbe predire questa osservazione bene, e la scelta del parametro di regolarizzazione dovrebbe essere indipendente dalle trasformazioni ortogonali di $\mathbf{b}$. Questo e altre motivazioni (7) conducono alla scelta di un parametro di regolarizzazione che minimizza la funzione GCV:

$$G \equiv \frac{\left\| A\mathbf{x}_{\text{reg}} - \mathbf{b} \right\|_2^2}{\text{trace}(I_m - AA^I)^2} \tag{4.23}$$

Dove A^I è la matrice che produce la soluzione regolarizzata $\mathbf{x}_{\text{reg}}$ quando è moltiplicata per $\mathbf{b}$, cioè $\mathbf{x}_{\text{reg}} = A^I\mathbf{b}$. Si noti che G è definita sia per parametri di regolarizzazione sia discreti che continui. Il denominatore della (4.23) può essere calcolato in $O(n)$ operazioni se si usa l'algoritmo di bidiagonalizzazione […]. In alternativa, si possono usare i fattori di filtro per valutare il denominatore tramite la semplice espressione:

$$\text{trace}(I_m - AA^I) = m - (n - p) - \sum_{i=1}^{p} f_i \tag{4.24}$$

Questo è l'approccio usato dalla funzione `gcv`. Si può mostrare che il metodo GCV cerca di bilanciare gli errori di perturbazione e di regolarizzazione e quindi, è legato all'angolo della curva L.

5. Test degli algoritmi per la SVD

5.1. Costruzione di un problema test per matrici quadrate

Costruire un problema test significa fissarne anticipatamente la soluzione. Una volta risolto, è possibile confrontare la soluzione ottenuta con la soluzione esatta.
Nel nostro caso occorre stabilire anticipatamente la dimensione della matrice. Di seguito si costruisce un vettore di valori singolari casuali (positivi) disposti in ordine decrescente. L'obiettivo è quello di costruire una matrice $A \in \mathbb{R}^{n\times n}$ i cui valori singolari siano quelli fissati, tale che $A = Q\Sigma Q^T$, dove Q è una matrice unitaria e Σ la diagonale contenente i valori singolari. Si usa $A = Q\Sigma Q^T$, cioè la *fattorizzazione di Takagi,* che è una forma particolare di SVD in cui $U = V$, in quanto si dispone di uno strumento (`Takagi Factorization Package`) per la rapida costruzione di matrici per questo tipo di fattorizzazione (per dettegli vedere (8)).
All'interno del `Takagi Factorization Package` sono disponibili tre funzioni utili per il nostro scopo: **`csgen`**, **`unitrand`**, **`house`**. Il problema test è generato come segue:

`test_svd:` genera una matrice con i valori singolari definiti in sv

```
randn('seed',41998);
sv = abs(randn(n,1)); %scelgo dei valori singolari
sv = sort(sv, 'descend');
A = csgen(sv); % genera una matrice simmetrica con i val.sing. sv
```

La funzione **`csgen`** genera una matrice simmetrica reale[16] i cui valori singolari sono quelli specificati in `sv`. Si tenga presente che i valori singolari devono essere reali e non negativi.

`csgen:`

```
function [A,U] = csgen(s)
if (any(imag(s) ~= 0))
    error('Singular values must be real.')
end
if (any(s < 0))
    error('Singular values must be nonnegative.')
end
n = length(s);
U = unitarand(n,n);
A = U*diag(s)*conj(U');
A = triu(A) + tril(conj(A'),-1); % forza la simmetria
```

La funzione costruisce una matrice unitaria secondo quanto prescritto in (9) usando successive trasformazioni di Householder.

`unitarand:` Costruizione di una matrice unitaria random

```
function U = unitarand(m,n)
% initialize a random diagonal unitary matrix
x= (ones(m,1) - 2*rand(m,1));
U= diag(sign(x));
%
for i=m:-1:max(2,m-n+1),
```

[16] In verità il pacchetto genera matrici complesse: è stato modificato per generare matrici reali.

```
    % generate an i-dim. random vector with
    % uniform distribution on [-1, 1]
    x= (ones(i,1) - 2*rand(i,1));
    % generate Householder matrix
    [u,beta,alpha]= house(x);
    % accumulate Householder transformations
    U(:,m-i+1:m)= U(:,m-i+1:m) - (U(:,m-i+1:m)*u)*(beta*u');
end;
U= U(:,1:n);
```

house:

```
function [u,beta,alpha]= house(x)
% [u,beta,alpha] = house(x)
% H = eye(length(x)) - beta*u*u'
% u is scaled so that u(1)=1, so u(2:n) is the essential part.
[nr,nc] = size(x);
if nc~=1
   error('Input x to house is not a column vector.')
end
% scaling to avoid over(under)flow
m = max(abs(x));
x = x/m;
%
normx = norm(x);     % normx >= 1.0
%
% get sign(x1)
if x(1)==0       % in matlab sign(0)=0
    signx1 = 1.0;
else
    signx1 = sign(x(1));
end
%
alpha = signx1*normx;    % |alpha| >= 1.0
%
u = x;
u(1)= u(1) + alpha;      % |u(1)| >= 1.0
beta = 1/(alpha'*u(1));
%
% make u(1)=1, so u(2:n) is the essential part of u
beta = beta*(u(1)*u(1)');   % update beta
u = u/u(1);
% scale back
alpha = m*alpha;
```

5.2. Algoritmi da testare, problema e criteri di prestazione

Sotto è mostrato nuovamente l'elenco degli algoritmi da testare (per dettagli sugli algoritmi vedere : Algoritmi per il calcolo della SVD).

Tabella 5.1 – Elenco degli algoritmi da testare.

Algoritmo	Algoritmo per il calcolo della QR	Id. Algoritmo	Rif. nel testo
`SVD_ver1`	Householder (1)	1	3.4.1
	MATLAB	2	3.4.2
`SVD_ver2` – Golub-Kahan-Reinsch	MATLAB	3	3.4.3
`SVD_ver3` – Fast SVD	Householder (1)	5	3.4.4
	Hessemberg (1) 5 iterazioni	6	
	Hessemberg (1) 10 iterazioni	8	
	Hessemberg (1) 15 iterazioni	9	
`svd` di MATLAB	MATLAB	7	3.4.5

Il problema da risolvere consiste nel calcolo dei valori singolari ed eventualmente anche delle matrici U, V. Segue un elenco dei criteri di prestazione:

Tabella 5.2 – Criteri di prestazione.

Criterio di prestazione	Formula
Errore di ortogonalizzazione su U	$\|UU^* - I\|_F$
Errore di ortogonalizzazione su V	$\|VV^* - I\|_F$
Errore sui valori singolari	$\|s - sv\|/\|sv\|$
Errore nella fattorizzazione QR	$\|A - U\Sigma V^*\|_F$
Tempo di calcolo	`tic, toc`

Criteri di prestazione:

```
Err_ortog_U(n,algor) = norm(NANSUM(U'*U-eye(n)), 'fro')/norm(1);
Err_ortog_V(n,algor) = norm(NANSUM(V'*V-eye(n)), 'fro')/norm(1);
Err_qr_fatt(n,algor) = norm(NANSUM(A-U*S*V'),'fro')/norm(A,'fro');
Err_sv(n,algor)      = norm(NANSUM(sort(diag(S))' - sort(sv)))/norm(sv);
tempo(n, algor);
```

Nel calcolo della norma per la valutazione dell'errore è stata impiegata la funzione di MATLAB NANSUM. Questa ritorna la somma degli argomenti trattando i NaN come se fossero dei valori mancanti. Questo risulta essere particolarmente utile nei casi in cui, ad esempio, gli ultimi elementi della diagonale sono NaN, mentre i primi contengono i valori singolari esatti.

Dai grafici riportati sotto è evidente il migliore funzionamento della funzione svd di MATLAB, sia dal punto di vista degli errori che del tempo di elaborazione. In Figura 5-2 si osserva un cattivo funzionamento nell'algoritmo 1 con picchi nell'errore che raggiungono valori prossimi a 1. Lo stesso discorso vale per la Figura 5-3. Le porzioni di curva dell'errore sono dovute all'assenza del calcolo delle matrici dei vettori destri e sinistri singolari. Questo fenomeno è marcato nel caso della fast svd calcolata anch'essa con fattorizzazione QR di Householder: le matrici U e V iniziano a 'mancare' a partire da $n = 20$. Gli altri algoritmi mantengono un profilo dell'errore piuttosto basso con alla base sempre l'algoritmo interno di

MATLAB. Seguono gli errori nel calcolo dei valori singolari, della fattorizzazione QR e la rapidità del calcolo degli algoritmi nel caso di matrici ‘magre’.

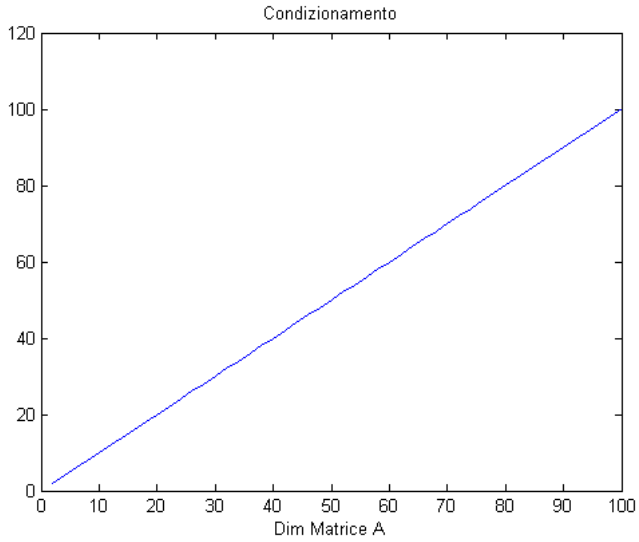


Figura 5-1 Condizionamento in norma 2 delle matrici A al crescere della dimensione.

