Metodo del gradiente coniugato per la soluzione di sistemi lineari e algoritmo QR per la ricerca di autovalori

Emanuele Tamponi e Luigi Tanca

11 aprile 2011

Parte I

Metodo del gradiente coniugato per la soluzione di sistemi lineari

1 Introduzione

Il metodo del gradiente coniugato è un metodo iterativo applicabile alla soluzione di alcuni sistemi lineari. Esso è un perfezionamento del metodo del gradiente, che ora descriviamo brevemente.

1.1 Metodo del gradiente

Sia A una matrice simmetrica definita positiva. Consideriamo la forma quadratica

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

Tale funzione è minima nel punto in cui si annulla il suo gradiente

$$\nabla\phi(\mathbf{y}) = \frac{1}{2}(A + A^T)\mathbf{y} - \mathbf{b} = A\mathbf{y} - \mathbf{b} = 0 \tag{2}$$

Quindi minimizzare ϕ della formula 1 è equivalente a risolvere il sistema lineare $A\mathbf{x} = \mathbf{b}$.

Si può calcolare il minimo applicando un metodo iterativo non stazionario del tipo

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)} \tag{3}$$

Chiamiamo $\mathbf{d}^{(k)}$ *direzione di decrescita* al passo k e α_k *lunghezza* del passo. Possiamo determinare α_k imponendo che, qualunque sia $\mathbf{d}^{(k)}$, la derivata di $\phi(\mathbf{x}^{(k+1)})$ rispetto a α si annulli. Abbiamo

$$\begin{aligned}\phi(\mathbf{x}^{(k+1)}) &= \frac{1}{2}\left(\mathbf{x}^{(k)}+\alpha\mathbf{d}^{(k)}\right)^T A\left(\mathbf{x}^{(k)}+\alpha\mathbf{d}^{(k)}\right)-\left(\mathbf{x}^{(k)}+\alpha\mathbf{d}^{(k)}\right)^T\mathbf{b}\\ &= \phi(\mathbf{x}^{(k)})+\frac{1}{2}\left(\mathbf{d}^{(k)}\right)^T A\mathbf{d}^{(k)}\cdot\alpha^2-\left(\mathbf{d}^{(k)}\right)^T\mathbf{r}^{(k)}\cdot\alpha\end{aligned}$$

in cui abbiamo indicato con $\mathbf{r}^{(k)}=\mathbf{b}-A\mathbf{x}^{(k)}$ il *residuo* al passo k. Imponiamo l'annullarsi della derivata:

$$\frac{d}{d\alpha}\phi(\mathbf{x}^{(k+1)})=\left(\mathbf{d}^{(k)}\right)^T A\mathbf{d}^{(k)}\cdot\alpha-\left(\mathbf{d}^{(k)}\right)^T\mathbf{r}^{(k)}=0$$

Da cui segue che il minimo si ottiene per

$$\alpha_k=\frac{\left(\mathbf{d}^{(k)}\right)^T\mathbf{r}^{(k)}}{\left(\mathbf{d}^{(k)}\right)^T A\mathbf{d}^{(k)}} \tag{4}$$

La formula 4 è valida anche per il metodo del gradiente coniugato. Ciò che contraddistingue il metodo del gradiente è la scelta della direzione di decrescita, che è quella opposta alla direzione del gradiente della funzione ϕ nel punto $\mathbf{x}^{(k)}$. Questa direzione coincide con il residuo al passo k, infatti

$$\nabla\phi(\mathbf{x}^{(k)})=A\mathbf{x}^{(k)}-\mathbf{b}=-\mathbf{r}^{(k)}$$

Dunque nel metodo del gradiente l'iterazione assume la forma

$$\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}+\alpha_k\mathbf{r}^{(k)} \tag{5}$$

1.2 Metodo del gradiente coniugato

È possibile effettuare una scelta più accurata della direzione di discesa. Si può dimostrare che il metodo del gradiente coniugato utilizza una scelta di tale direzione in modo tale da convergere alla soluzione esatta in n iterazioni, se si opera in aritmetica infinita. Ma poiché esso viene utilizzato con l'aritmetica di macchina, e poiché è conveniente utilizzarlo per una stima rapida della soluzione, è chiamato metodo iterativo.

Si dice che il vettore $\mathbf{x}$ è *ottimale* rispetto alla direzione $\mathbf{p}$ se

$$\phi(\mathbf{x})\leq\phi(\mathbf{x}+\lambda\mathbf{p}),\qquad\forall\lambda\in\mathbb{R} \tag{6}$$

Se $\mathbf{x}$ è ottimale rispetto ad ogni direzione di un sottospazio vettoriale V, allora si dice che $\mathbf{x}$ è ottimale rispetto a V.

Si dimostra che $\mathbf{x}$ è ottimale rispetto a $\mathbf{p}$ se e solo se la direzione $\mathbf{p}$ è ortogonale al residuo $\mathbf{r}$, cioè

$$\mathbf{p}^T\mathbf{r} = 0 \tag{7}$$

Si verifica facilmente che nel metodo del gradiente il vettore $\mathbf{x}^{(k+1)}$ è reso ottimale rispetto alla direzione di discesa $\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ dal parametro α_k. Cioè $\mathbf{p}^T\mathbf{r} = \left(\mathbf{d}^{(k)}\right)^T \mathbf{r}^{(k+1)} = \left(\mathbf{r}^{(k)}\right)^T \mathbf{r}^{(k+1)} = 0$.

Nel metodo del gradiente coniugato la direzione di discesa $\mathbf{d}^{(k)}$ è scelta in modo tale da far sì che il vettore $\mathbf{x}^{(k+1)}$ sia ottimale rispetto anche a tutte le precedenti direzioni. In questo modo il vettore $\mathbf{x}^{(k+1)}$ è ottimale rispetto al sottospazio V di dimensione $k+1$ individuato dalle direzioni $\mathbf{d}^{(i)}$, $i = 0, ..., k$. Dunque, in virtù della definizione di ottimalità espressa nella formula 6, possiamo dire che non è possibile individuare nel sottospazio V un vettore $\mathbf{y}$ tale che $\phi(\mathbf{y}) < \phi(\mathbf{x}^{(k+1)})$. Questo significa che il vettore $\mathbf{x}^{(n)}$ è ottimale rispetto all'intero spazio vettoriale, e quindi coincide col minimo e con la soluzione del sistema lineare in esame.

