Algoritmi iterativi per la risoluzione di sistemi lineari

Esame di Calcolo numerico 2

Simone Locci 33517

4 luglio 2005

Indice

Introduzione

Lo scopo di questa tesina è analizzare gli algoritmi iterativi per la risoluzione dei sistemi lineari di Jacobi, Gauss-Seidel e del gradiente coniugato (con e senza precondizionamento), studiandone le prestazioni al variare della dimensione della matrice, della sua sparsità e del suo condizionamento. Per problemi malcondizionati, inoltre, si valuterà l'effetto di rumore addittivo gaussiano alla soluzione. Infine si confronteranno le proprietà di regolarizzazione dei metodi iterativi rispetto a metodi basati sull'uso della decomposizione a valori singolari, con particolare attenzione al metodo di Tikhonov ed agli algoritmi L-Curve e GCV.

1 Algoritmi iterativi

Gli algoritmi iterativi che sono stati utilizzati, mediante implementazione con Matlab, sono i seguenti:

- Jacobi;
- Gauss-Seidel;
- gradiente coniugato;
- gradiente coniugato precondizionato.

A meno che non sia indicato differentemente, il sistema di riferimento considerato sarà il seguente:

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \tag{1}$$

Al fine di massimizzare le prestazioni si è evitato, per quanto possibile, l'uso di strutture cicliche, in particolare di tipo `for`, le quali abbattono drasticamente le prestazioni di un linguaggio interpretato quale è Matlab.

1.1 Metodo di Jacobi e Metodo di Gauss-Seidel

Tali metodi sono costruiti a partire dallo splitting additivo della matrice $\boldsymbol{A}$ del sistema (1):

$$\boldsymbol{A} = \boldsymbol{P} - \boldsymbol{N}$$

ovvero

$$\boldsymbol{P}\mathbf{x} = \boldsymbol{N}\mathbf{x} + \mathbf{b}$$

il quale conduce alla definizione del metodo iterativo

$$\boldsymbol{P}\mathbf{x}^{(k+1)} = \boldsymbol{N}\mathbf{x}^{(k)} + \mathbf{b} \tag{2}$$

Si consideri ora lo splitting addittivo:

$$\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{E} - \boldsymbol{F}$$

dove:

$$D_{ij} = \begin{cases} a_{ii} & i = j \\ 0 & i \neq j \end{cases}, \quad E_{ij} = \begin{cases} -a_{ij} & i > j \\ 0 & i \leq j \end{cases}, \quad F_{ij} = \begin{cases} -a_{ij} & i < j \\ 0 & i \geq j \end{cases}$$

Il metodo di Jacobi corrisponde alla scelta:

$$\boldsymbol{P} = \boldsymbol{D}, \quad \boldsymbol{N} = \boldsymbol{E} + \boldsymbol{F}$$

mentre per il metodo di Gauss-Seidel:

$$\boldsymbol{P} = \boldsymbol{D} - \boldsymbol{E}, \quad \boldsymbol{N} = \boldsymbol{F}$$

Di conseguenza il metodo di Jacobi porta alla risoluzione, alla k-esima iterazione, del seguente sistema, derivato dal (2):

$$\boldsymbol{D}\mathbf{x}^{(k+1)} = \mathbf{b} + \boldsymbol{E}\mathbf{x}^{(k)} + \boldsymbol{F}\mathbf{x}^{(k)}$$

Invece il metodo di Gauss-Seidel richiede la risoluzione di:

$$(\boldsymbol{D} - \boldsymbol{E})\,\mathbf{x}^{(k+1)} = \mathbf{b} + \boldsymbol{F}\mathbf{x}^{(k)}$$

Per quanto riguarda la convergenza, sono importanti i seguenti risultati:

- se la matrice $\boldsymbol{A}$ del sistema è a predominanza diagonale stretta, oppure è irriducibilmente diagonalmente dominante, allora i metodi di Jacobi e Gauss-Seidel convergono;
- se la matrice $\boldsymbol{A}$ del sistema è definita positiva il metodo di Gauss-Seidel converge.

In fase di implementazione, per comodità, è stato sempre scelto di utilizzare il vettore nullo quale vettore iniziale $\mathbf{x}^{(0)}$. Le matrici necessarie sono allocate prima che venga avviato il ciclo `while`, in modo da minimizzare gli interventi del sistema operativo per la gestione della memoria. Per quanto riguarda i criteri di arresto utilizzati, oltre al numero massimo di iterazioni, si è implementato il criterio di Cauchy, per il quale, fissato il parametro τ, si arresta l'esecuzione quando:

$$||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}|| \leq \tau||\mathbf{x}^{(k)}||$$

Di seguito sono riportati i codici sorgenti utilizzati per i test. Per Jacobi:

```
function [x,iter] = jacobi(a,b,tau,nmax)
%Algoritmo di Jacobi

[n,n]=size(a);                          %Determina la dimensione di a
x0 = zeros(n,1);                        %Sceglie un vettore iniziale nullo
iter = 0;                               %Inizializzazione dell'algoritmo

%Errore con criterio di Cauchy
nx = norm(x0); scarto = 10^300;         %inizializziamo anche lo scarto

%Calcolo della matrice P di Jacobi
p = zeros(n,n); enne = zeros(n,n);
p = diag(diag(a));
```

```
enne = p - a;
x = x0;

while scarto > tau*nx & iter < nmax
    iter = iter + 1;
    x0 = x;
    x = p\(enne*x+b);
    nx = norm(x);
    scarto = norm(x-x0);
end
```

Per Gauss-Seidel:

```
function [x,iter] = gauss(a,b,tau,nmax)
%Algoritmo di Gauss-Seidel

[n,n]=size(a);                          %Determina la dimensione di a
x0 = zeros(n,1);                        %Sceglie un vettore iniziale nullo
iter = 0;                               %Inizializzazione dell'algoritmo

%Errore con criterio di Cauchy
nx = norm(x0); scarto = 10^300;         %inizializziamo anche lo scarto

%Calcolo della matrice P di Gauss
p = zeros(n,n); enne = zeros(n,n);
p = triu(a);
enne = p - a;
x = x0;

while scarto > tau*nx & iter < nmax
    iter = iter + 1;
    x0 = x;
    x = p\(enne*x+b);
    nx = norm(x);
    scarto = norm(x-x0);
end
```

1.2 Il gradiente coniugato ed il gradiente coniugato precondizionato

L'algoritmo del gradiente coniugato parte dall'ipotesi che la matrice $\boldsymbol{A}$ sia simmetrica definita positiva. Si consideri la forma quadratica:

$$\phi(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T \boldsymbol{A}\mathbf{y} - \mathbf{y}^T\mathbf{b}$$

Il minimo di tale funzione si ha quando si annulla il suo gradiente:

$$\nabla\phi(\mathbf{y}) = \boldsymbol{A}\mathbf{y} - \mathbf{b} = 0$$

Quindi la minimizzazione di $\phi(\mathbf{y})$ è equivalente alla risoluzione del sistema lineare di interesse. Il metodo iterativo che si vuole impiegare non è stazionario (nel senso che i parametri che lo caratterizzano variano ad ogni iterazione) e, ad ogni iterazione:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$$

partendo da un vettore iniziale $\mathbf{x}^{(0)}$. Il vettore $\mathbf{p}^{(k)}$ fornisce la direzione di decrescita, mentre α_k il passo. Si può dimostrare [1] che, qualunque sia la direzione di discesa, l'α_k che minimizza la funzione obiettivo è dato da:

$$\alpha_k = \frac{\left(\mathbf{p}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{p}^{(k)}\right)^T \boldsymbol{A}\left(\mathbf{p}^{(k)}\right)}$$

dove $\mathbf{r}^{(k)}$ è il residuo alla k-esima iterazione.