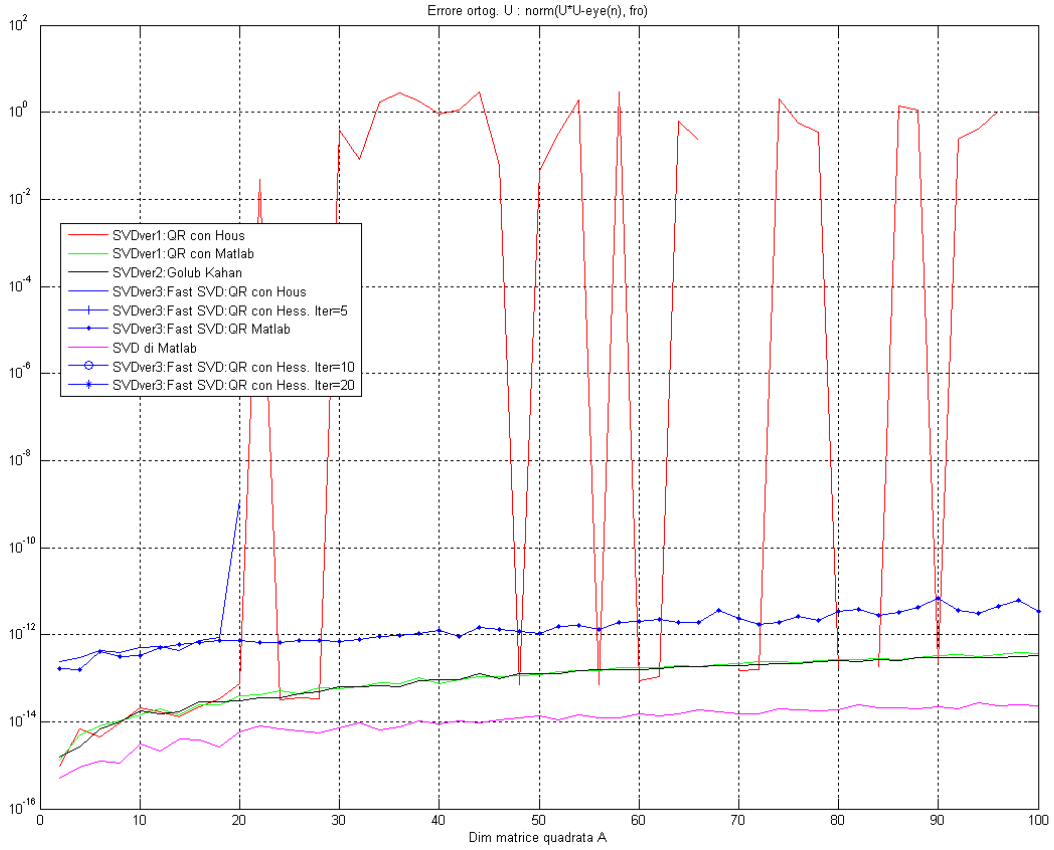


Figura 5-2 - Errore nell'ortogonalizzazione di U, la prestazione dell'algoritmo SVD_ver1, della QR con Householder è disastrosa. Se mancano dati nella curva significa che la U non è stata calcolata.

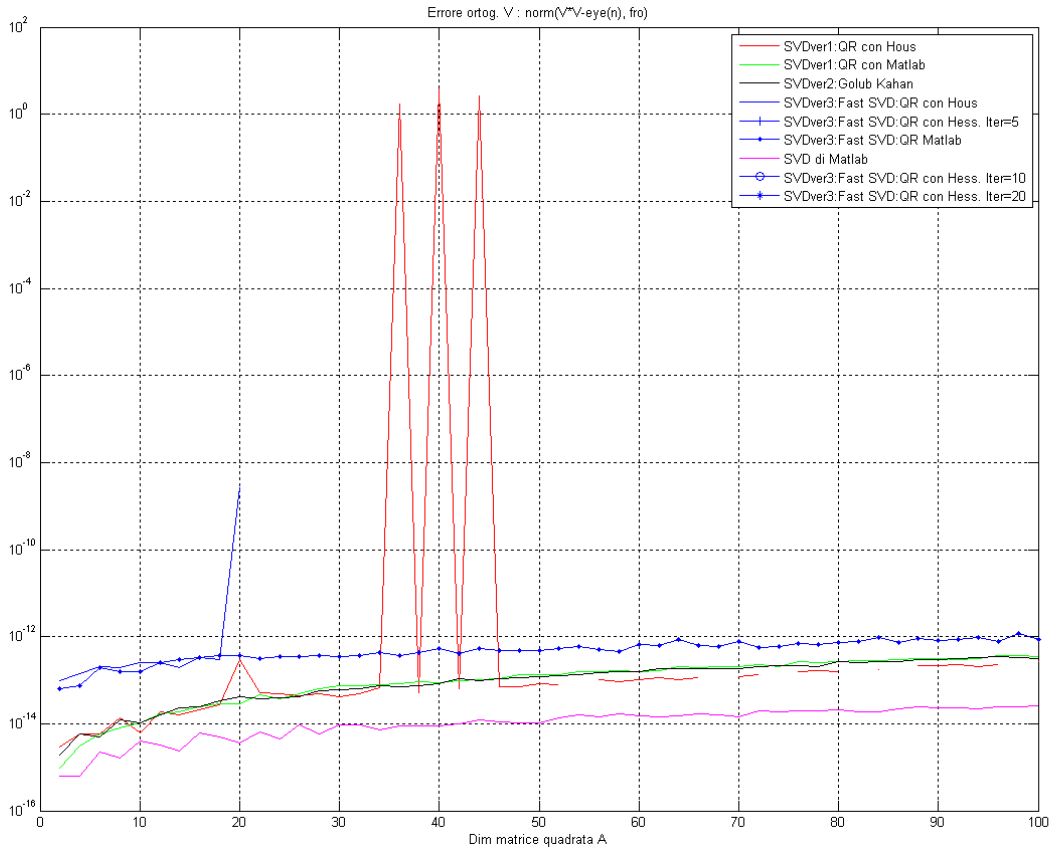


Figura 5-3 - Errore nell'ortogonalizzazione di V.

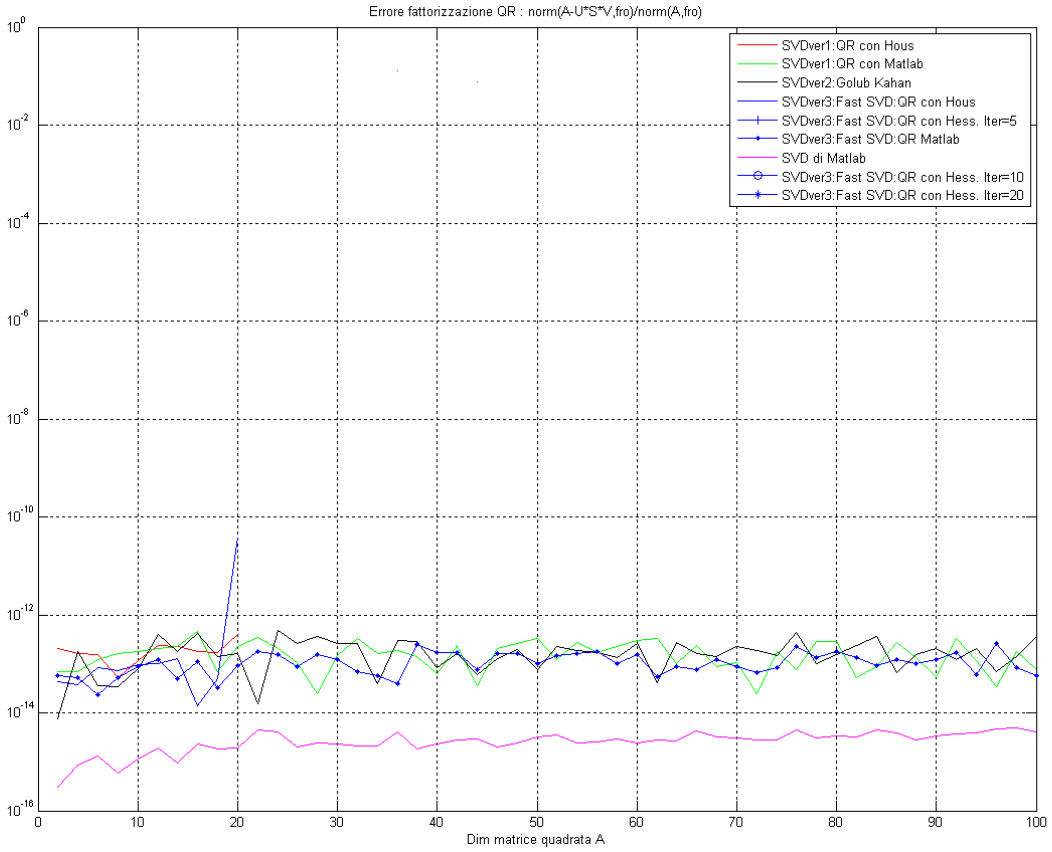


Figura 5-4 - Errore nella fattorizzazione QR

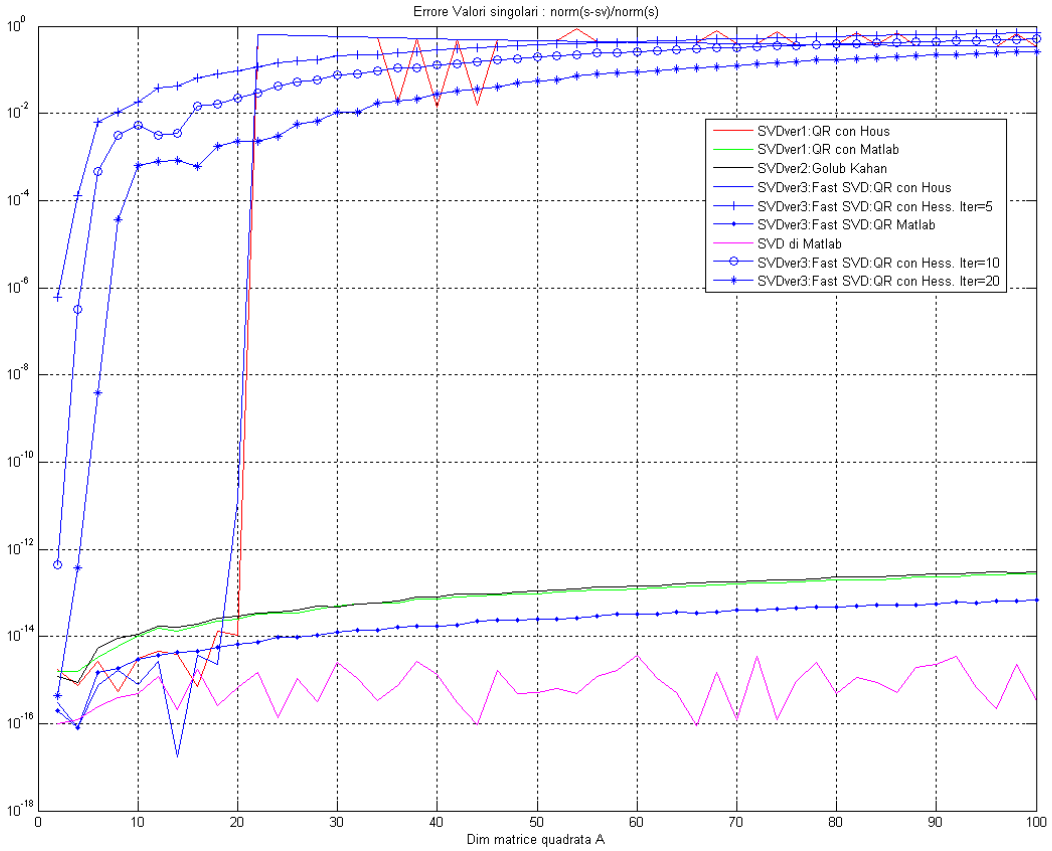


Figura 5-5 - Errori nel calcolo dei valori singolari. E' interessante osservare come la prestazione peggiori significativamente per la ast SVD con QR di Householder e per la SVD_ver1 con QR di Householder superata una certa dimensione.