Vediamo com'è possibile effettuare la scelta della direzione di discesa al passo k.

Supponiamo che $\mathbf{x}^{(k)}$ sia ottimale rispetto a $\mathbf{p}$ e, quindi, che $\mathbf{p}^T\mathbf{r}^{(k)} = 0$. Poniamo

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

e imponiamo che anche $\mathbf{x}^{(k+1)}$ sia ottimale rispetto a $\mathbf{p}$. Dev'essere

$$\mathbf{p}^T\mathbf{r}^{(k+1)} = \mathbf{p}^T\left(\mathbf{r}^{(k)} - A\mathbf{q}\right) = -\mathbf{p}^T A\mathbf{q} = 0$$

cioè le direzioni $\mathbf{p}$ e $\mathbf{q}$ devono essere A-ortogonali o *A-coniugate*.

Scegliamo $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$ e consideriamo direzioni del tipo

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

Occorre che valga

$$\left(\mathbf{p}^{(k+1)}\right)^T A\mathbf{p}^{(i)} = 0, \qquad i = 0, ..., k$$

Cioè che la direzione $\mathbf{p}^{(k+1)}$ sia A-coniugata con tutte le precedenti direzioni. Tale richiesta si traduce nella condizione

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

A questo punto si può scrivere il metodo del gradiente coniugato come

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{p}^{(k)} \tag{8}$$

L'ottimalità di $\mathbf{x}^{(k+1)}$ rispetto a $\mathbf{q} = \alpha_k\mathbf{p}^{(k)}$ è garantita dalla scelta di α_k secondo la formula 4, come nel metodo del gradiente.

1.3 Precondizionamento

È possibile accelerare la convergenza del metodo del gradiente e del gradiente coniugato attraverso un opportuno precondizionatore P, che approssimi A e sia nel contempo invertibile con un basso costo computazionale.

Con l'uso del precondizionatore, le formule per il metodo gradiente diventano

$$\begin{aligned} \mathbf{z}^{(k)} &= P^{-1}\mathbf{r}^{(k)} \\ \alpha_k &= \frac{\left(\mathbf{z}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{z}^{(k)}\right)^T A\mathbf{z}^{(k)}} \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k P^{-1}\mathbf{r}^{(k)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)} \end{aligned} \tag{9}$$

La formula finale del metodo gradiente coniugato è sempre la 8, ma nel calcolo di $\mathbf{p}^{(k)}$ si precondiziona il residuo k-esimo

$$\begin{aligned} \mathbf{z}^{(k)} &= P^{-1}\mathbf{r}^{(k)} \\ \mathbf{p}^{(0)} &= \mathbf{z}^{(0)} \\ \mathbf{p}^{(k+1)} &= \mathbf{z}^{(k+1)} - \beta_k \mathbf{p}^{(k)} \\ \alpha_k &= \frac{\left(\mathbf{p}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{p}^{(k)}\right)^T A\mathbf{p}^{(k)}} \\ \beta_k &= \frac{\left(\mathbf{p}^{(k)}\right)^T A\mathbf{z}^{(k+1)}}{\left(\mathbf{p}^{(k)}\right)^T A\mathbf{p}^{(k)}} \end{aligned} \tag{10}$$

1.4 Ottimizzazioni

È opportuno che ad ogni passo dell'iterazione la complessità computazionale resti $O(n^2)$, e cioè che l'operazione più onerosa sia una moltiplicazione matrice per vettore.

Il primo miglioramento si ha aggiornando il residuo nel modo seguente

$$\mathbf{r}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k)} - \alpha_k A\mathbf{d}^{(k)} = \mathbf{r}^{(k)} - \alpha_k A\mathbf{d}^{(k)}$$

in cui la scelta di $\mathbf{d}^{(k)}$ varia a seconda del metodo utilizzato. In questo modo, è sufficiente calcolare una sola volta il prodotto $A\mathbf{d}^{(k)}$ e impiegarlo dove necessario.

Un'ulteriore ottimizzazione si trova evitando di invertire esplicitamente il precondizionatore P, risolvendo invece ad ogni passo il sistema lineare

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

Algorithm 1 Struttura generale

```
function [x,flag,relres,k,resvec,errvec] = ...
          cn2_method(A,b,tol,maxit,M1,M2,x0,x_true)
    %...
end
```

1.5 Generalizzazione

I metodi del gradiente e del gradiente coniugato sono utilizzabili solo se A è simmetrica definita positiva. Se non lo è, è possibile applicare entrambi i metodi al sistema normale

$$A^T A\mathbf{x} = A^T\mathbf{b} \tag{11}$$

che ha la stessa soluzione del problema di partenza, $A\mathbf{x} = \mathbf{b}$, se A è quadrata non singolare. Tuttavia, per motivi di stabilità numerica e per il fatto che la complessità computazionale del calcolo di $A^T A$ è $O(n^3)$, il metodo va utilizzato evitando di calcolare esplicitamente la matrice normale. Resta il fatto che il numero di condizionamento del sistema 11 è pari al quadrato di quello del sistema di partenza.

2 Implementazione

Tutti i metodi sono stati implementati in *Matlab*®, con gli stessi input e output, com'è mostrato nell'Algoritmo 1.

Inoltre la sintassi e il significato dei parametri sono gli stessi della chiamata alla funzione Matlab `pcg` (a cui si rimanda), a cui è stato aggiunto un parametro opzionale, `x_true`, e un ulteriore vettore di output, `errvec`. Qualora `x_true` venisse specificato, verrebbe interpretato come la soluzione esatta del problema in esame; quindi `errvec` conterrebbe l'errore assoluto tra soluzione esatta e soluzione approssimata ad ogni passo dell'algoritmo, cioè $\mathbf{e}^{(k)} = \left\|\mathbf{x} - \mathbf{x}^{(k)}\right\|$. Ovviamente ha senso utilizzare `x_true` solo un eventuale test dell'algoritmo.