Per la scelta della direzione $\mathbf{p}^{(k)}$ sono possibili diverse opzioni: ad esempio, il gradiente semplice utilizza il residuo, che fornisce la direzione di massima discesa, essendo esso l'opposto del gradiente. Sono possibili, tuttavia, scelte più efficaci. Il gradiente coniugato utilizza, alla $(k+1)$-esima iterazione, la direzione di discesa:

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} - \beta_k \mathbf{p}^{(k)}$$

Il parametro β_k (cfr. [1]) è dato da:

$$\beta_k = \frac{\left(\mathbf{p}^{(k)}\right)^T \boldsymbol{A}\mathbf{r}^{(k+1)}}{\left(\mathbf{p}^{(k)}\right)^T \boldsymbol{A}\mathbf{p}^{(k)}}$$

Per questa scelta del parametro β_k è possibile rendere *ottimale*, ad ogni iterazione, il vettore soluzione rispetto ad una direzione dello spazio: il gradiente coniugato, allora, converge alla soluzione esatta in n iterazioni (in aritmetica esatta).

L'algoritmo del gradiente coniugato precondizionato sfrutta il precondizionamento del residuo per accelerare la convergenza. La matrice di precondizionamento $\boldsymbol{P}$ deve essere semplice da invertire e tale che $\boldsymbol{P}^{-1}\boldsymbol{A} \simeq \boldsymbol{I}$. Detto $\mathbf{z}^{(k)}$ il residuo precondizionato, ad ogni iterazione, l'algoritmo dovrà risolvere:

$$\boldsymbol{P}\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$$

L'uso del residuo precondizionato modifica sia il calcolo di β_k che della direzione di discesa $\mathbf{p}^{(k+1)}$:

$$\beta_k = \frac{\left(\mathbf{p}^{(k)}\right)^T \boldsymbol{A}\mathbf{z}^{(k+1)}}{\left(\mathbf{p}^{(k)}\right)^T \boldsymbol{A}\mathbf{p}^{(k)}}$$

$$\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} - \beta_k \mathbf{p}^{(k)}$$

Per l'algoritmo del gradiente coniugato c'è convergenza se la matrice $\boldsymbol{A}$ del sistema è simmetrica e definita positiva. I criteri di stop implementati sono gli stessi degli algoritmi di Jacobi e Gauss-Seidel. Di seguito è riportato il sorgente per l'algoritmo del gradiente coniugato.

```
function [x,iter] = simocg(a,b,tau,nmax)
% Esegue il gradiente coniugato senza precondizionamento

[n,n]=size(a);          %Determina la dimensione di a
x0 = zeros(n,1);        %Sceglie un vettore iniziale nullo

r = b - a*x0;
p = r;
iter = 0;               %Inizializzazione dell'algoritmo

%Errore con criterio di Cauchy
nx = norm(x0); scarto = 10^300; %inizializziamo anche lo scarto
alpha = 0;
beta = 0;
delta = 0;
x = x0;
while scarto > tau*nx & iter < nmax
    iter = iter + 1;
    x0 = x;
    s = a*p;
    delta = p'*s;
    alpha = p'*r/delta;
    x = x + alpha*p;
    r = r - alpha*s;
    beta = s'*r/delta;
    p = r - beta*p;
    nx = norm(x);
    scarto = norm(x-x0);
end
```

2 Analisi comparativa

In questa sezione si vogliono confrontare le prestazioni degli algoritmi proposti al variare della dimensione della matrice, della sua sparsità e del suo condizionamento. Infine ci si focalizzerà sui sistemi malcondizionati, valutando anche gli effetti di rumore addittivo gaussiano alla soluzione.

2.1 Dimensioni della matrice

La generazione della matrice $\boldsymbol{A}$ è il primo passo dei test. La soluzione, per il momento, viene imposta come vettore di componenti unitarie. Per ottenere $\boldsymbol{A}$ (che è una matrice piena) si utilizza una matrice casuale che viene resa simmetrica, definita positiva e diagonalmente dominante. La dominanza cresce al crescere di un parametro kk, che deve essere sempre maggiore di 1. Il codice è il seguente:

```
A = rand(n);
A = A - diag(diag(A));
A = A'*A;
s = A*ones(n,1);
A = A + kk*diag(s);
e = ones(n,1);
b = A*e;
nmax = n;
```

La tolleranza τ è stata imposta, in questo test, pari a 10^{-13}. Per confronto, ogni sistema è stato risolto anche con l'algoritmo di Gauss con pivoting; il parametro è pari a $kk = 1.5$ mentre il numero massimo di iterazioni è pari alla dimensione della matrice. La tabella 1 mostra i

n	t (Gauss)	t (CG)	t (GS)	t (JAC)	iter. (CG)	iter. (GS)	iter. (JAC)
100	0.010	0.010	0.051	0.051	13	17	73
200	0.020	0.010	0.080	0.171	11	17	75
500	0.161	0.090	0.260	0.871	11	17	76
1000	0.931	0.221	0.951	3.255	10	17	76
2000	6.039	0.661	5.508	12.938	9	17	76

Tabella 1: Confronto fra gli algoritmi

tempi impiegati, in secondi, ed il numero di iterazioni necessari per raggiungere la tolleranza fissata. Sono mostrate inoltre le iterazioni necessarie a ciascun metodo per convergere. Si noti che i tempi di calcolo dei metodi iterativi sono inizialmente superiori al metodo di Gauss con pivoting, la cui implementazione ottimizzata è affidata alle routine LAPACK. Quando

la dimensione della matrice inizia a crescere si può riconoscere la dipendenza da n^3 per la complessità computazionale, mentre il gradiente coniugato si limita ad n^2. Si noti inoltre come al gradiente coniugato, in particolare, siano necessarie pochissime iterazioni per raggiungere convergenza, mentre il metodo di Jacobi si conferma il peggiore.

La tabella 2 mostra quanto la soluzione trovata da ciascun algoritmo si discosti da quella vera (in norma 2); in tabella 3 sono mostrati invece gli errori relativi. Si nota che, in generale, l'algoritmo di Gauss con pivoting ed il gradiente coniugato offrono il minor errore.

n	Gauss	CG	G-Seidel	JAC
100	7.0427e-015	9.1139e-015	5.8559e-014	2.7171e-013
200	1.5584e-014	5.4925e-014	1.0756e-013	3.7977e-013
500	3.8254e-014	3.8773e-014	1.9984e-013	6.6146e-013
1000	8.3850e-014	7.9028e-014	3.0028e-013	1.1074e-012
2000	1.4055e-013	1.6651e-013	4.3980e-013	1.7042e-012

Tabella 2: Errore assoluto rispetto alla soluzione vera ($||x - x_n||_2$)

n	Gauss	CG	G-Seidel	JAC
100	7.0427e-016	9.1139e-016	5.8559e-015	2.7171e-014
200	1.1020e-015	3.8838e-015	7.6056e-015	2.6854e-014
500	1.7108e-015	1.7340e-015	8.9372e-015	2.9581e-014
1000	2.6516e-015	2.4991e-015	9.4957e-015	3.5022e-014
2000	3.1428e-015	3.7234e-015	9.8342e-015	3.8107e-014

Tabella 3: Errore relativo in norma euclidea

Considerando ancora il metodo di Jacobi, le sue capacità di convergenza risultano determinate essenzialmente dal valore di kk il quale, se elevato, assicura una maggiore dominanza diagonale. Se, invece, esso viene posto prossimo ad 1, il metodo di Jacobi converge molto lentamente. Ne è prova il seguente test (con $n = 2000$), in cui è stato posto $kk = 1.01$. Il metodo del gradiente coniugato e di Gauss-Seidel convergono con la stessa rapidità dei test precedenti, mentre Jacobi non raggiunge la convergenza nemmeno nel numero massimo di iterazioni consentito. I risultati sono riportati in tabella 4.