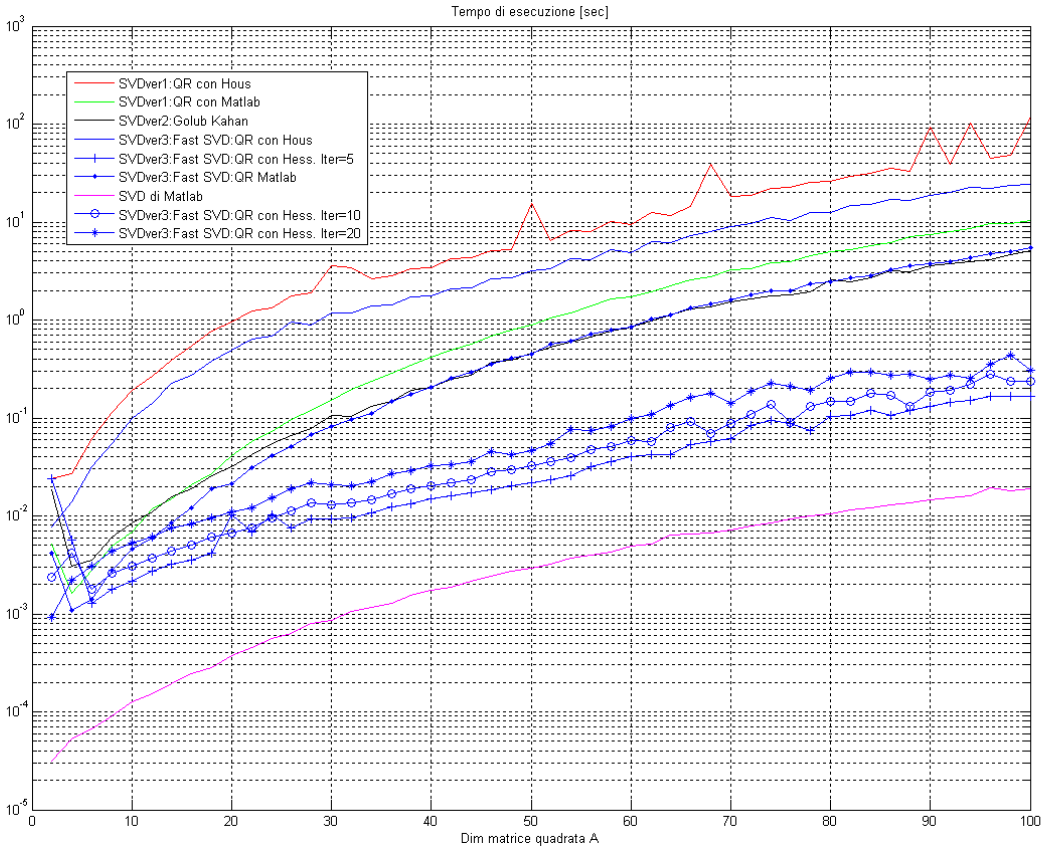


Figura 5-6 - Tempi di esecuzione. Ancora la svd di MATLAB è quella che ha la migliore prestazione.

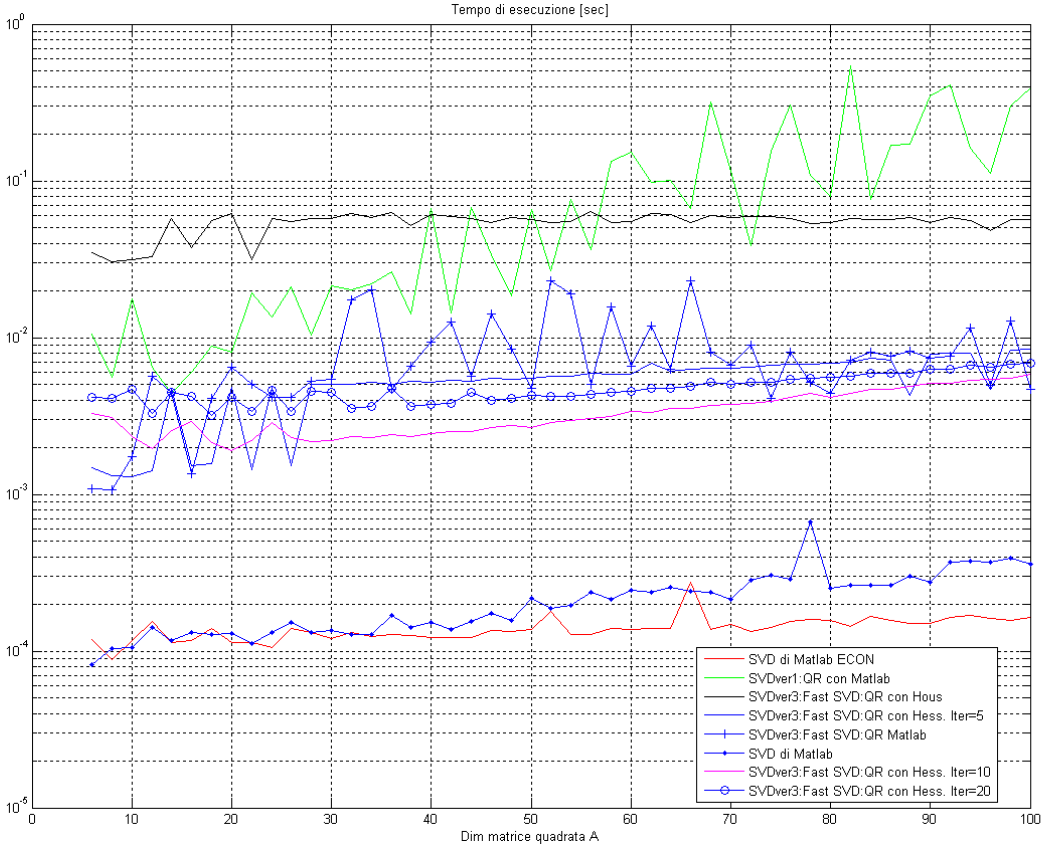


Figura 5-7 - Test di esecuzione su matrici rettangolari avendo fissato $m = 6$ e avendo fatto variare $n = 2{:}2{:}100$. Si noti che gli algoritmi non sono tutti gli stessi di quelli usati prima. Notevole la distinzione tra un calcolo 'intero' e uno 'economico'.

6. Prove con il REGULARIZATION TOOLS

6.1. Condizione discreta di Picard

Tramite la funzione `picard` è possibile controllare graficamente la condizione discreta di Picard. Occorre generare un problema discreto mal posto usando uno dei tanti problemi test disponibili nel pacchetto; uno di questi è `shaw` che è un modello mono-dimensionale di un problema di ricostruzione dell'immagine. Eseguendo `shaw(n)` si ottengono la matrice A e i vettori $\mathbf{b}, \mathbf{x}$ tali che $A\mathbf{x} = \mathbf{b}$. Calcolando il condizionamento della matrice A si ottene:

$$k_2(A) = 5.1241e + 017 \tag{6.1}$$

Obiettivo della regolarizzazione sarà rendere il problema risolubile. Come prima cosa aggiungiamo rumore bianco all'RHS, per produrre un problema più realistico:

$$\mathbf{b} = \bar{\mathbf{b}} + \mathbf{e} \tag{6.2}$$

$$\mathbf{e} = ds\tilde{\mathbf{e}}_{rand} \tag{6.3}$$

Dove ds varia tra 0 e $eps \times 10^{-16}$ con un passo di 10^{-4}. Prima di effettuare l'analisi del problema, calcoliamo la SVD della matrice dei coefficienti che verrà usata in seguito. La funzione `csvd` coincide con quella usata in MATLAB. A questo punto è possibile usare `picard` per rappresentare i valori singolari e i coefficienti di Fourier per il problema perturbato e per quelli imperturbati. I risultati sono mostrati in Figura 6-1.

Test_reg: visualizzazione della condizione discreta di Picard

```
[A,b_bar,x] = shaw(32);
[U,s,V] = csvd(A);
figure(1)
ds = [ 0 eps eps*10^4 eps*10^8 eps*10^12 eps*10^16]
for i = 1 : length(ds)
    randn('seed',41997);
    e(i) = ds(i)*randn(size(b_bar));
    b(i) = b_bar + e(i);
    figure(1)
    picard(U,s,b(i));
    hold on;
end
```

La maggior parte dei coefficienti di Fourier per il problema imperturbato soddisfa la condizione discreta di Picard – sebbene eventualmente, per grandi valori di *i,* sia i valori singolari che i coefficienti di Fourier siano dominati dagli errori di arrotondamento. Per i problemi test 'rumorosi', vediamo che i coefficienti di Fourier per l'RHS diventano dominanti a causa della perturbazione per *i* molto più piccoli di prima. Vediamo anche che nella parte sinistra della curva i coefficienti di Fourier decadono più rapidamente dei valori singolari, indicando che l'RHS soddisfa alla condizione discreta di Picard. Il caso limite è fornito da $ds = eps \times 10^{-16}$ per il quale non ci sono coefficienti di Fourier che soddisfano la condizione discreta di Picard.

Per regolarizzare questo problema dovremmo preferibilmente smorzare i componenti per i quali la perturbazione domina, e lasciare il resto dei componenti (per piccoli *i*) intatti.