Come criterio di stop utilizziamo

$$\frac{\left\|\mathbf{r}^{(k)}\right\|}{\|\mathbf{b}\|} \leq \tau \ \wedge \ k > N$$

Nel codice:τ è `tol`, N è `maxit`.

L'Algoritmo 2 mostra la funzione di supporto `precond` che, date due matrici (che possono anche essere vuote), e il residuo `r`, restituisce il residuo precondizionato `z`. In questo modo è possibile effettuare una sola volta la fattorizzazione LU del precondizionatore P.

Oltre al metodo del gradiente e del gradiente coniugato, è stato implementato un metodo iterativo generico nella forma

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + P^{-1}\mathbf{r}^{(k)}$$

Algorithm 2 Funzione `precond`

```
function z = precond(M1,M2,r)
    z = r;
    if ~isempty(M1)
        z = M1\z;
    end
    if ~isempty(M2)
        z = M2\z;
    end
end
```

Tabella 1: Parametri per i test

Parametro	Valore	Significato
`tol`	10^{-8}	Tolleranza sul residuo relativo
`maxit`	Pari alla grandezza della matrice	Numero massimo di iterazioni
`x0`	$(0, \ldots, 0)^T$	Vettore iniziale
`x_true`	$(1, \ldots, 1)^T$	Soluzione esatta

3 Test e conclusioni

3.1 Implementazione dei test

Abbiamo effettuato due tipi di test. Nel primo abbiamo messo a confronto i vari metodi nella soluzione di vari tipi di matrice di dimensione media, tra 100 e 500. Nel secondo test abbiamo utilizzato una matrice sparsa di dimensione 2000. L'errore percentuale mostrato nelle tabelle è

$$Err\% = 100 \times \frac{\left\|\mathbf{x} - \mathbf{x}^{(k)}\right\|}{\|\mathbf{x}\|}$$

3.2 Risultati

Per i test sono stati utilizzati i parametri mostrati in Tabella 1.

Il primo tipo di test è stato effettuato su una matrice random di dimensione 200 (Figura 1). Il numero di condizionamento è nell'ordine di 10^8. Si vede subito come le prestazioni del metodo di Gauss-Seidel siano molto basse comparate agli altri due metodi, e come i risultati della nostra implementazione del metodo del gradiente coniugato siano molto vicini all'algoritmo di Matlab. Come ci si aspettava, entrambi i metodi basati su gradiente presentano vistose oscillazioni sul residuo, dovute al malcondizionamento della matrice di test. Inoltre si nota che il residuo non varia in maniera significativa dopo le prime 20-40 iterazioni: considerazione importante nel caso in cui si utilizzino i metodi iterativi per una soluzione veloce.

Figura 1: Andamento del residuo per una matrice di tipo `rand(200)`

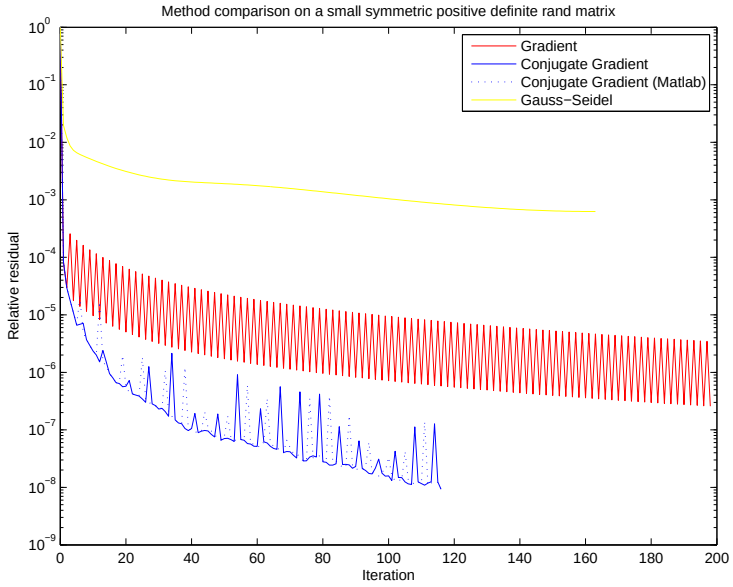
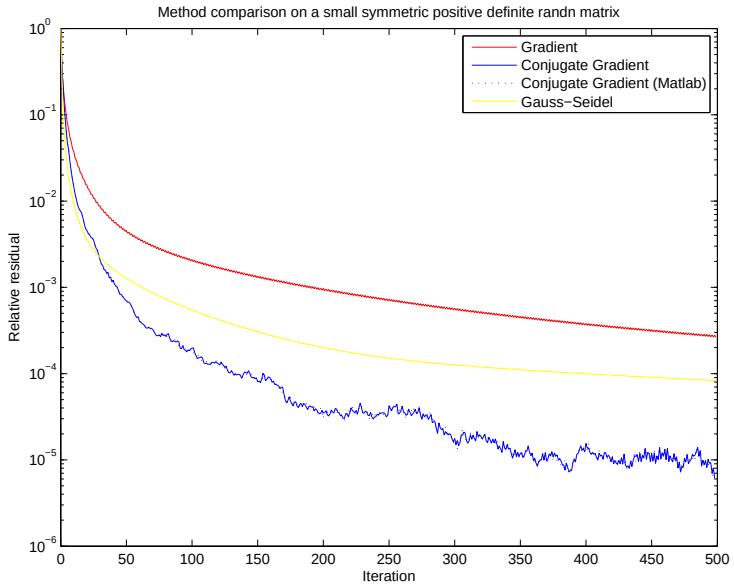


Tabella 2: Risultati su una matrice di tipo `rand(200)`

Metodo	`k`	`flag`	`relres`	Errore	Tempo
Gradiente	200	1	$2,247 \cdot 10^{-7}$	$1,124\%$	97 ms
Grad. coniugato	135	0	$9,869 \cdot 10^{-9}$	$0,429\%$	29 ms
Grad. coniugato (Matlab)	132	0	$9,230 \cdot 10^{-9}$	$0,431\%$	38 ms
Gauss-Seidel	200	1	$1,539 \cdot 10^{-3}$	$902,8\%$	53 ms

Figura 2: Andamento del residuo per una matrice di tipo `randn(500)`



È importante sottolineare come, su matrici come questa, il ridursi della norma del residuo non sia una garanzia totale sull'effettiva convergenza del metodo. Ricordiamo infatti che vale la disuguaglianza

$$\frac{\left\|\mathbf{e}^{(k)}\right\|}{\left\|\mathbf{x}^{(k)}\right\|} \leq \kappa(A)\frac{\left\|\mathbf{r}^{(k)}\right\|}{\|\mathbf{b}\|}$$

E che quindi l'errore relativo può risultare molto più alto del residuo relativo, come si nota dai risultati mostrati in Tabella 2. In particolare il metodo di Gauss-Seidel compie un errore relativo del 900% (inaccettabile, e dovuto al fatto che $\rho(B_{GS}) \approx 1$), mentre gli altri metodi raggiungono soglie d'errore inferiori all'1%.