2.2 Matrici sparse

In questo paragrafo saranno analizzate le prestazioni degli algoritmi finora visti, assieme ai gradienti coniugati precondizionati con fattorizzazione incompleta LU e di Cholesky, quando la matrice $\boldsymbol{A}$ è sparsa. Per la sua generazione sarà utilizzata la funzione di Matlab `sprandsym`, alla

Metodo	Iterazioni	Tempo (s)	Errore	Errore rel.
Gauss	NA	6.049	1.3431e-013	3.0034e-015
JAC	**2000**	**320.101**	**7.1633e-009**	**1.6018e-010**
G-Seidel	21	5.227	7.036328e-013	1.5734e-014
CG	9	0.661	1.484692e-013	3.3199e-015

Tabella 4: Test con $kk = 1.01$

quale saranno passati, come argomenti, la dimensione della matrice, la percentuale di elementi non nulli, il reciproco del condizionamento. Verrà inoltre passato un quarto parametro, posto a 1, che consente di ottenere una matrice definita positiva.

Le matrici sparse, come quella prodotta da `sprandsym`, vengono allocate da Matlab in maniera ottimizzata, mediante un array che contiene posizione e valore dei soli elementi non nulli: ciò consente un notevole risparmio di memoria. Inoltre le routine di moltiplicazione matrice vettore sono ottimizzate per questo tipo di formato. I test effettuati hanno una densità di elementi non nulli pari al 10% ed un condizionamento delle matrici pari a 2.5. La dimensione della matrice è stata fissata a 1000: in formato sparso questo significa un'occupazione di 2287892bytes, mentre in formato *full* la stessa matrice avrebbe richiesto 8000000bytes. In figura 1 è stato effettuato uno zoom della struttura della matrice, che ne evidenzia la sparsità. Gli stessi test sono stati eseguiti sia con le matrici sparse che convertendo le matrici utilizzate

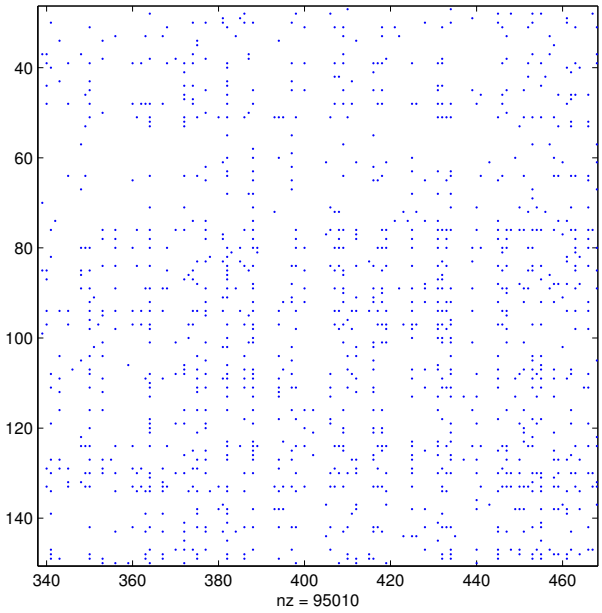

Figura 1: Zoom della struttura della matrice sparsa

in matrici piene. In generale si è notato un rallentamento, tranne che per Gauss con pivoting, le cui prestazioni sono migliori se la matrice viene passata in formato full. I risultati ottenuti sono mostrati in tabella 5. Per quanto riguarda i gradienti coniugati precondizionati, sono

Metodo	Tempo(sp.)	Tempo (full)	Errore	Err. rel.	Iterazioni
Gauss	3.906	0.951	3.3666e-014	1.0646e-015	NA
JAC	0.591	2.163	1.1326e-012	3.5817e-014	49
G-Seidel	0.310	0.901	2.3024e-013	7.2811e-015	17
CG	0.260	0.411	2.2743e-013	7.1922e-015	22
PCG (LU)	0.831	0.972	1.6798e-014	5.3123e-016	8
PCG (Chol.)	0.962	1.031	1.6305e-014	5.1563e-016	8

Tabella 5: Confronto fra gli algoritmi

state utilizzate la fattorizzazione LU incompleta e la fattorizzazione di Cholesky incompleta, entrambe con una tolleranza fissata a 10^{-3}. Al diminuire di tale valore la fattorizzazione diventa sempre più completa, di conseguenza il numero di iterazioni necessario per giungere a convergenza diminuisce, ma aumenta il costo computazionale. In tabella 6 sono mostrati il tempo di elaborazione necessario ed il numero di iterazioni per una tolleranza variabile fra 10^{-6} e 10^{-1}. Per confronto, il gradiente coniugato non precondizionato è comunque più veloce, in questi casi, convergendo in 0.26s in 22 iterazioni.

Toll.	Tempo (s)	Iterazioni
1e-1	0.390	20
1e-2	0.501	12
1e-3	0.791	8
1e-4	1.782	6
1e-5	3.675	6
1e-6	6.299	5

Toll.	Tempo (s)	Iterazioni
1e-1	0.611	20
1e-2	0.641	12
1e-3	0.902	8
1e-4	1.913	6
1e-5	3.876	6
1e-6	6.569	5

Tabella 6: PCG LU (a sinistra) e Chol. (a destra) al variare della tolleranza

2.3 Sistemi malcondizionati

Per lo studio di sistemi malcondizionati iniziamo con l'analisi di matrici sparse di dimensione 200 con sparsità al 10%. Il numero massimo di iterazioni è stato fissato a 10000, la tolleranza a 10^{-13}. Rispetto agli algoritmi utilizzati in precedenza, in questi test saranno calcolati anche i vettori con le norme dei residui e con le norme degli errori rispetto alla soluzione esatta ad ogni

iterazione. Saranno eseguite tre batterie di test, con numeri di condizionamento pari a 10^2, 10^4 e 10^6.

I test hanno portato ai risultati di tabella 7 e 8. Poiché l'algoritmo di Jacobi non è stato mai in grado di giungere a convergenza, è stato omesso dalle tabelle. La tabella 7 mostra le prestazioni degli algoritmi di Gauss con pivoting e di Gauss-Seidel, mentre la tabella 8 confronta l'algoritmo del gradiente coniugato senza precondizionamento e con precondizionamento di tipo Cholesky incompleto. In quest'ultimo caso è netta la diminuzione delle iterazioni necessarie per giungere a convergenza, tuttavia a questo è abbinata una riduzione dei tempi di elaborazione solo per i primi due test, a $\kappa = 10^2$ e $\kappa = 10^4$.

κ	Err. rel. (Gauss)	T (Gauss)	Err rel. (GS)	Iter. (GS)	T (GS)
10^2	1.9151e-015	0.070	1.3809e-012	366	0.350
10^4	5.1112e-014	0.070	4.5328e-006	10000	17.224
10^6	3.0223e-012	0.070	0.4159	10000	15.292

Tabella 7: Test a malcondizionamento crescente - algoritmi di Gauss e Gauss-Seidel

κ	Err. rel. (CG)	Iter. (CG)	T (CG)	Err rel. (CGRR)	Iter. (CGRR)	T (CGRR)
10^2	3.3682e-013	127	0.130	1.8490e-015	10	0.050
10^4	2.1169e-011	721	0.401	6.7345e-014	28	0.080
10^6	8.1198e-010	3562	1.733	5.4604e-011	3164	5.608

Tabella 8: Test a malcondizionamento crescente - gradiente coniugato

Per $\kappa = 10^6$ sono stati tracciati i grafici, in figura 2, 3 e 4, che rappresentano l'andamento della norma dei residui e dell'errore in scala semilogaritmica: è di rilievo come il residuo sia molto più piccolo rispetto all'errore sulla soluzione.