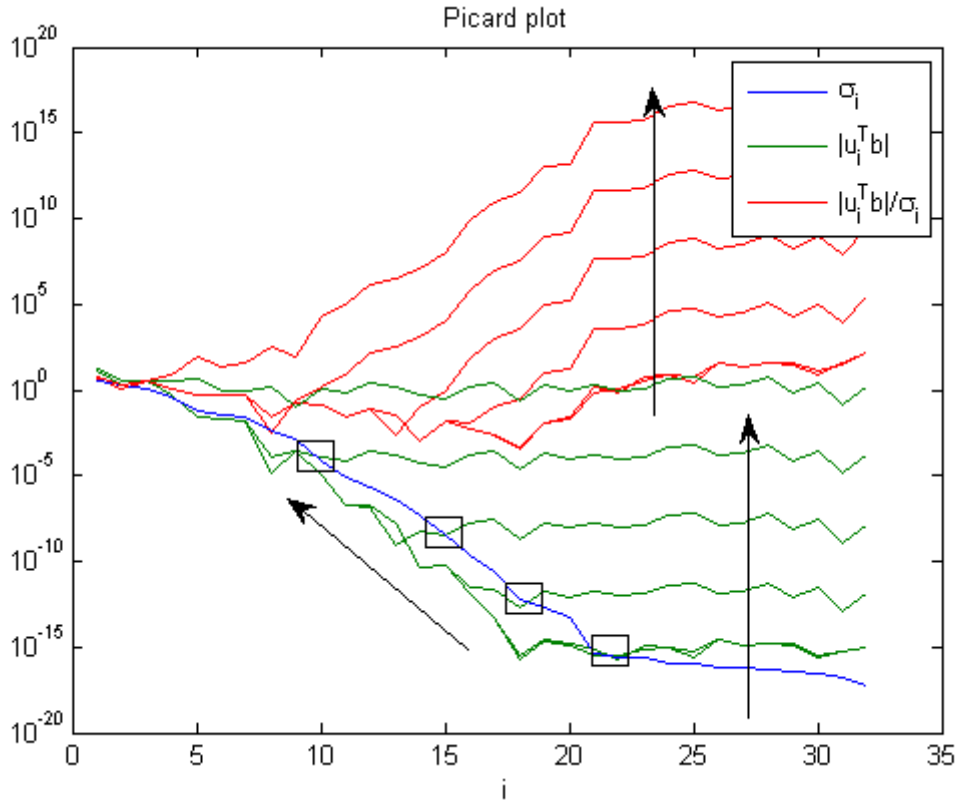


Figura 6-1 – Picard Plot. La prima curva in basso è quella corrispondente all'RHS imperturbata, all'aumentare della perturbazione si ottengono nuove curve nella direzione della freccia. Al crescere dell'errore ci saranno sempre meno coefficienti di Fourier che soddisfano alla condizione discreta di Piacard.

In Figura 6-1 è mostrato l'effetto di una progressiva violazione della condizione di Picard. All'aumentare dell'errore sull'RHS aumentano le oscillazioni della soluzione calcolata tramite:

```
studio = [x A\b(:,1:6)]
surf((studio))
```

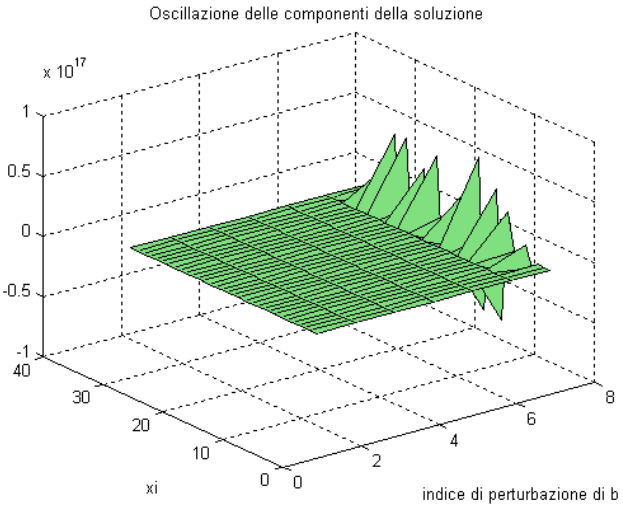


Figura 6-2 – Oscillazione delle componenti della soluzione.

6.2. Fattore di Filtro, soluzioni ed errore

In questa parte consideriamo solo i problemi test 'rumorosi' dell'esempio precedente, calcoliamo le soluzioni regolarizzate tramite il metodo (standard, quindi con $L = I$) di Tikhonov, TSVD e CGLS. Nelle figure In Figura 6-3, Figura 6-4, Figura 6-5 sono mostrati gli andamenti di `rho` (la norma della soluzione) ed `eta` (la norma del residuo). Nelle figure Figura 6-6, Figura 6-7, Figura 6-8 è stato usato il comando `mesh` per rappresentare tutte le soluzioni regolarizzate. Vediamo la tipica situazione per i metodi di regolarizzazione:

- Quando applichiamo molta regolarizzazione la soluzione è *sovraregolarizzata,* cioè troppo *smooth;*
- In seguito si ottiene una migliore approssimazione della soluzione imperturbata;
- Infine la soluzione diventa *irregolarizzata* e iniziano a dominare gli errori di perturbazione, a la sua (semi)norma esplode.

Metodo di Tikhonov

E' stato valutato al variare di λ. In Figura 6-3 in verde è mostrato l'andamento

```
for j = 0 : 20
    lambda(j+1) = 10^-j ;
end
index = 1 : 21  ;

% Diverse curve al variare della perturbazione dell'RHS.
for i = 1 : 6
figure(1);
[X_tikh rho_tikh eta_tikh] = tikhonov(U,s,V,b(:,i),lambda);
F_tikh = fil_fac(s,lambda);
semilogy(index, rho_tikh(index), index, eta_tikh(index)), title('rho, eta'),
xlabel('lambda')
hold on
pause
end

% Diverse curve al
i = 5 ; % eps*10^12= 2.2204e-004
[X_tikh rho_tikh eta_tikh] = tikhonov(U,s,V,b(:,i),lambda);
F_tikh = fil_fac(s,lambda);
subplot(1,2,1);
surf(X_tikh), title('Tikhonov solutions'), xlabel('lambda'), ylabel('x_i')
subplot(1,2,2)
surf(log10(F_tikh)), title('Tikh filter factors, log scale'), xlabel('lambda'),
ylabel('x_i')
```

Metodo di TSVD

E' stato valutato al variare di k

```
i = 5;
k_tsvd = 1 : size(b_bar)
[x_tsvd rho_tsvd eta_tsvd] = tsvd(U,s,V,b(:,i),k_tsvd);
F_tsvd = fil_fac(s,k_tsvd,'tsvd');
subplot(1,2,1);
surf(x_tsvd), title('TSVD solutions'), xlabel('k'), ylabel('x_i')
subplot(1,2,2)
surf(log10(F_tsvd)), title('TSVD filter factors, log scale'), xlabel('k'),
ylabel('x_i')
```

```
for i = 1 : 6
figure(1);
[x_tsvd rho_tsvd eta_tsvd] = tsvd(U,s,V,b(:,i),k_tsvd);
F_tsvd = fil_fac(s,k_tsvd,'tsvd');
semilogy(k_tsvd, rho_tsvd(k_tsvd), k_tsvd, eta_tsvd(k_tsvd)), title('rho, eta,
loglog scale'), xlabel('k')
hold on
pause
end
```

Metodo di CGLS

E' stato valutato al variare di k

```
k = 40
reorth = 1 ;
i = 5
[X_cgls,rho_cgls,eta_cgls,F] = cgls(A,b(:,i),k,reorth,s) ;
index_cgls = 1 : k
surf(X_cgls), title('GCLS solutions'), xlabel('k'), ylabel('x_i')

for i = 1 : 6
    [X_cgls,rho_cgls,eta_cgls,F] = cgls(A,b(:,i),k,reorth,s) ;
figure(1);
semilogy(index_cgls, rho_cgls(index_cgls), index_cgls, eta_cgls(index_cgls)),
title('rho, eta, loglog scale'), xlabel('k')
hold on
pause
end
```

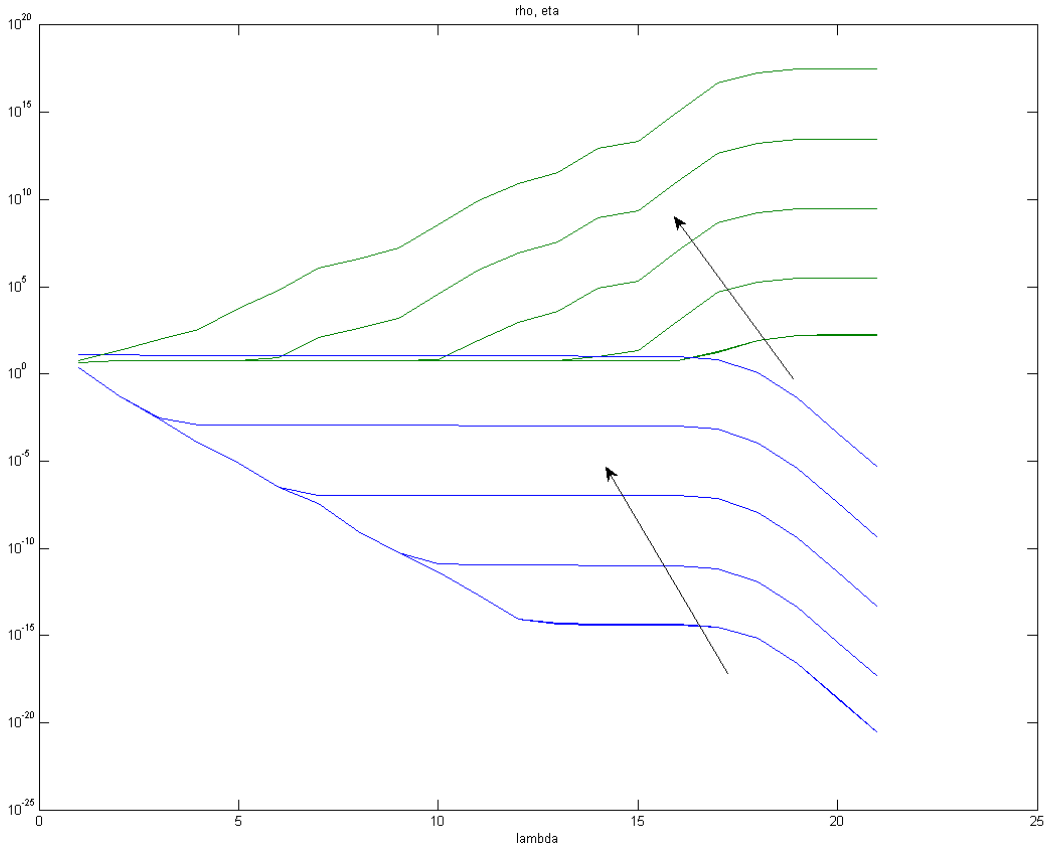


Figura 6-3 – Tikhonov – in verde eta, in blu rho. Le frecce indicano la direzione della curva al crescere dell'errore.