Se si sceglie una matrice random con lo stesso numero di condizionamento[1], ma contenente elementi casuali con distribuzione normale anziché omogenea (cioè se si usa `randn` anziché `rand`), i risultati cambiano significativamente. Il metodo di Gauss-Seidel migliora in modo incredibile (errore relativo attorno al 20%), mentre gli altri metodi peggiorano significativamente (Figura 2 e Tabella 3).

Abbiamo effettuato il test su altri due tipi di matrici: `hilb(100)` (numero di condizionamento 10^{19}) e `pascal(100)` (numero di condizionamento 10^{60}).

Per quanto riguarda la matrice `pascal(100)`, non abbiamo potuto effettuare il test del metodo di Gauss-Seidel a causa di singolarità numeriche sul

[1] Per ottenere lo stesso numero di condizionamento, si è passati da una matrice di dimensione 200 a una di dimensione 500. Vedremo tra poco come il numero di condizionamento influirà sui vari metodi.

Tabella 3: Risultati su una matrice di tipo `randn(500)`

Metodo	`k`	`flag`	`relres`	Errore	Tempo
Gradiente	500	1	$2,198 \cdot 10^{-4}$	$16,08\%$	125 ms
Grad. coniugato	500	1	$3,826 \cdot 10^{-6}$	$6,571\%$	155 ms
Grad. coniugato (Matlab)	500	1	$3,880 \cdot 10^{-6}$	$6,574\%$	201 ms
Gauss-Seidel	500	1	$7,004 \cdot 10^{-5}$	$17,69\%$	641 ms

Figura 3: Andamento del residuo per una matrice di tipo `pascal(100)`

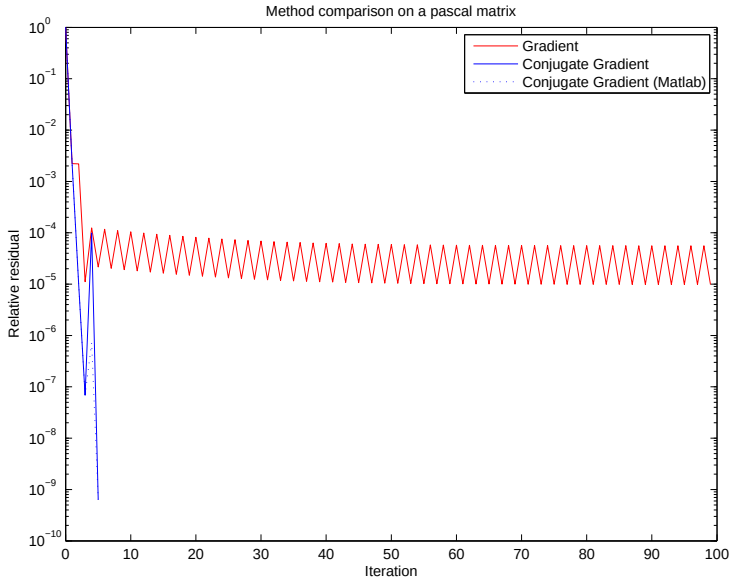
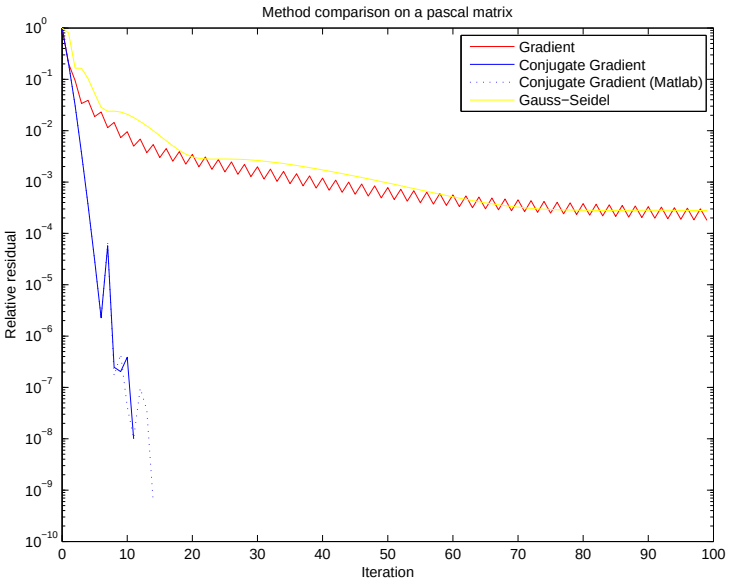


Tabella 4: Risultati su una matrice di tipo `pascal(100)`

Metodo	`k`	`flag`	`relres`	Errore	Tempo
Gradiente	100	1	$9,825 \cdot 10^{-6}$	97%	31 ms
Grad. coniugato	5	0	$6,355 \cdot 10^{-10}$	94%	1 ms
Grad. coniugato (Matlab)	5	0	$6,355 \cdot 10^{-10}$	94%	2 ms

Figura 4: Andamento del residuo per una matrice di tipo `hilb(100)`



precondizionatore. Tutti gli altri metodi, pur raggiungendo un residuo relativo estremamente basso (il metodo del gradiente coniugato converge dopo 5 iterazioni!), commettono un errore di circa il 95% (Tabella 4 e Figura 3).