Se ora viene introdotto del rumore nei dati, questo verrà amplificato a causa dell'elevato numero di condizionamento, rendendo inaccettabile la soluzione. Utilizzeremo rumore gaussiano a media nulla e varianza unitaria, di ampiezza pari a 10^{-3}, che sarà sovrapposto al termine noto del sistema. Come soluzione sarà utilizzato il campionamento di una sinusoide di ampiezza unitaria (questo è un caso più vicino ai reali sistemi di interesse). Il condizionamento è stato portato a 10^8. I risultati sono riportati in tabella 9. Il metodo di Jacobi nuovamente non converge, mentre tutti gli altri metodi si assestano su un errore dell'ordine di 10^5. Si rende necessaria una regolarizzazione della soluzione.

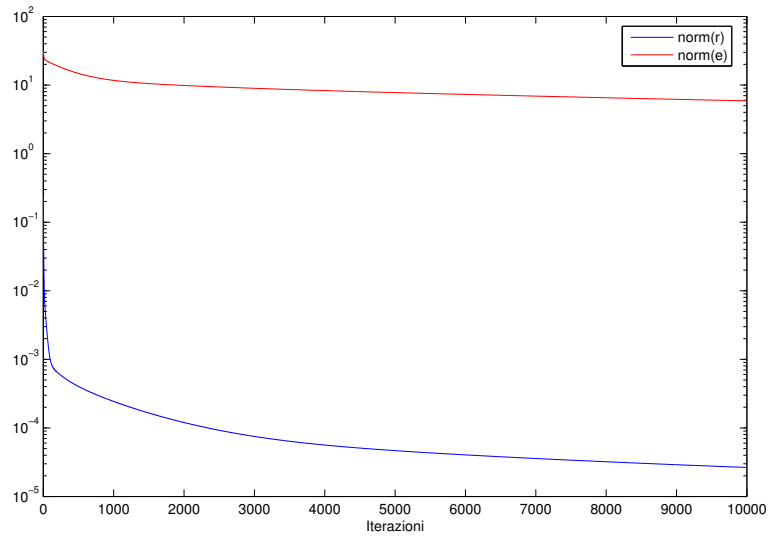

Figura 2: Andamento dei residui e dell'errore per Gauss-Seidel

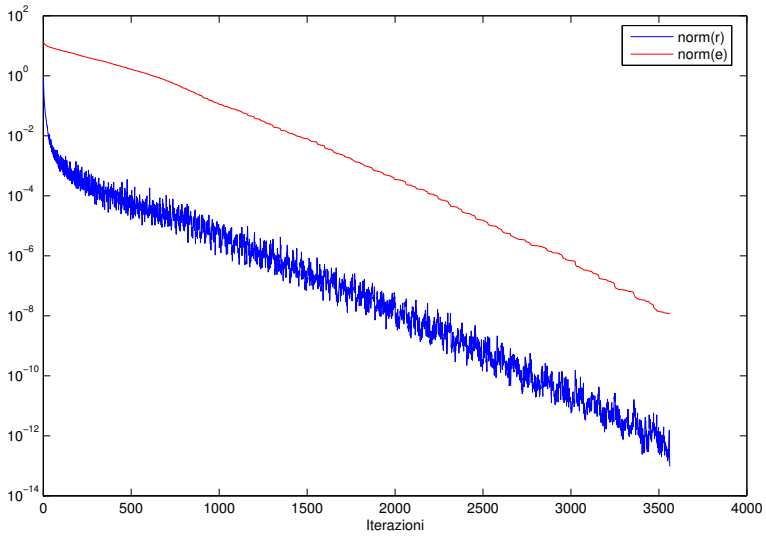

Figura 3: Andamento dei residui e dell'errore per il gradiente coniugato

Metodo	Errore	Err. rel.
Gauss	2.3265e+005	2.3265e+004
G-Seidel	1.3273e+005	1.3273e+004
CG	2.3265e+005	2.3265e+005

Tabella 9: Confronto fra gli algoritmi per $\kappa = 10^8$

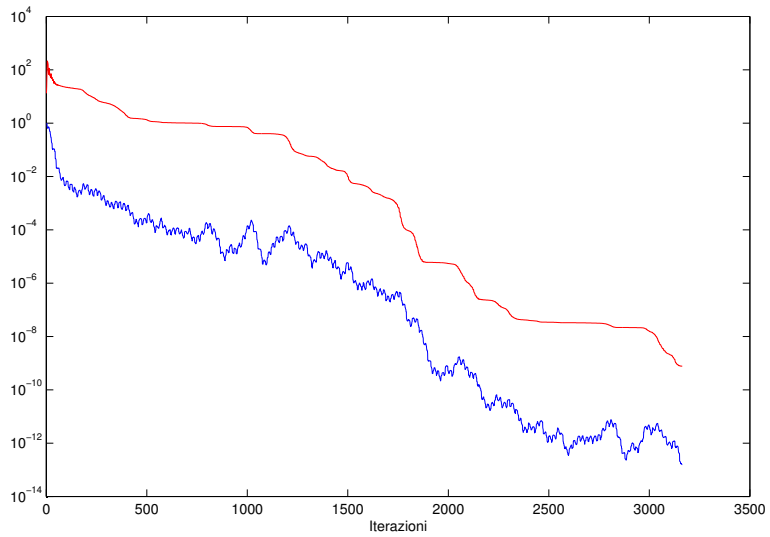

Figura 4: Andamento dei residui e dell'errore per per il gradiente coniugato precondizionato

3 Regolarizzazione

Quando si deve affrontare la risoluzione di sistemi malcondizionati è necessario ricorrere a tecniche di regolarizzazione. Si vorrebbe, innanzitutto, che la soluzione $\mathbf{x}$ soddisfacesse al seguente problema ai minimi quadrati:

$$\min_{\mathbf{x}} = ||\boldsymbol{A}\mathbf{x} - \mathbf{b}||_2$$

Inoltre si vorrebbe ottenere una soluzione la cui norma fosse la minore possibile. Ciò significa aggiungere il vincolo di minimizzazione della quantità

$$\Omega(\mathbf{x}) = ||\boldsymbol{L}(\mathbf{x} - \mathbf{x}^*)||_2$$

dove $\mathbf{x}^*$ è una stima iniziale della soluzione e $\boldsymbol{L}$ può essere, ad esempio, una matrice che approssima un operatore di derivazione o una matrice identità. Sarà il metodo di regolarizzazione a determinare come soddisfare queste due condizioni. Una condizione, tuttavia, deve essere comunque soddisfatta: è la condizione di Picard discreta. Per introdurla dobbiamo ricorrere alla SVD (Singular Value Decomposition), ma si può estendere la definizione alla GSVD (Generalized SVD). La SVD di una matrice $\boldsymbol{A} \in R^{m \times n}$ $(m \geq n)$ è:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \sum_{i=1}^{n} \mathbf{u}_i \sigma_i \mathbf{v}_i^T$$

dove $\boldsymbol{U}$ e $\boldsymbol{V}$ sono matrici con colonne ortonormali e dove $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n)$ ha elementi diagonali non negativi per cui $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$. I numeri σ sono i valori singolari della

matrice. Si può dimostrare che la soluzione $\mathbf{x}$ è data da:

$$\mathbf{x} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

La condizione di Picard afferma che i coefficienti di Fourier $\mathbf{u}_i^T \mathbf{b}$ in media devono tendere a zero più in fretta dei valori singolari (nel caso della GSVD dei valori singolari generalizzati). Poiché nella realtà questo non avviene, è il metodo di regolarizzazione a dover produrre una soluzione regolarizzata che, nel caso di $\mathbf{x}^* = 0$ e $\boldsymbol{L} = \boldsymbol{I}$), può essere espressa come segue:

$$\mathbf{x}_{\text{reg}} = \sum_{i=1}^{n} f_i \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i$$

I numeri f_i costituiscono i fattori di filtro per il particolare metodo di regolarizzazione utilizzato e fanno sì che i contributi $(\mathbf{u}_i^T \mathbf{b})\mathbf{x}_i$ alla soluzione per i valori piccoli di σ_i siano filtrati.