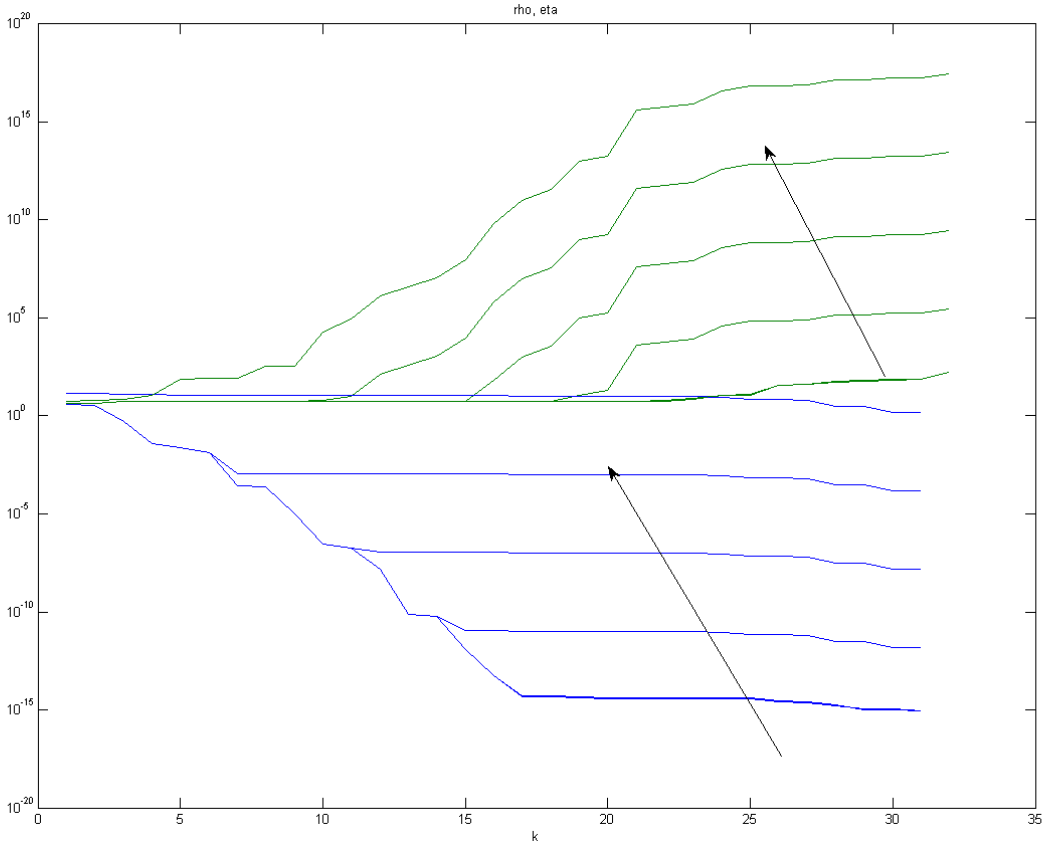


Figura 6-4 - TSVD

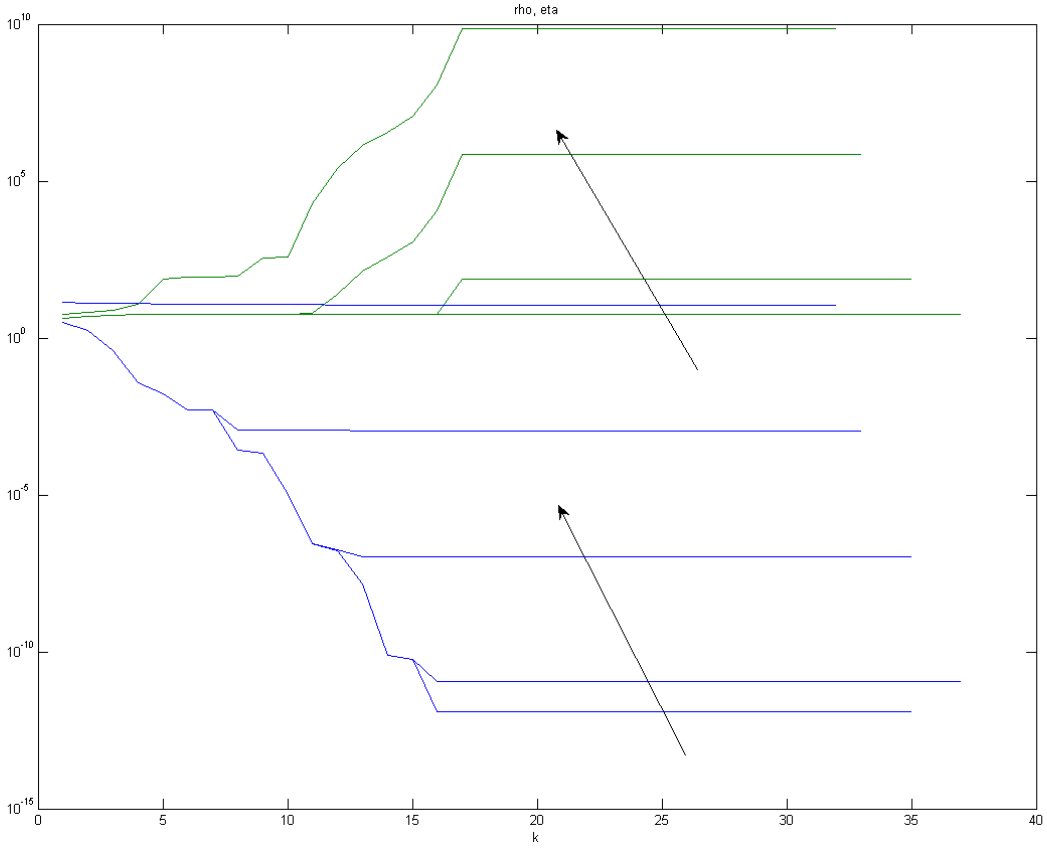


Figura 6-5 - CGLS

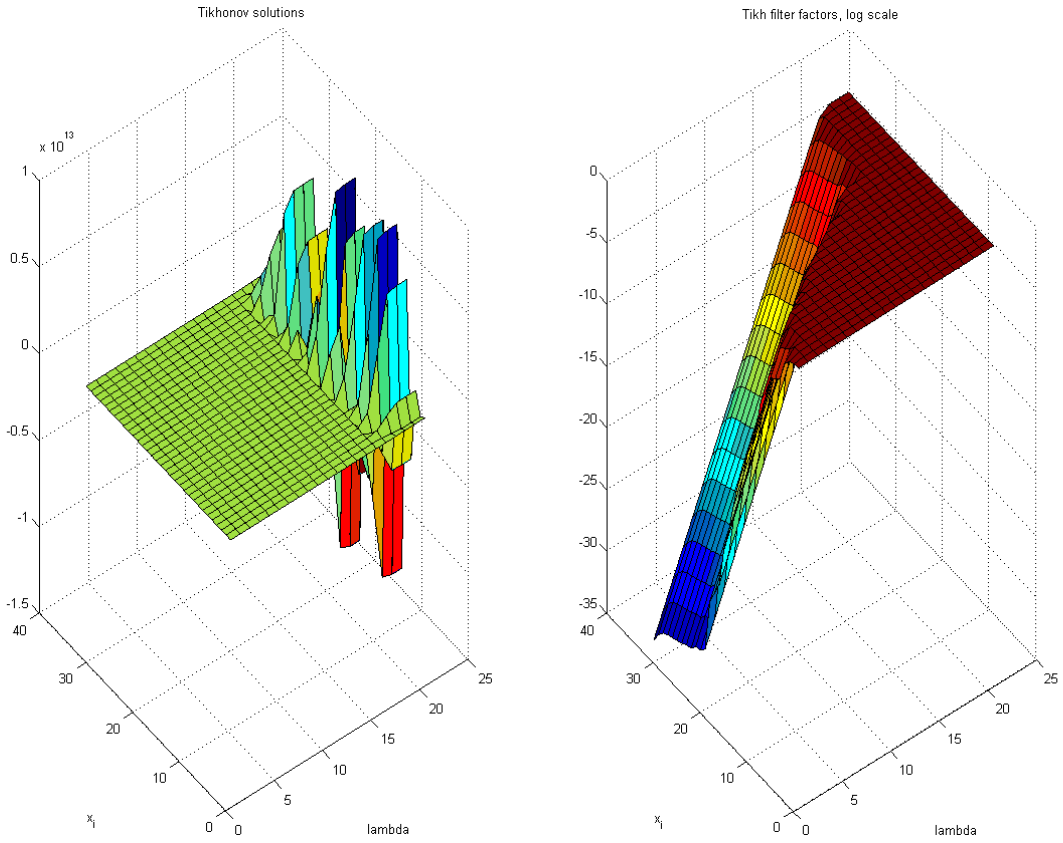


Figura 6-6 – Soluzione di Tikhonov.

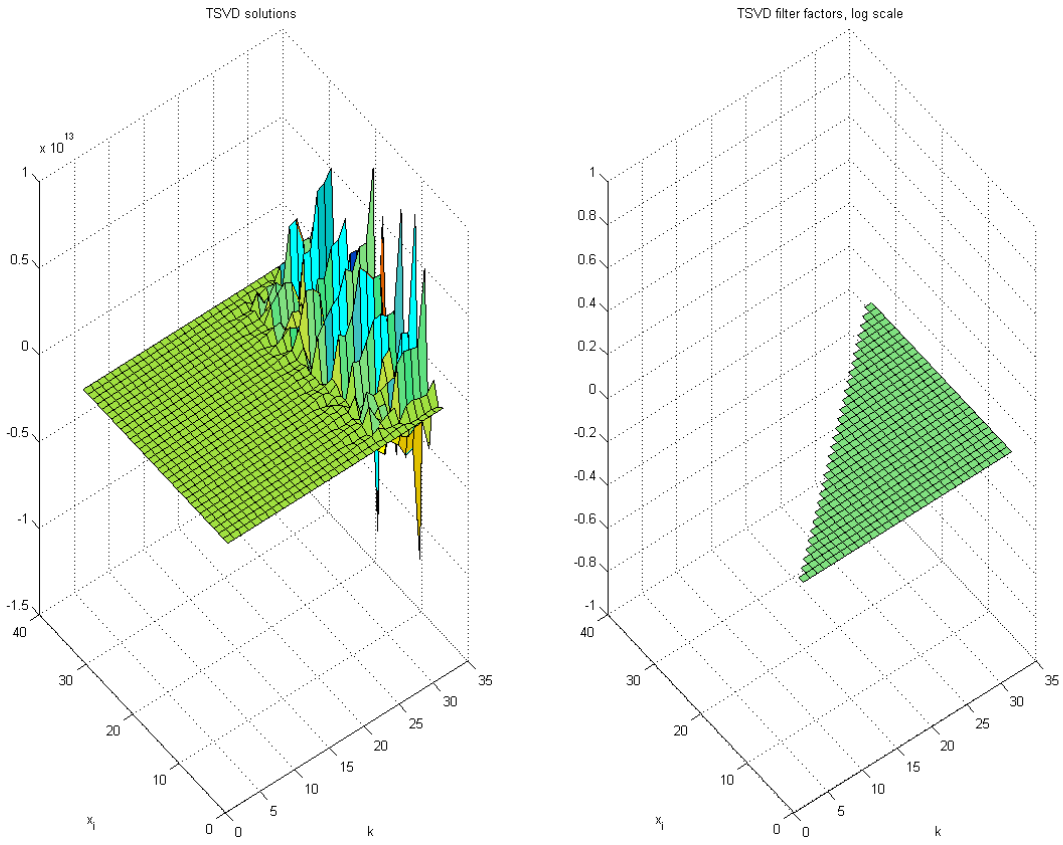


Figura 6-7 – Soluzione TSVD.