I risultati sulla matrice `hilb(100)` sono stupefacenti: nonostante il numero di condizionamento sia ben superiore alla precisione di macchina, tutti i metodi (compreso Gauss-Seidel) raggiungono precisioni migliori rispetto a tutti i precedenti problemi. Il metodo del gradiente coniugato ha una convergenza velocissima (Figura 4 e Tabella 5).

Il secondo tipo di test è stato effettuato su una matrice generata con la funzione Matlab `sprandsym(2000,...)`, con numero di condizionamento (stimato) 10^4 e densità 5%. I risultati sono mostrati in Figura 5 e in Tabella 7. Anche in questo caso, il metodo di Gauss-Seidel e del gradiente si attestano quasi subito su un margine d'errore che poi non migliorerà sensibilmente. Al contrario, il metodo del gradiente coniugato migliora in maniera esponenziale la sua precisione fino a raggiungere la convergenza in un numero relativamente basso di passi (la grandezza della matrice è 2000).

Abbiamo successivamente testato i metodi del gradiente e del gradiente co-

Tabella 5: Risultati su una matrice di tipo `hilb(100)`

Metodo	`k`	`flag`	`relres`	Errore	Tempo
Gradiente	100	1	$1,792 \cdot 10^{-4}$	$3,830\%$	10 ms
Grad. coniugato	11	0	$9,974 \cdot 10^{-9}$	$0,208\%$	2 ms
Grad. coniugato (Matlab)	14	0	$5,839 \cdot 10^{-10}$	$0,081\%$	7 ms
Gauss-Seidel	100	1	$2,783 \cdot 10^{-4}$	$13,93\%$	11 ms

Figura 5: Andamento del residuo per una matrice di tipo `sprandsym(2000,...)`

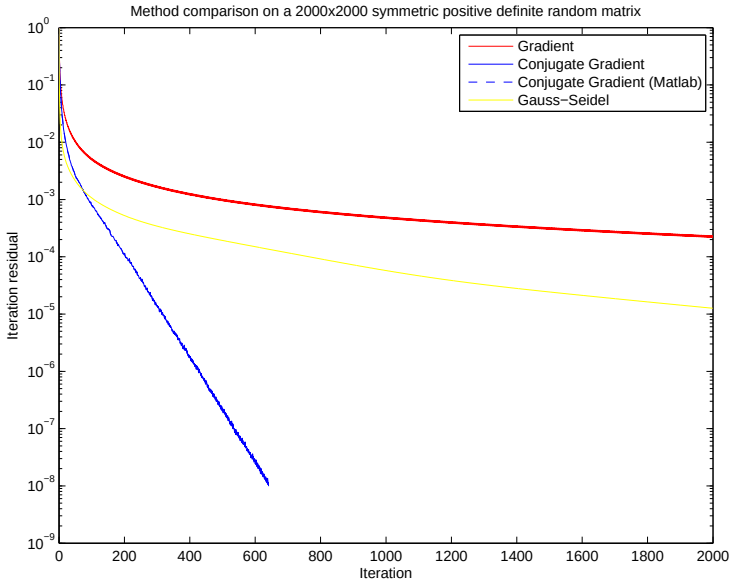
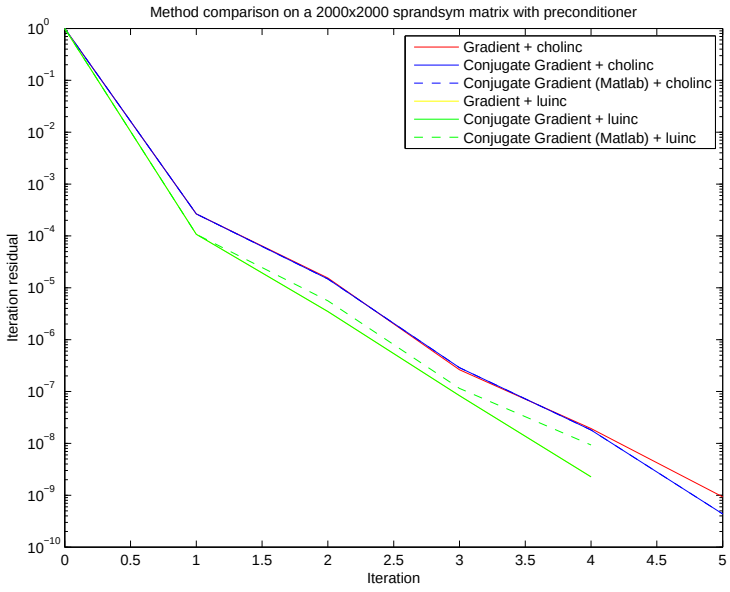


Tabella 6: Risultati su una matrice di tipo `sprandsym(2000,...)`

Metodo	`k`	`flag`	`relres`	Errore	Tempo
Gradiente	2000	1	$2,034 \cdot 10^{-4}$	$17,88\%$	1872 ms
Grad. coniugato	637	0	$9,956 \cdot 10^{-9}$	$0,0002\%$	636 ms
Grad.coniugato (Matlab)	643	0	$0,738 \cdot 10^{-9}$	$0,0002\%$	749 ms
Gauss-Seidel	2000	1	$1,021 \cdot 10^{-5}$	$1,055\%$	3706 ms

Figura 6: Andamento del residuo per una matrice di tipo `sprandsym(2000,...)` utilizzando un precondizionatore



niugato sulla stessa matrice, utilizzando un precondizionatore del tipo Cholesky incompleto e LU incompleto. Il parametro di tolleranza `sigma` è stato impostato a 10^{-5}. La convergenza avviene in maniera velocissima, con 4-5 iterazioni. Il tempo necessario per il calcolo della fattorizzazione incompleta può quindi essere recuperato agevolmente, come mostrato in Tabella 7, per quanto riguarda la fattorizzazione di Cholesky, mentre la fattorizzazione LU incompleta riduce il numero di iterazioni senza ridurre il tempo totale impiegato. L'andamento del residuo è mostrato in Figura 6.