3.1 I metodi di regolarizzazione

Si è scelto di utilizzare, per la regolarizzazione, il metodo di Tikhonov, che si basa sulla SVD della matrice del sistema, ed il gradiente coniugato, nella sua forma ai minimi quadrati qualora la matrice non si presentasse simmetrica definita positiva.

Il metodo di Tikhonov parte dall'idea di calcolare la soluzione regolarizzata $\mathbf{x}_\lambda$ nel seguente modo:

$$\mathbf{x}_\lambda = \text{argmin} \left\{ ||\boldsymbol{A}\mathbf{x} - \mathbf{b}||_2^2 + \lambda^2 ||\boldsymbol{L}(\mathbf{x} - \mathbf{x}^*)||_2^2 \right\}$$

Si cerca dunque di trovare il miglior compromesso fra norma del residuo e norma della soluzione (nel caso in cui $\boldsymbol{L} = \boldsymbol{I}$ e $\mathbf{x}^* = 0$). La scelta del parametro λ è cruciale. Non avendo conoscenze a priori sulla perturbazione del termine noto si ricorre a metodi euristici: nel nostro caso sono stati considerati la L-curve ed il GCV (Generalized Cross Validation).

La L-curve è un grafico $\log ||\boldsymbol{L}\mathbf{x}||_2$ vs. $\log ||\boldsymbol{A}\mathbf{x} - \mathbf{b}||_2$ come quello riportato in figura 5. Grazie al toolbox [2] per Matlab è possibile determinare λ in modo tale da porsi nel punto di ginocchio della curva.

L'algoritmo GCV si basa su considerazioni statistiche e sceglie il parametro di regolarizzazione minimizzando la funzione GCV:

$$G \equiv \frac{||\boldsymbol{A}\mathbf{x}_{\text{reg}} - \mathbf{b}||_2^2}{\left(\text{traccia}\left(\boldsymbol{I}_m - \boldsymbol{A}\boldsymbol{A}^I\right)\right)^2}$$

dove $\boldsymbol{A}^I$ è una matrice che produce la soluzione regolarizzata quando viene moltiplicata per $\mathbf{b}$, cioè $\mathbf{x}_{\text{reg}} = \boldsymbol{A}^I \mathbf{b}$.

Infine l'algoritmo del gradiente coniugato: anch'esso effettua una regolarizzazione, poiché le componenti a bassa frequenza della soluzione tendono a convergere più in fretta rispetto a quelle

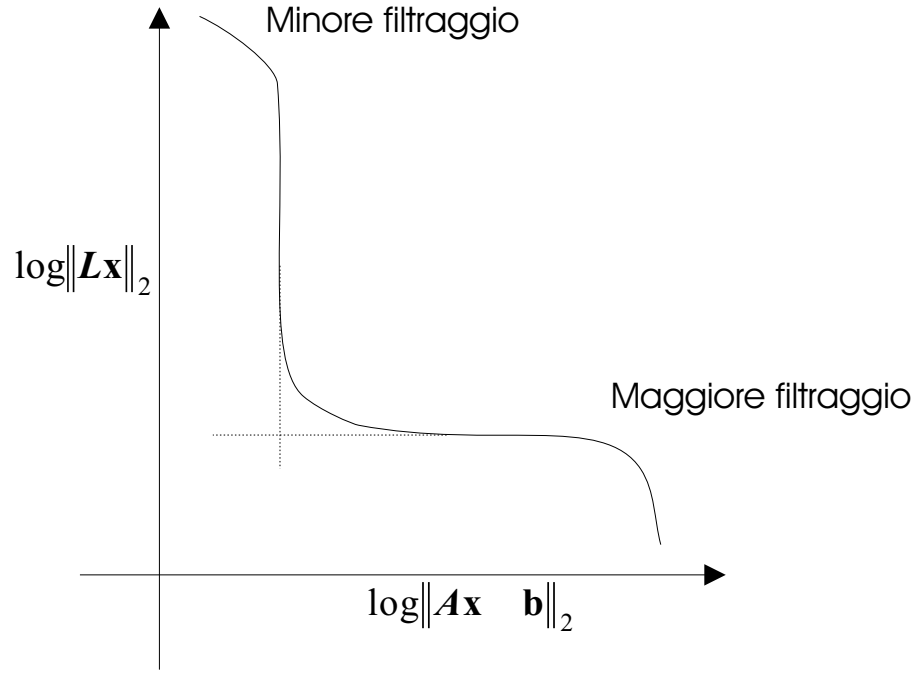

Figura 5: L-curve

ad alta frequenza. Di conseguenza il parametro di regolarizzazione del CG è proprio il numero di iterazioni. Poiché il CG può essere applicato solo in presenza di matrici simmetriche definite positive, si rende necessario, qualora queste non lo fossero, a passare alle equazioni normali associate $\boldsymbol{A}^T\boldsymbol{A}\mathbf{x} = \boldsymbol{A}^T\mathbf{b}$. Una limitazione di questo approccio è che il condizionamento del sistema è stato quadrato dal prodotto matriciale. Inoltre, non è facile dare una stima accurata del numero ottimo di iterazioni.

3.2 Primo test: matrice malcondizionata

Riprendiamo ora il sistema dell'ultimo test eseguito nella sezione precedente, in cui $\kappa = 10^8$. In figura 6 e 7 sono rappresentati l'andamento dei valori singolari e dei coefficienti di Fourier. Si può notare come la condizione di Picard sia rispettata, ma in presenza di rumore i coefficienti di Fourier che tendono a zero con la velocità dei valori singolari siano in numero minore, rendendo necessaria la regolarizzazione del sistema. Provando a calcolare i λ ottimali con la L-curve ed il GCV si ottengono i risultati riportati in tabella 10 e le curve di figura 8 e 9.