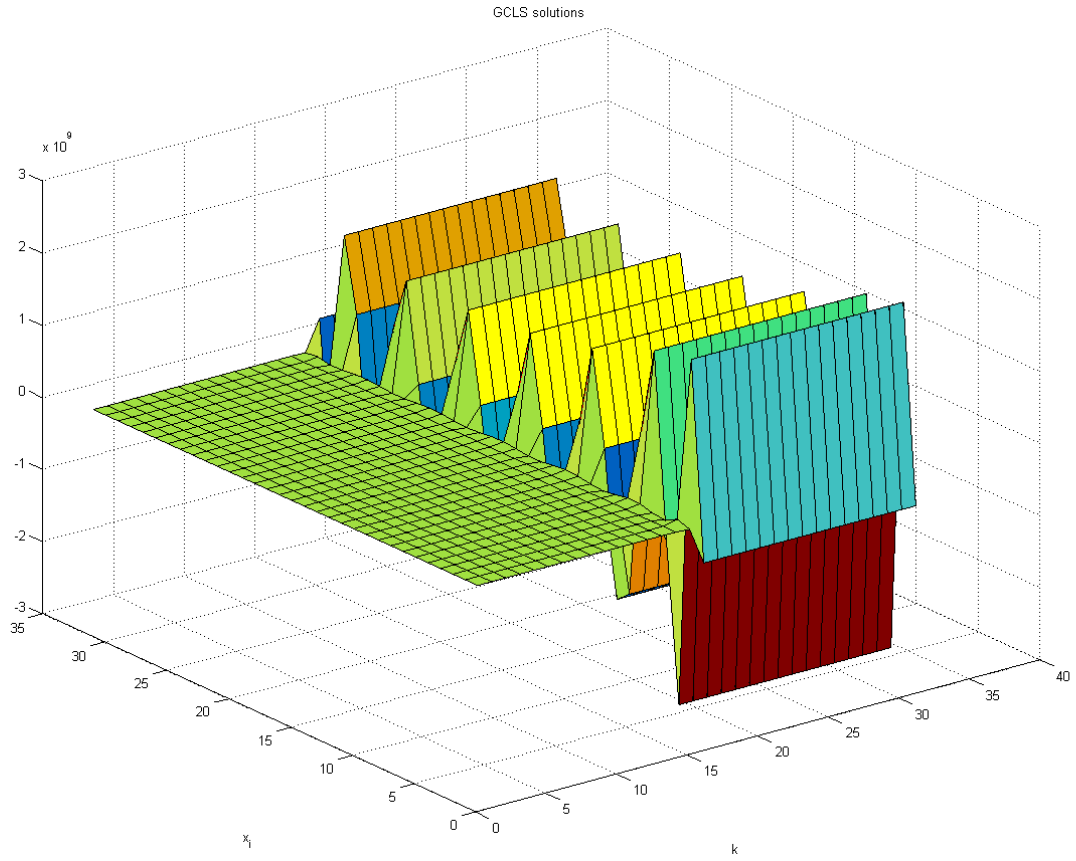


Figura 6-8 – Soluzione GSLS

6.3. La curva L

L'analisi della curva L gioca un ruolo importante nella fase di analisi dei problemi di regolarizzazione. Essa mostra il compromesso tra la minimizzazione di due quantità nel problema di regolarizzazione: la norma del residuo e la (semi)norma della soluzione, e mostra come queste quantità dipendono dal parametro di regolarizzazione. In aggiunta, la curva L può essere usata per il calcolo del parametro di regolarizzazione ottimo come detto. La curva L può essere usata anche per studiare le somiglianze tra diversi metodi di regolarizzazione – se le loro curve L sono prossime, allora le soluzioni regolarizzate sono simili.

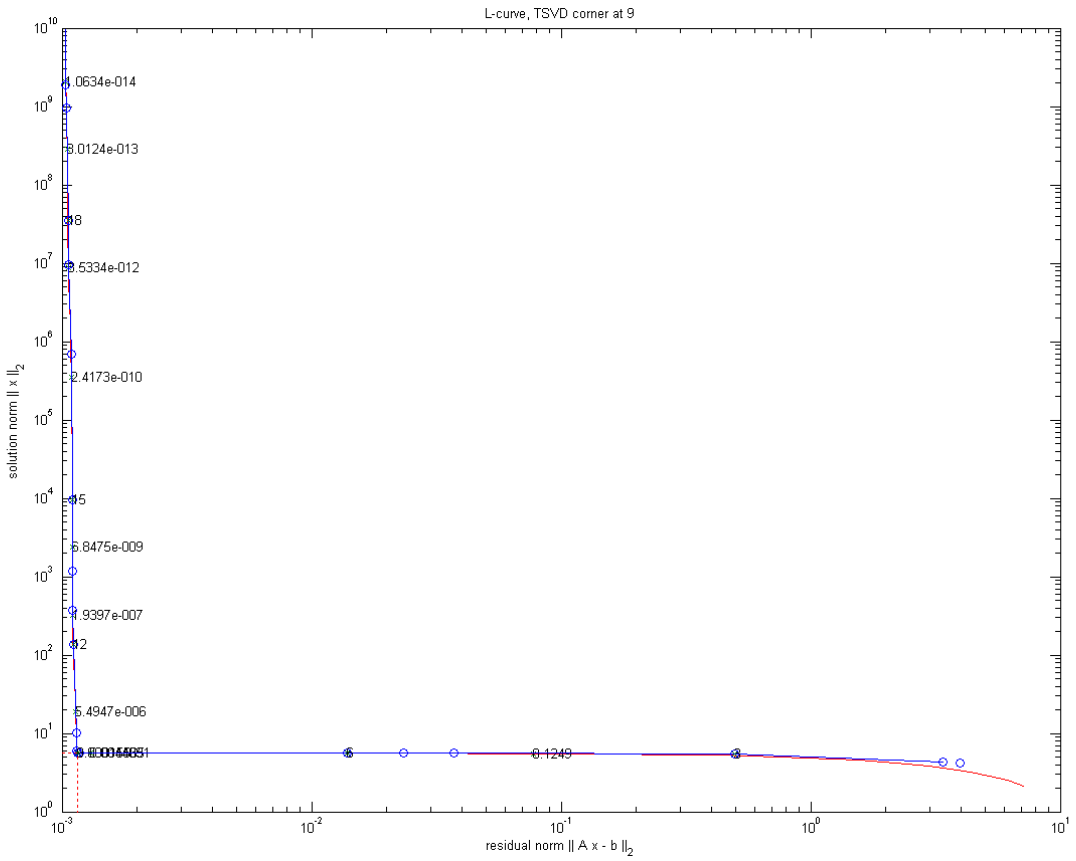


Figura 6-9 – Curva L per Tikhonov (in blu) e per la TSVD (in rosso) corrispondente ad un $ds \sim 10^{-4}$.

In Figura 6-9 sia per la regolarizzazione di Tikhonov che per la TSVD la curva L ha un angolo particolarmente appuntito. Questo angolo corrisponde ad una soluzione regolarizzata in cui l'errore di perturbazione e quello di regolarizzazione sono bilanciati. La somiglianza delle curve L per i due metodi indica la somiglianza tra i metodi.

La funzione `l_curve` fornisce le ascisse e le ordinate per il grafico della Figura 6-9 e il parametro di regolarizzazione relativo all'angolo.

```
for i = 1 : 6
[reg_corner_tikh(i),rho_tikh(:,i),eta_tikh(:,i),reg_param_tikh(:,i)] =
l_curve(U,s,b(:,i),'Tikh' );
[reg_corner_tsvd(i),rho_tsvd(:,i),eta_tsvd(:,i),reg_param_tsvd(:,i)] =
l_curve(U,s,b(:,i),'tsvd' );
end
```

6.4. Parametri di regolarizzazione

La funzione `l_corner` prende in ingresso i valori calcolati con la funzione `l_curve` e fornisce i parametri relativi all'angolo: parametro di regolarizzazione, eta, rho.

```
for i = 1 : 6
[reg_c_tikh(i),rho_c_tikh(i),eta_c_tikh(i)] =
l_corner(rho_tikh(:,i),eta_tikh(:,i),reg_param_tikh(:,i),...
                                    U,s,b(:,i),'Tikh') ;
[reg_c_tsvd(i),rho_c_tsvd(i),eta_c_tsvd(i)] =
l_corner(rho_tsvd(:,i),eta_tsvd(:,i),reg_param_tsvd(:,i),...
```

```
                                        U,s,b(:,i),'tsvd') ;
end
```

Tabella 6.1 – Parametri di regolarizzazione ricavati con la funzione l_corner

ds	reg_c_tikh	reg_c_tsvd	rho_c_tikh	rho_c_tsvd	eta_c_tikh	eta_c_tsvd
0	1.1374e-014	20	4.2360e-015	4.2417e-015	5.6468	5.6468
2.2204e-016	1.1374e-014	21	3.8148e-015	3.8103e-015	5.6468	5.6881
2.2204e-012	3.0492e-012	18	1.0573e-011	1.0572e-011	5.6477	5.6576
2.2204e-008	1.8193e-008	5	1.1000e-007	0.0235	5.6473	5.5854
2.2204e-004	2.9601e-004	9	0.0012	0.0012	5.6451	5.6448
2.2204	0.9941	2	13.0869	13.4923	5.8077	6.0252