Tabella 7: Risultati su una matrice di tipo `sprandsym(2000,...)` utilizzando un precondizionatore

Metodo	`k`	`flag`	`relres`	Errore	Tempo
Gradiente + `cholinc`	5	0	$9,803 \cdot 10^{-10}$	$0,318\,ppm$	79 ms
Grad. con. + `cholinc`	5	0	$5,725 \cdot 10^{-10}$	$0,0901\,ppm$	74 ms
Grad. con. (Matlab) + `cholinc`	5	0	$5,725 \cdot 10^{-10}$	$0,0901\,ppm$	63 ms
Gradiente + `luinc`	4	0	$7,180 \cdot 10^{-9}$	$0,718\,ppm$	722 ms
Grad. con. + `luinc`	4	0	$1,834 \cdot 10^{-9}$	$0,719\,ppm$	731 ms
Grad. con. (Matlab) + `luinc`	5	0	$6,575 \cdot 10^{-10}$	$0,274\,ppm$	736 ms

3.3 Conclusioni

Il metodo del gradiente coniugato si è sempre di mostrato di gran lunga migliore del metodo del gradiente e del metodo di Gauss-Seidel, che sono gli altri due metodi testati (per la loro somiglianza col metodo in esame e anche per la tipologia di matrici su cui sono utilizzabili). Inoltre l'implementazione da noi effettuata risulta comparabile con quella di Matlab, sia in termini di velocità che di prestazioni sull'errore.

Abbiamo visto che l'utilizzo di un precondizionatore migliora eccezionalmente le prestazioni di tutti i metodi (non è presente nei test, ma anche il metodo di Richardson stazionario con l'utilizzo del precondizionatore Cholesky raggiunge prestazioni ottime), e che la scelta del metodo di calcolo del precondizionatore altera nettamente la velocità con cui avvengono le iterazioni.

Possiamo dire con sicurezza che è soprattutto il *tipo* di matrice ad influenzare la convergenza del metodo, e solo in secondo luogo interviene il numero di condizionamento. Solo in alcuni tipi di matrice questo è influente, come abbiamo visto applicando i tre metodi alle matrici `hilb` e `pascal`.

Infine, dai grafici è emerso come una "prima approssimazione" emerga in maniera estremamente veloce, dopo appena 5-10 iterazioni, anche senza l'uso dei precondizionatori: ciò fa comprendere come i metodi iterativi siano effettivamente utilizzabili soprattutto per una stima veloce della soluzione, laddove non sia necessaria una precisione relativa superiore a $10^{-3} - 10^{-5}$.

Parte II
Algoritmo QR per la ricerca di autovalori

4 Introduzione

4.1 Fattorizzazione QR di Householder

Ogni matrice A $m \times n$ è fattorizzabile nella forma

$$A = QR$$

con Q matrice $m \times m$ ortogonale e R triangolare superiore con le stesse dimensioni di A. Tale fattorizzazione non è unica: la nostra implementazione utilizza le matrici elementari di Householder. Una matrice elementare di Householder reale è del tipo

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

È immediato osservare che H è simmetrica e ortogonale. Per un dato vettore $\mathbf{x}$, esiste $\mathbf{w}$ tale che

$$H\mathbf{x} = k\mathbf{e}_1$$

Una possibile scelta per $\mathbf{w}$ è la seguente

$$\begin{aligned} k &= -sign(x_1)\,\|\mathbf{x}\| \\ \mathbf{w} &= \frac{\mathbf{x} - k\mathbf{e}_1}{\|\mathbf{x} - k\mathbf{e}_1\|} \end{aligned}$$

A partire dalla matrice A da fattorizzare, utilizzeremo le matrici elementari di Householder per generare una successione di matrici $A^{(i)}$ tale che $A^{(n)}$ sia trangolare superiore. Cominciamo scrivendo A in termini delle sue colonne

$$A = A^{(1)} = \left[\mathbf{a}_1^{(1)}\, \mathbf{a}_2^{(1)}\, \cdots\, \mathbf{a}_n^{(1)}\right]$$

Si può scrivere una matrice di Householder H_1 tale che

$$H_1\mathbf{a}_1^{(1)} = k_1\mathbf{e}_1$$

Applichiamo tale matrice a sinistra di $A^{(1)}$ per generare la prossima matrice della successione

$$A^{(2)} = H_1A^{(1)} = \left[k_1\mathbf{e}_1\, \mathbf{a}_2^{(2)}\, \cdots\, \mathbf{a}_n^{(2)}\right]$$

Al generico passo i, la matrice $A^{(i)}$ è nella forma:

$$A^{(i)} = \begin{bmatrix} k_1 & * & \cdots & * & \cdots & * \\ 0 & \ddots & \ddots & \vdots & \ddots & \vdots \\ \vdots & \ddots & k_{i-1} & * & \cdots & * \\ 0 & \cdots & 0 & & & \\ \vdots & \ddots & \vdots & & \hat{A}^{(i)} & \\ 0 & \cdots & 0 & & & \end{bmatrix}$$

Una opportuna matrice di Householder $\hat{H}_1$ applicata a $\hat{A}^{(i)}$ e orlata fino alla dimensione di A, fa sì che:

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

Procedendo in modo analogo, dopo $n-1$ nel caso di matrici quadrate, o dopo n passi nel caso di matrici rettangolari con $m > n$, si avrà una matrice $A^{(n)}$ triangolare superiore. Perciò

$$R = A^{(n)} = H_{n-1}A^{(n-1)} = H_{n-1}H_{n-2}\cdots H_1 A^{(1)} = Q^T A$$

La matrice

$$Q = H_1 H_2 \cdots H_{n-1}$$

è ortogonale, pertanto $A = QR$ cosituisce una fattorizzazione QR della matrice A.

4.2 Algoritmo QR per la ricerca di autovalori

Si tratta di un metodo iterativo che consente di approssimare tutti gli autovalori di una matrice A. Lo schema dell'algoritmo è particolarmente semplice.

Si parte ponendo $A^{(0)} = A$. Nella generica iterazione si effettua la fattorizzazione QR della matrice $A^{(k)}$ e si calcola la nuova iterata moltiplicando i fattori in ordine inverso

$$\begin{aligned} A^{(k)} &= Q^{(k)}R^{(k)} \\ A^{(k+1)} &= R^{(k)}Q^{(k)} \end{aligned}$$

Le matrici della successione sono unitariamente simili, e come tali avranno tutte gli stessi autovalori. Sotto l'ipotesi di *autovalori distinti in modulo*, la successione converge ad una matrice triangolare superiore, che essendo simile alla matrice A di partenza, contiene i suoi autovalori sulla diagonale principale.