Metodo	λ	Errore	Err. rel.
L-curve	0.0017	7.8317	0.78317
GCV	6.5526e-004	8.0134	0.80134

Tabella 10: Risultati della regolarizzazione con Tikhonov - primo test

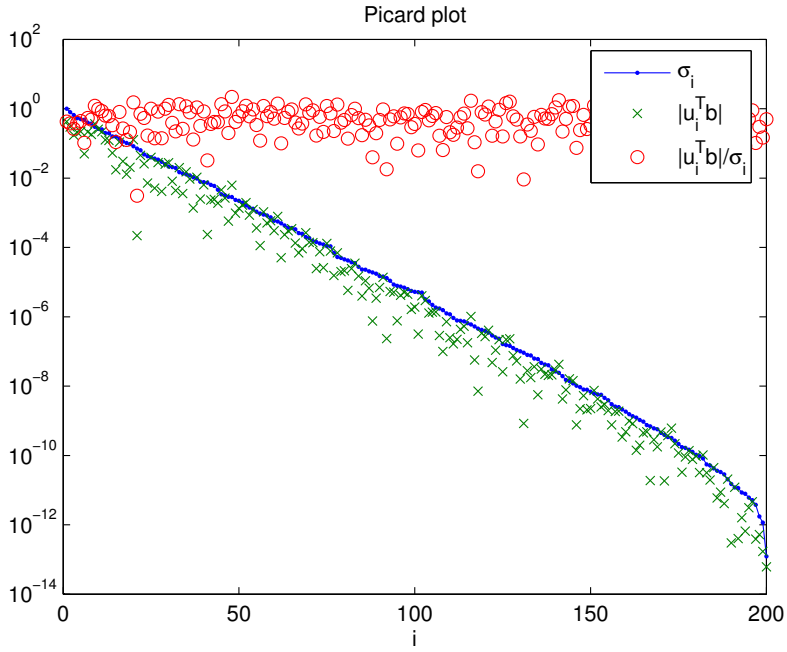

Figura 6: Andamento dei valori singolari e dei coefficienti di Fourier per il sistema imperturbato

L'errore commesso nel calcolo della soluzione regolarizzata è ancora elevato, ma molto minore (cinque ordini di grandezza) rispetto alla soluzione non regolarizzata. Per quanto concerne il gradiente coniugato, l'errore sulla soluzione è minimo dopo poche iterazioni, dopodiché esplode per assestarsi a quello della soluzione di Gauss. L'andamento dell'errore al crescere delle iterazioni è riportato in figura 10. In figura 11 è mostrato uno zoom dell'errore che mostra come il suo minimo sia raggiunto all'iterazione 13, con un errore relativo pari a 0.7979 e confrontabile con gli errori dati da Tikhonov. Se proviamo ad utilizzare il gradiente coniugato ai minimi quadrati (figura 12) la situazione è simile: alla iterazione 141 l'errore rispetto alla soluzione raggiunge un minimo, con errore relativo pari a 0.7738.

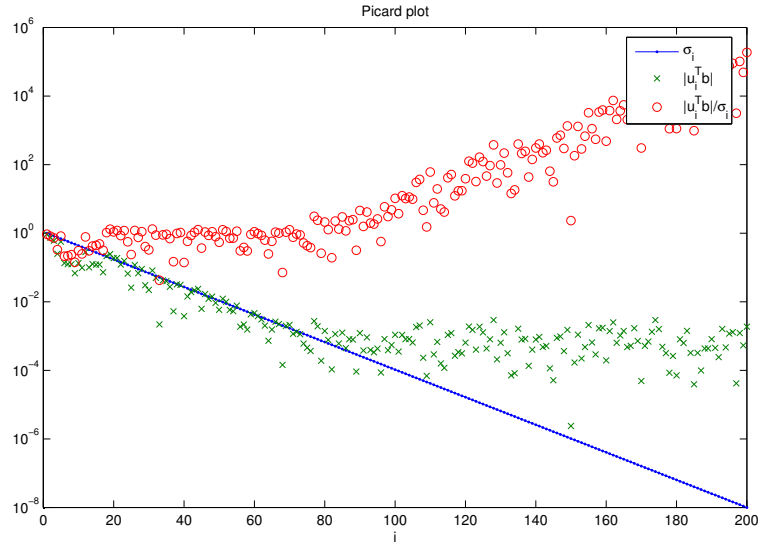

Figura 7: Andamento dei valori singolari e dei coefficienti di Fourier per il sistema perturbato

3.3 Secondo test: equazione di Frehdolm

Un problema più realistico può essere ottenuto discretizzando un'equazione di Frehldom di prima specie. Si fa riferimento al problema `baart` del toolbox [2] e si prova a risolverlo con Tikhonov e con il gradiente coniugato ai minimi quadrati. La perturbazione sui dati è di tipo gaussiano a media nulla, varianza unitaria e ampiezza 10^{-3}; la dimensione della matrice è 100. Innanzitutto vengono rappresentate, in figura 13 e 14 gli andamenti dei coefficienti di Fourier e dei valori singolari rispettivamente in assenza ed in presenza di rumore. Si noti come la condizione di Picard sia questa volta rispettata e come il rumore non consenta ai coefficienti di Fourier di decadere a zero come i valori singolari. Successivamente viene risolto il sitema utilizzando il gradiente coniugato ai minimi quadrati. In figura 15 e 16 sono mostrati i risultati ottenuti. La soluzione migliore trovata dal CGLS è quella della terza iterazione, con un errore relativo di 0.1680. Altrettanto valido è il comportamento di Tikhonov. In figura 17 e 18 sono mostrati la curva L e la curva GCV. I risultati sono riportati in tabella 11.

Metodo	λ	Errore	Err. rel.
L-curve	0.0580	0.2570	0.2050
GCV	0.0547	0.2644	0.2110

Tabella 11: Risultati della regolarizzazione con Tikhonov - (Baart)

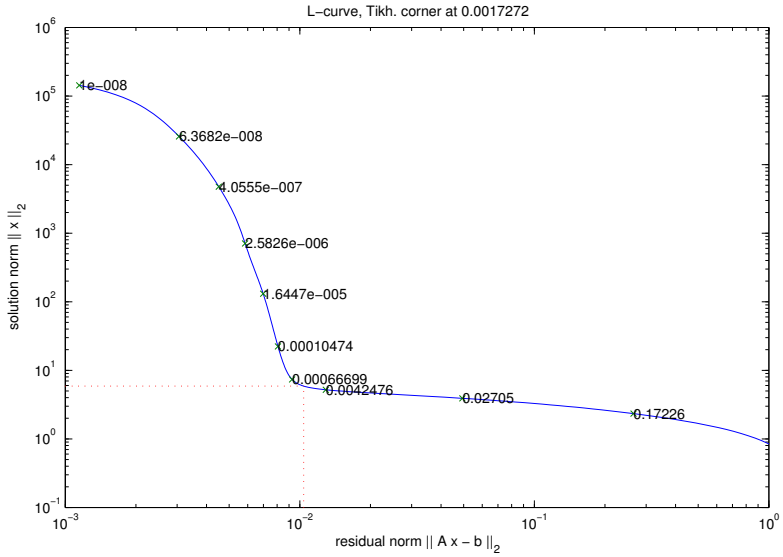

Figura 8: L-curve - primo test

Bibliografia

[1] A. Quarteroni, Sacco R., Saleri F. - *Matematica numerica* - Springer;

[2] Per Christian Hansen - *Regularizazion Tools* - Department of Mathematical Modeling, University of Denmark