In alternativa si può usare il metodo GCV per il calcolo dei parametri di regolarizzazione 'ottimi' per i due metodi: regolarizzazione di Tikhonov e TSVD. È opportuno effettuare un confronto tra l_curve e gcv.
La funzione `gcv` fornisce un grafico della funzione GCV e a seconda del metodo di regolarizzazione impiegato fornisce il parametro di regolarizzazione che corrisponde al minimo di tale curva.
I grafici che seguono sono relativi a $ds \sim 10^{-4}$ cioè $i = 5$.

```
for i = 5 : 5
lambda_l(:,i) = l_curve(U,s,b(:,i));    axis([1e-3,1,1,1e3]);
k_l(:,i) = l_curve(U,s,b(:,i),'tsvd'); axis([1e-3,1,1,1e3]);
lambda_gcv(:,i) = gcv(U,s,b(:,i));      axis([1e-6,1,1e-9,1e-1]);
k_gcv(:,i) = gcv(U,s,b(:,i),'tsvd');    axis([0,20,1e-9,1e-1]);
x_tikh_l(:,i)    = tikhonov(U,s,V,b(:,i),lambda_l(:,i));
x_tikh_gcv(:,i) = tikhonov(U,s,V,b(:,i),lambda_gcv(:,i));
if isnan(k_l(:,i))
else
  x_tsvd_l(:,i) = tsvd(U,s,V,b(:,i),k_l(:,i));
end
x_tsvd_gcv(:,i) = tsvd(U,s,V,b(:,i),k_gcv(:,i));
[norm(x-x_tikh_l(:,i)),norm(x-x_tikh_gcv(:,i)),...
 norm(x-x_tsvd_l(:,i)),norm(x-x_tsvd_gcv(:,i))]/norm(x);
end
```

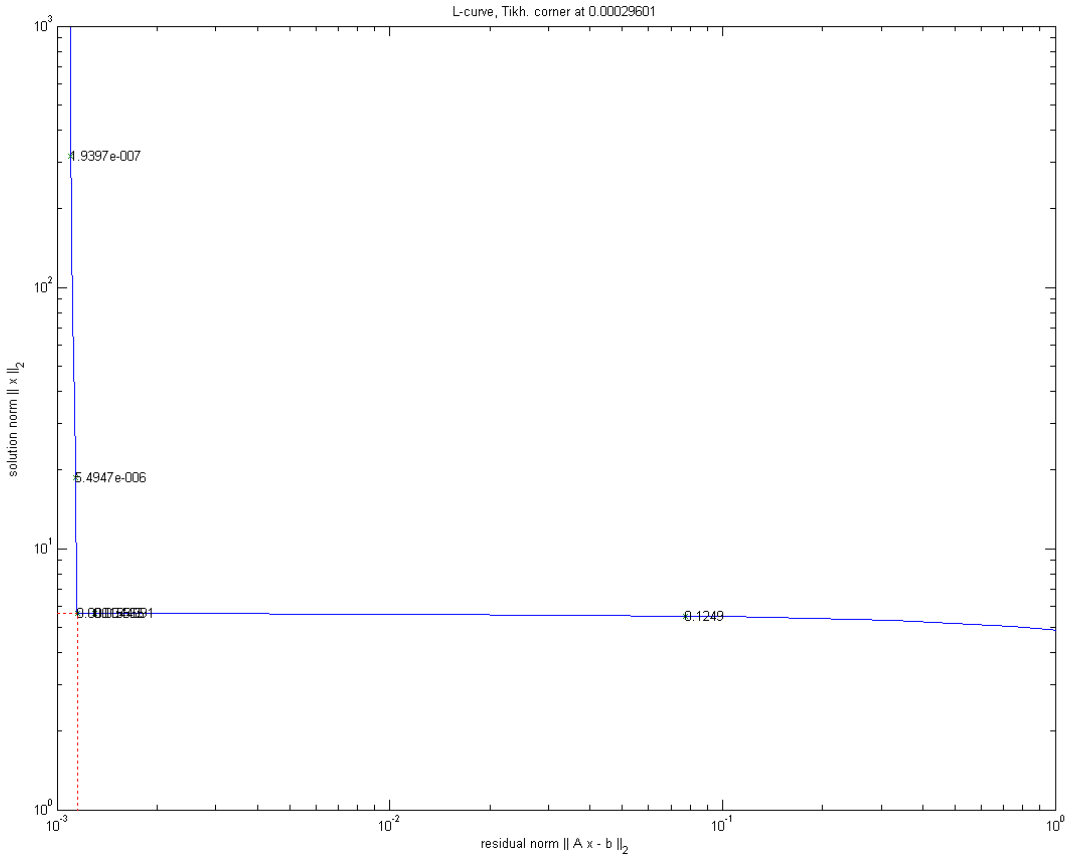


Figura 6-10 – l_curve per Tikhonov, con i=5.