4.3 Ottimizzazioni

Implementata in maniera ottimizzata, la fattorizzazione QR di Householder richiede soltanto $O(\frac{2}{3}n^3)$ moltiplicazioni. Notiamo innanzittutto che la matrice H si può scrivere come

$$H = I - \frac{1}{\beta}\mathbf{v}\mathbf{v}^T, \quad \mathbf{v} = \mathbf{x} - k\mathbf{e}_1, \quad \beta = \sigma(\sigma + |x_1|), \quad \sigma = \|\mathbf{x}\|$$

Osserviamo inoltre che non è indispensabile costruire H, né effettuare esplicitamente il prodotto per una matrice B (che ha complessità $O(n^3)$). Infatti, dato un generico vettore $\mathbf{z}$

$$\begin{aligned} H\mathbf{z} &= \mathbf{z} - \delta\mathbf{v}, \qquad \delta = \frac{1}{\beta}\mathbf{v}^T\mathbf{z} \\ \mathbf{z}^T H &= \mathbf{z}^T - \gamma\mathbf{v}^T, \qquad \gamma = \frac{1}{\beta}\mathbf{z}^T\mathbf{v} \end{aligned}$$

Un'ulteriore ottimizzazione riguarda l'algoritmo QR: è opportuno, prima di applicare il metodo, applicare alla matrice una serie di trasformazioni per passarla in *forma di Hessenberg*

$$\tilde{A} = \begin{bmatrix} * & * & \cdots & \cdots & * \\ * & * & \ddots & \ddots & * \\ 0 & * & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * \end{bmatrix}$$

Per passare in forma di Hessenberg sono sufficienti $n-2$ opportune trasformazioni di Householder, applicate simultaneamente alla destra e alla sinistra di A, in modo da mantenere le matrici della successione unitariamente simili.

Data una matrice in forma di Hessenberg, l'algoritmo QR dovrà annullare solo gli $n-1$ elementi della sottodiagonale, accelerando in questo modo la convergenza.

4.4 Criteri di stop per l'algoritmo QR

Considerata una matrice A quadrata di dimensione n, preventivamente portata in forma di Hessenberg, un criterio di stop utilizzabile per l'algoritmo QR è, dato τ come parametro di tolleranza

$$|a_{i+1,i}| \leq \tau, \qquad i = 1, \ldots, n-1 \tag{12}$$

Introduciamo anche un altro criterio di stop, alternativo a quello della formula 12

$$\left| a_{ii}^{(k)} - a_{ii}^{(k-1)} \right| \leq \tau, \qquad i = 1, \ldots, n \tag{13}$$

La formula 13 dà un diverso significato alla tolleranza τ. Nel primo caso, essa è una misura di quanto la matrice $A^{(k)}$ si avvicina alla forma triangolare superiore. Nel secondo caso, stiamo ipotizzando che la differenza nei valori tra due iterate non possa che diminuire, e quindi τ viene utilizzata direttamente per garantire una certa precisione per gli autovalori trovati. Verificheremo nei test il valore di questa ipotesi.

5 Implementazione

Abbiamo implementato in *Matlab*® la fattorizzazione QR di Householder, il passaggio in forma di Hessenberg, la moltiplicazione ottimizzata di una matrice generica per una matrice elementare di Householder (sia a destra sia a sinistra, e anche nel caso sia necessario orlare la matrice H) e infine l'algoritmo QR, mostrato nel listato 3. La funzione `cn2_qreig` prende quattro parametri: la matrice `A` di cui calcolare gli autovalori, `toltype`, il cui valore (1 o 2) permette di

Algorithm 3 Algoritmo QR, `cn2_qreig(...)`

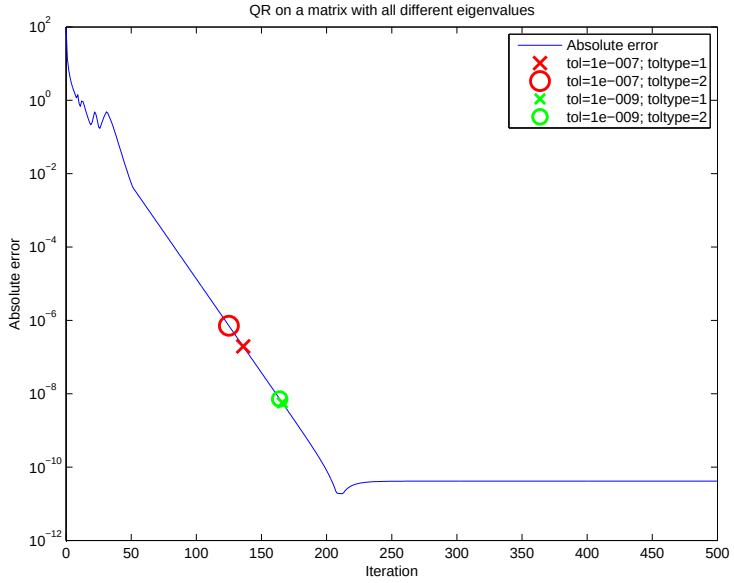
```
function [lambda,k] = cn2_qreig(A,toltype,tol,maxit)
    lambda = nan(size(A,1),maxit+1);
    A = cn2_hess(A);
    lambda(:,1) = diag(A);
    for k = 1:maxit
        [Q,R] = cn2_qr(A);
        A = R*Q;
        lambda(:,k+1) = diag(A);
        if checktol(toltype,diag(A,-1),lambda(:,k+1),lambda(:,k),tol)
            break
        end
    end
end

function val = checktol(toltype,ud,lcur,lpre,tol)
    if toltype == 1
        val = all(abs(ud)<=tol);
    end
    if toltype == 2
        val = all(abs(lcur-lpre)<=tol);
    end
end
```

scegliere il criterio col quale utilizzare il parametro di tolleranza `tol` (se `toltype = 1`, si utilizza il criterio 12, altrimenti il criterio 13) e infine `maxit` che indica il numero massimo di iterazioni da effettuare. Restituisce `lambda`, che contiene la stima degli autovalori per ciascuna iterata, e `k`, che è l'indice dell'ultima iterazione effettuata (la stima iniziale ha `k = 0`). Nel listato è possibile osservare come prima di cominciare le iterazioni la matrice `A` venga passata in forma di Hessenberg per una convergenza più veloce del metodo.