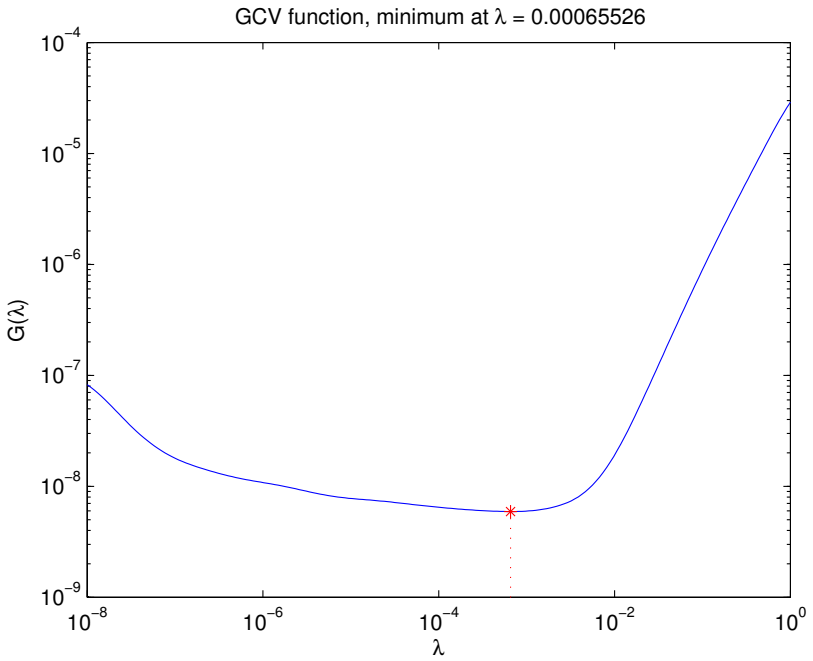

Figura 9: Curva GCV - primo test

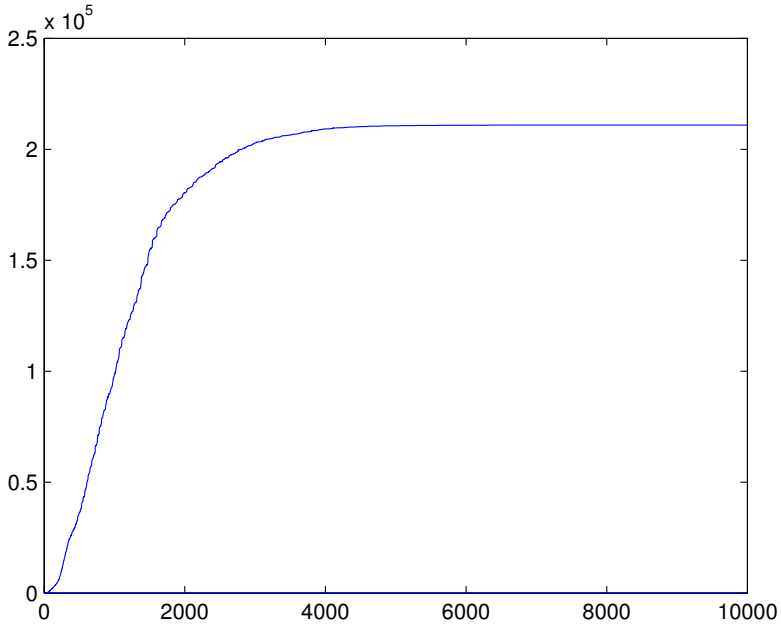

Figura 10: Errore per il gradiente coniugato - primo test

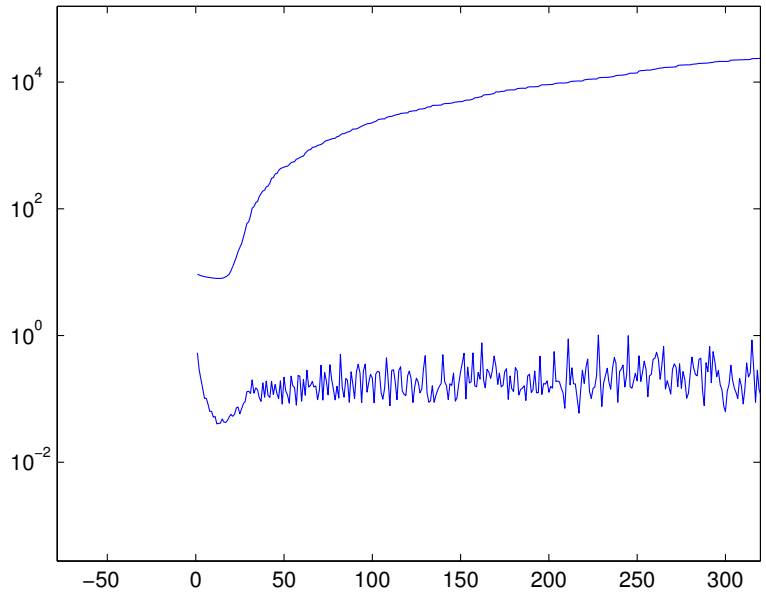

Figura 11: Errore per il gradiente coniugato - zoom - primo test

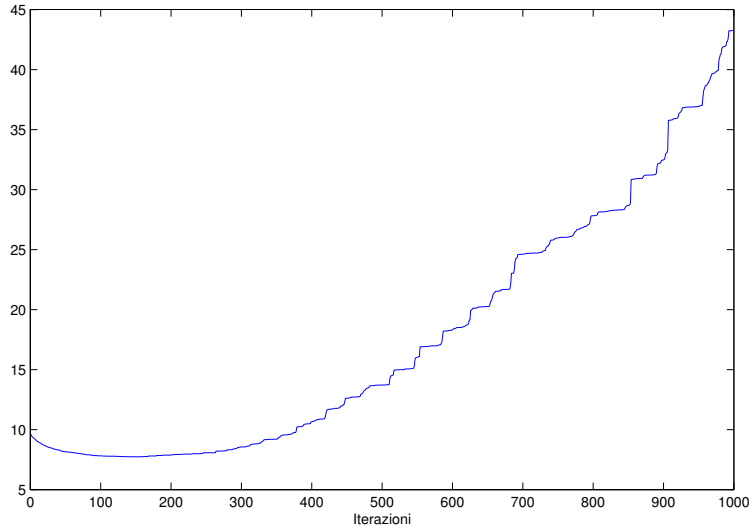

Figura 12: Errore per il gradiente coniugato ai minimi quadrati - primo test

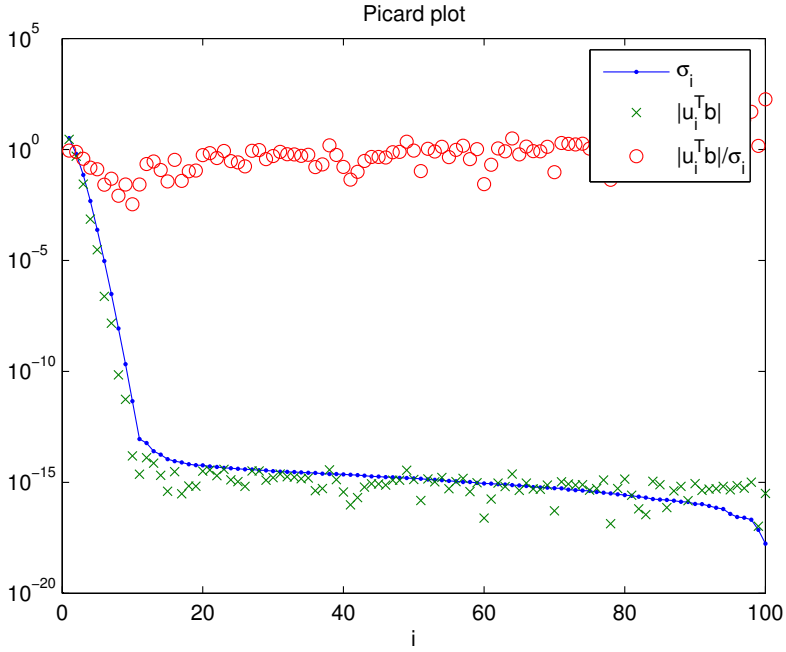

Figura 13: Andamento dei valori singolari e dei coefficienti di Fourier in assenza di rumore (Baart)