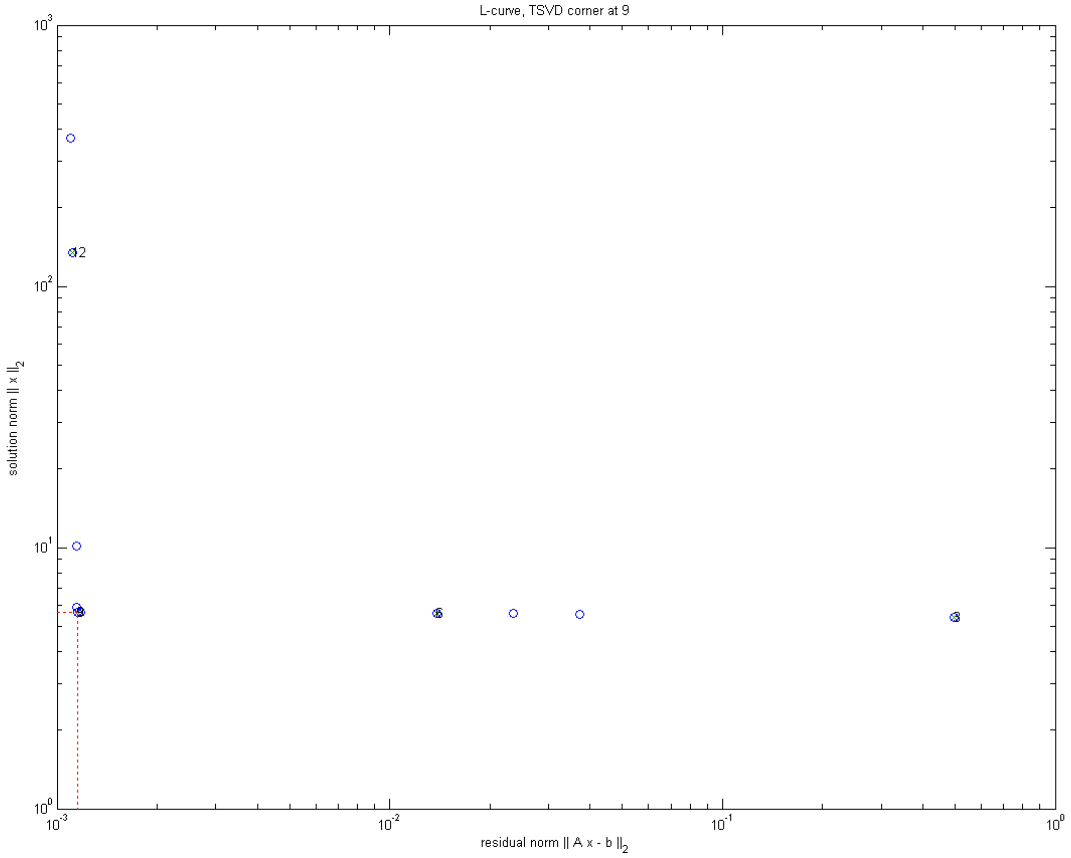


Figura 6-11 – l_curve per TSVD con i = 5

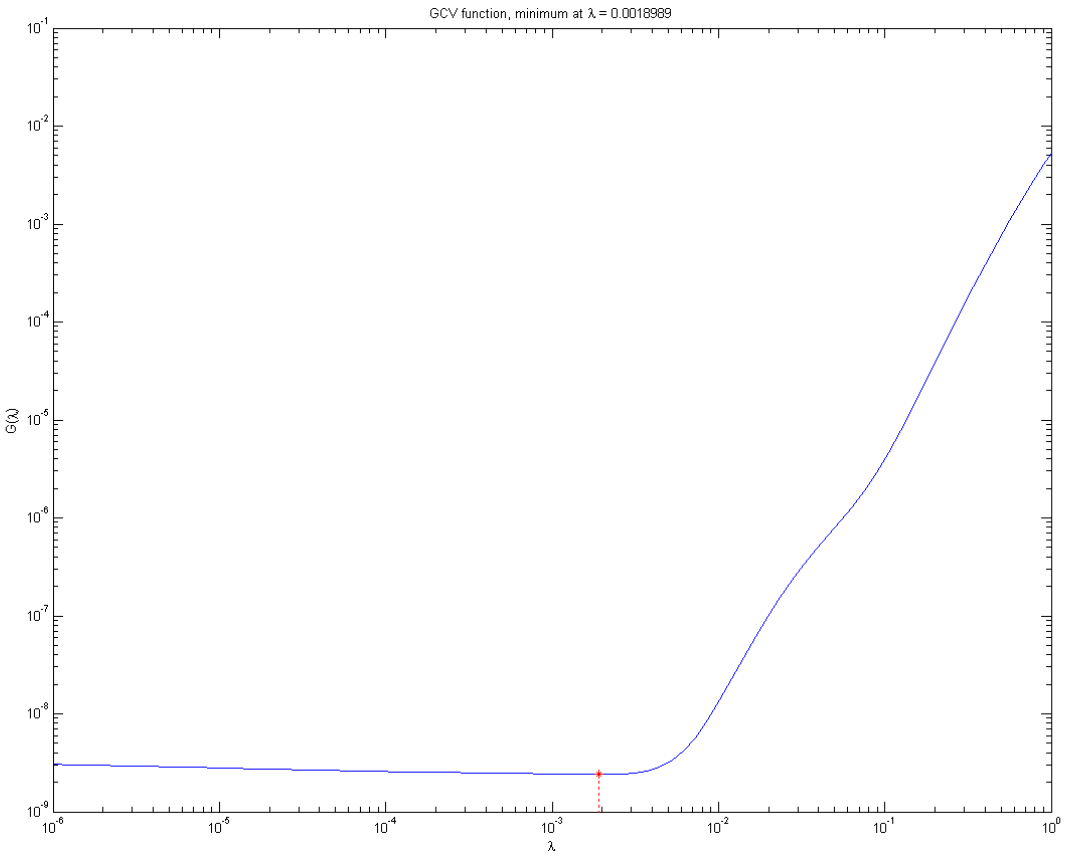


Figura 6-12 – GCV per il metodo di Tikhonov

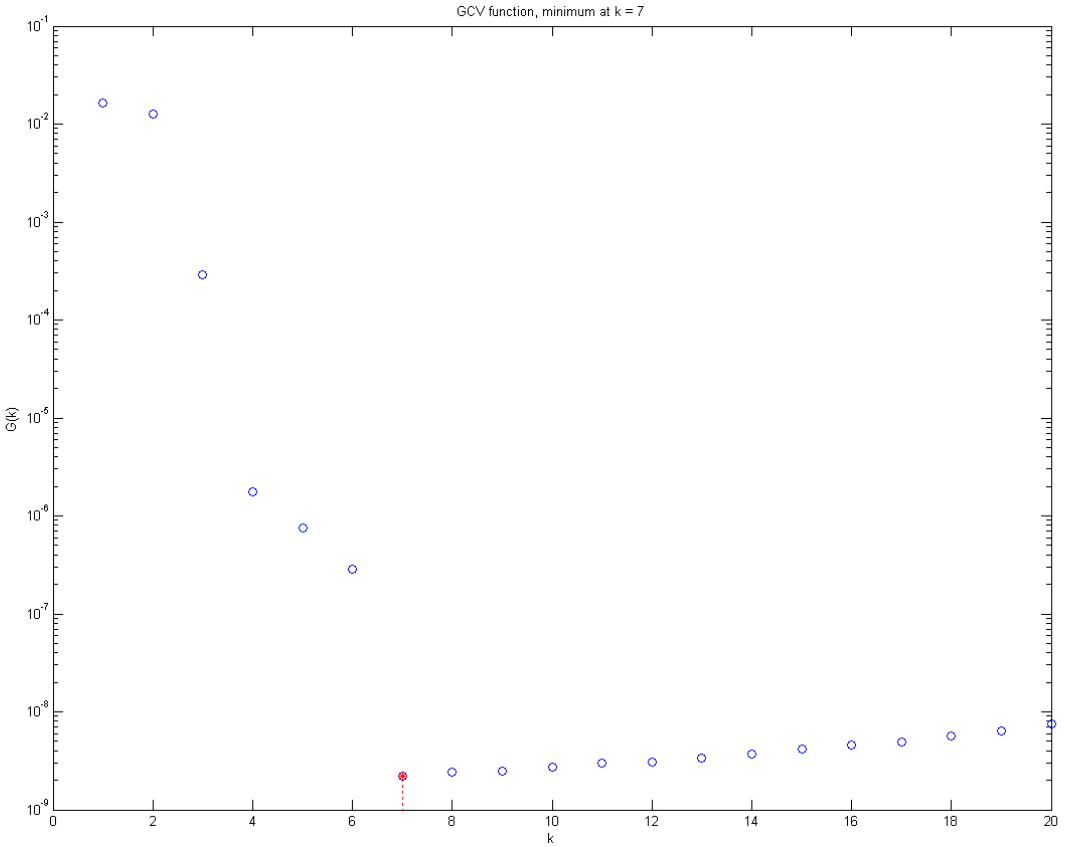


Figura 6-13 – GCV per il metodo TSVD

Seguono i risultati calcolati non solo per $ds \sim 10^{-4}$ ma anche per gli altri valori. Sono messe a confronto le due tecniche: l_curve e gcv.

```
for i = 1 : 6
[norm(x-x_tikh_l(:,i)),norm(x-x_tikh_gcv(:,i)),...
 norm(x-x_tsvd_l(:,i)),norm(x-x_tsvd_gcv(:,i))]/norm(x)
end
```

ds	norm(x-x_tikh_l(:,i))/norm(x)	norm(x-x_tikh_gcv(:,i)/norm(x)	norm(x-x_tsvd_l(:,i)/norm(x)	norm(x-x_tsvd_gcv(:,i)/norm(x)
0	0.0053	0.0019	0.0055	0.0003
2.2204e-016	0.0040	0.0003	0.1214	0.0003
2.2204e-012	0.0184	0.0005	0.0617	0.0011
2.2204e-008	0.0203	0.0079	0.1470	0.0076
2.2204e-004	0.0465	0.0404	0.0324	0.0474
2.2204	0.5401	6.0710	0.7090	13.2633

I risultati sono abbastanza soddisfacenti. Infatti se si provasse a calcolare 'normalmente' la soluzione si otterrebbe una soluzione inutile.

```
norm(x-A\b(:,1))/norm(x)
```

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.518416e-018.

ans =

12.1634

7. Appendice

7.1. Norma e seminorma

Una norma è una funzione che assegna ad ogni vettore di uno spazio vettoriale, tranne lo zero, una lunghezza positiva. Una seminorma può assegnare la lunghezza zero anche ad un vettore diverso da zero.

Definizione. Una norma su uno spazio vettoriale reale o complesso X su K è una funzione:

$$\|\cdot\|: \begin{matrix} X \to [0,\infty) \\ x \longmapsto \|x\| \end{matrix} \tag{7.1}$$

Che verifica le seguenti condizioni:

1. $\|x\| = 0 \iff x = 0$ (funzione definita positiva);
2. $\|\lambda x\| = |\lambda|\|x\| \quad \forall \lambda \in K$ (omogeneità);
3. $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in X$ (disuguaglianza triangolare).

La coppia $(X, \|\cdot\|)$ è detta spazio normato. Una funzione che verifichi solo la seconda e la terza condizione è detta seminorma.

7.2. null space

Il *null space* (o *nullspace*) è di un operatore A è un insieme di tutti gli operandi $\mathbf{v}$ soluzione dell'equazione $A\mathbf{v} = \mathbf{0}$. Esso è detto anche *kernel* di A:

$$\mathrm{Null(A)} = \{\mathbf{v} \in V : A\mathbf{v} = \mathbf{0}\} \tag{7.2}$$

Ovviamente questo *kernel* non c'entra nulla con i *kernel integrali* come quello dell'equazione di Fredholm o della trasformata di Fourier. Se l'operatore è un operatore lineare in uno spazio vettoriale, allora $\mathrm{Null(A)}$ è uno spazio lineare.
Se A è una matrice, $\mathrm{Null(A)}$è un sottospazio lineare dello spazio di tutti i vettori. La dimensione del sottospazio è detta *nullità* di A. Tale sottospazio può essere calcolato come il numero di colonne che non contengono pivot nella *row echelon form* della matrice A. Il teorema *rango-nullità* afferma che il rango di qualsiasi matrice più la sua nullità equivale al numero di colonne della matrice.
I vettori singolari di destra di A corrispondenti ai valori singolari nulli formano una base per $\mathrm{Null(A)}$. Il $\mathrm{Null(A)}$ può essere usato per trovare tutte le soluzioni (la soluzione completa) dell'equazione $A\mathbf{x} = \mathbf{b}$. Se $\mathbf{x}_{\mathrm{part}}$ risolve questa equazione è detta soluzione particolare. La soluzione completa dell'equazione è uguale alla soluzione particolare aggiunta ad un qualsiasi vettore di $\mathrm{Null(A)}$. La soluzione particolare dipende da $\mathbf{b}$ ma non $\mathrm{Null(A)}$.
Se $\det A \neq 0$, cioè se A è un isomorfismo, allora $\mathrm{Null(A)} = \{\mathbf{0}\}$. In modo equivalente se $\mathrm{Null(A)}$ è non nullo, A rappresenta una mappatura che non è uno a uno.

Bibliografia

1. **Quarteroni A., Sacco R., Saleri F.** *Numerical Mathematics.* New Yourk : Springer-Verlang, 2000.
2. **Rodriguez, G.** *Dispense di Calcolo Numerico 2.* Cagliari : s.n., 2007.
3. **Godfrey, Paul.** Simple SVD . *MATLAB file exchange.* [Online] 2006. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=12674.
4. —. Bidiag. [Online] 2003. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3568&objectType=file.
5. **Strimmer, Korbinian.** http://strimmerlab.org/software/corpcor/html/fast.svd.html. [Online]
6. **Neumaier, A.** *Solving ill-conditioned and singular linear systems: A tutorial on regularization.* s.l. : SIAM Review 40, 1198. 636-666.
7. **Hansen, C.** *Regularization Tools, A Matlab Package for Analysis and Solution of Discrete Ill-Posed Problems, V.3.1.* Lyngby, Denmark : Dept. of Mathematical Modelling, Technical University of Denmark, 2001.
8. **Qiao, Sanzheng.** Fast SVD of Hankel/Toeplitz and . [Online] McMaster University, Hamilton, Ontario L8S 4K1, Canada, 9 May 2007. http://www.cas.mcmaster.ca/~qiao/software/takagi/matlab/readme.html.
9. *The efficient generation of random orthogonal.* **G.W.Stewart.** 3, s.l. : SIAM J. Numer. Anal., June 1980, Vol. 17, pp.403-409.
10. **Appell J., Wurzburg, Conti G.** Alcune osservazioni sul rango numerico per operatori non lineari. *Mathematica Bohemica.* 24 Novembre 1998.
11. **Abdi, H.** Singular Value Decomposition (SVD) and Generalized Singular Value Decomposition (GSVD). *Encyclopedia of Measurement and Statistics.* 2007.
12. **Demmel, J. and Kahan, W.** Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy. *SIAM J. Sci. Statist. Comput.* 1990, Vol. 11 (5), 873-912.
13. **Golub, G. H. and Van Loan, C. F.** *Matrix Computations 3rd ed.* Baltimore. : Johns Hopkins University Press, 1996. ISBN 0-8018-5414-8.
14. **Eckart, C., & Young, G.** *The approximation of one matrix by another of lower rank.* s.l. : Psychometrika, 1936.
15. **Halldor, Bjornsson and Venegas, Silvia A.** *"A manual for EOF and SVD analyses of climate data".* Montréal, Québec : McGill University, 1997.
16. *The truncated SVD as a method for regularization.* **Hansen, P. C.** 1987, Vol. BIT, 27, 534-553. .
17. **VA.** http://en.wikipedia.org/wiki/Singular_value_decomposition . [Online]
18. **Ben-Israel, Adi, Thomas N.E. Greville.** *Generalized Inverses.* s.l. : Springer-Verlag., 2003. ISBN 0-387-00293-6. .
19. **Moore, E. H.** On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society.* 1920, Vol. 26: 394-395. .
20. **Penrose, Roger.** A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society.* 1955, Vol. 51: 406-413. .
21. http://en.wikipedia.org/wiki/Pseudoinverse. [Online]

Indice

N

P

Q

R

S

T

U

V