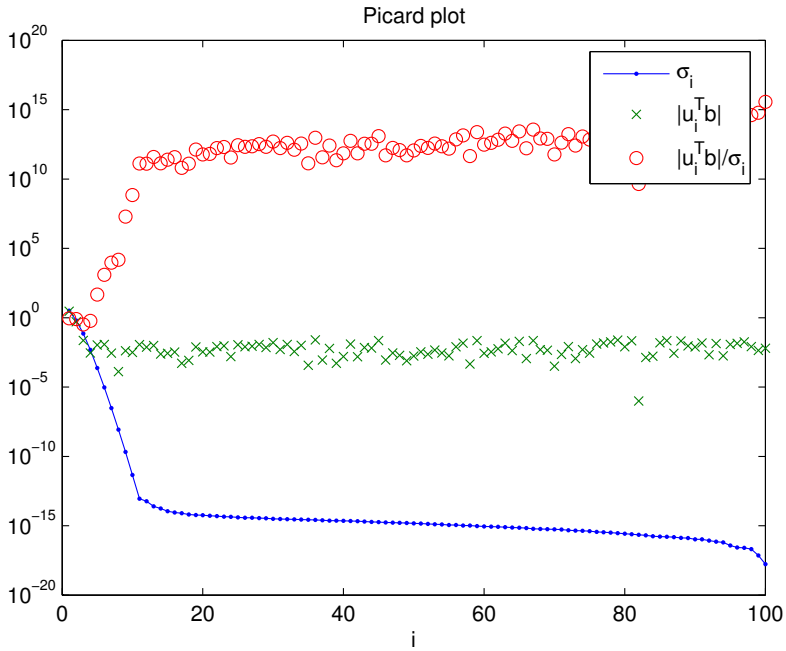

Figura 14: Andamento dei valori singolari e dei coefficienti di Fourier in presenza di rumore (Baart)

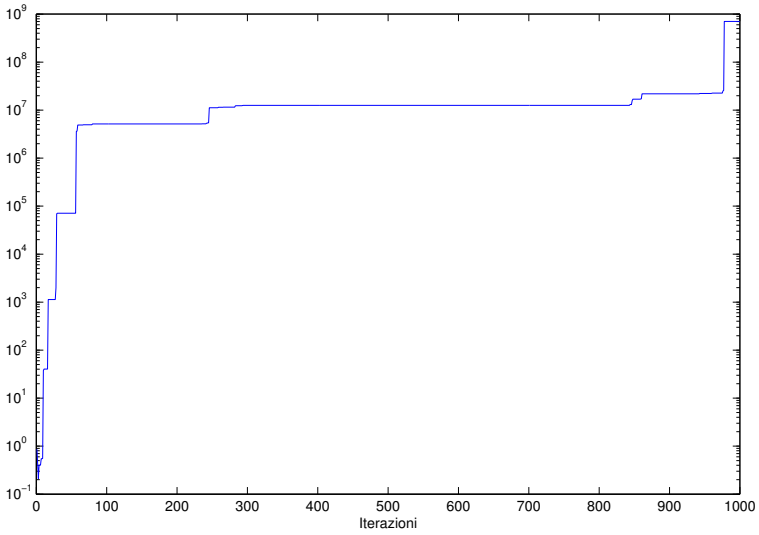

Figura 15: Andamento dell'errore per il gradiente coniugato (Baart)

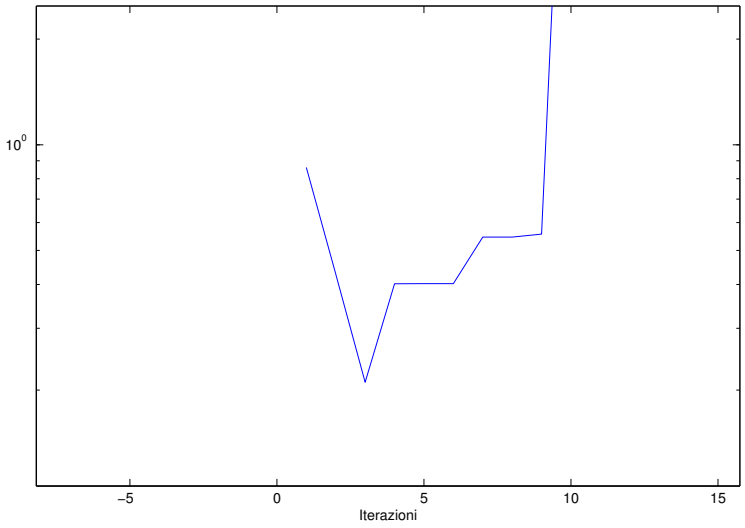

Figura 16: Andamento dell'errore per il gradiente coniugato (Baart) - zoom

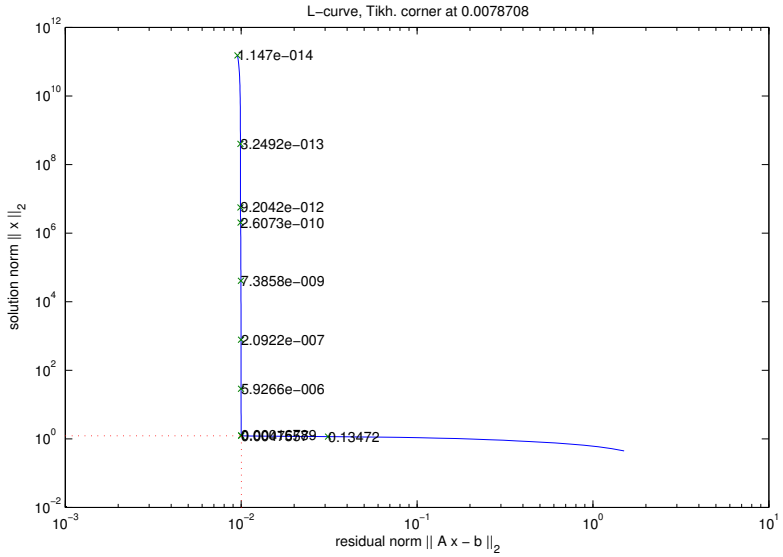

Figura 17: Curva L (Baart)

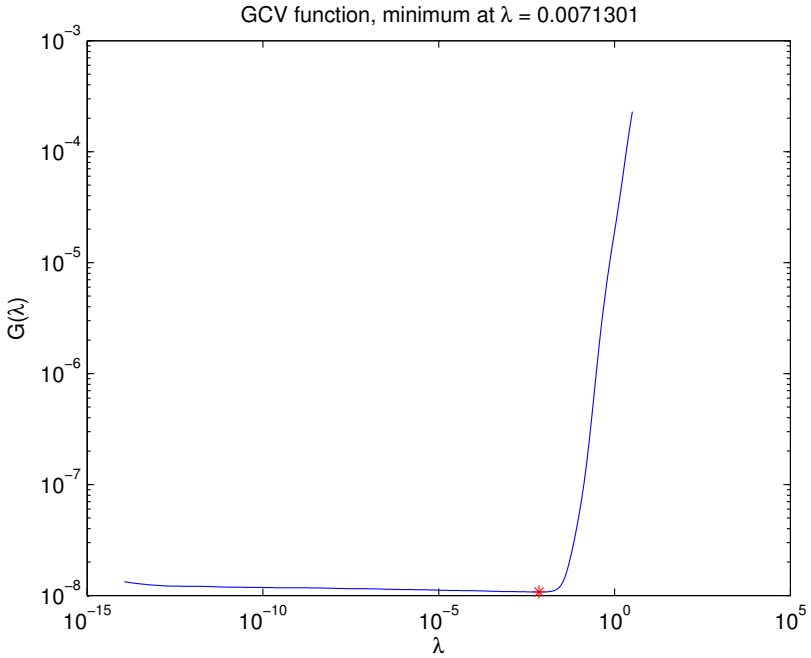

Figura 18: Curva GCV (Baart)