Successivamente abbiamo implementato una versione modificata dell'algoritmo QR (che abbiamo chiamato `cn2_eig`) che si può utilizzare anche in presenza di autovalori multipli o complessi coniugati, e che migliora le prestazioni in presenza di autovalori difettivi. L'approccio utilizzato è semplice: se nella diagonale principale è presente un "blocchetto 2x2", che chiamiamo B_i ($i = 1, ...n-1$), in cui il valore b_{21} risulti maggiore di un valore di tolleranza τ_{zero}, allora per i due autovalori corrispondenti (λ_i e λ_{i+1}) viene applicato il calcolo polinomiale "algebrico"; in caso contrario, vengono utilizzati direttamente i valori presenti nella diagonale. Come criterio di stop viene utilizzato soltanto 13.

Figura 7: Andamento dell'errore assoluto per il metodo `cn2_qreig(...)` su una matrice con autovalori distinti in modulo



6 Test e conclusioni

Per tutti i test sono state utilizzate matrici di dimensione 9. Il numero massimo di iterazioni è stato fissato a 500 (vedremo come tale valore sia largamente sufficiente per queste matrici). τ_{zero} è stato posto pari a 10^{-5}. Come criterio di comparazione si è considerato sempre l'*errore assoluto massimo*, calcolato come la massima differenza tra il valore assoluto dell'autovalore calcolato e il valore assoluto dell'autovalore esatto. Chiamando $\mathbf{\Lambda}$ il *vettore* degli autovalori e $\mathbf{\Lambda}^{(k^*)}$ il *vettore* degli autovalori calcolati, l'errore è dunque

$$E = \left\| \left| \mathbf{\Lambda}^{(k^*)} \right| - |\mathbf{\Lambda}| \right\|_\infty$$

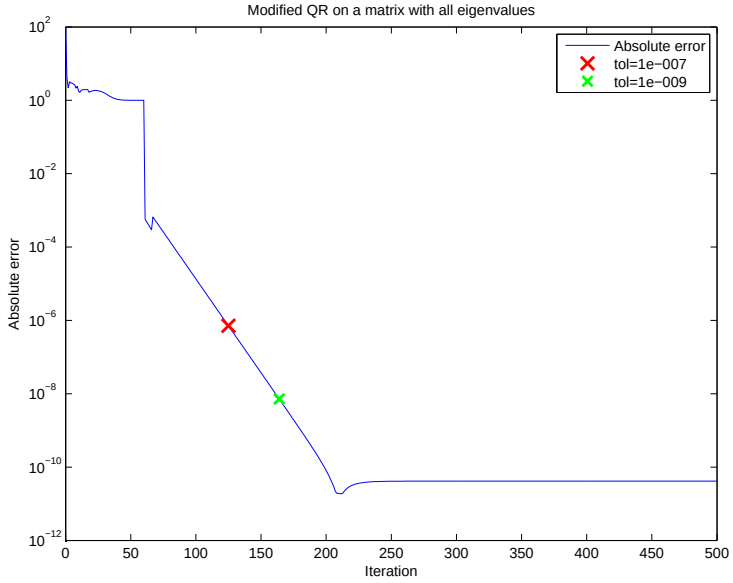
In cui, con abuso di notazione, vale:

$$|\mathbf{\Lambda}| = (|\lambda_1|, |\lambda_2|, ..., |\lambda_n|)$$

Tutti i test sull'algoritmo QR classico sono stati effettuati utilizzando sia il criterio di stop 12(crocette) sia 13 (cerchi). Entrambi i criteri di stop sono stati utilizzati con due valori di tolleranza τ (`tol` in Matlab).

Il primo test è stato effettuato su una matrice casuale con autovalori $(1, 2, \ldots, 9)$. Essendo tutti distinti in modulo e reali, l'algoritmo QR classico si comporta già molto bene, come si nota in Figura 7. Si nota come l'ipotesi fatta sul criterio 13 risulti esatta: il valore di τ risulta sempre molto vicino al valore dell'errore

Figura 8: Andamento dell'errore assoluto per il metodo `cn2_eig(...)` su una matrice con autovalori distinti in modulo



commesso. Le prestazioni sono dunque simili a quelle ottenute utilizzando il criterio 12.

L'andamento metodo QR modificato è visibile in Figura 8. L'andamento dell'errore presenta una vistosa discontinuità: è il superamento della soglia τ_{zero}.

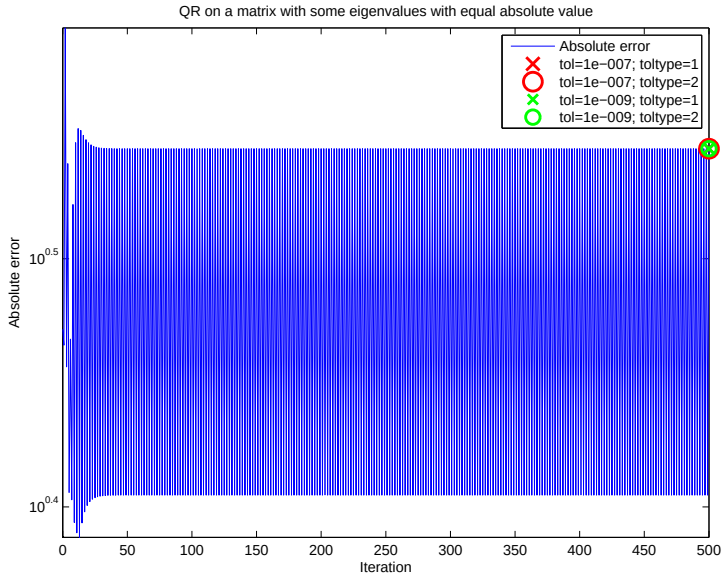
È interessante notare come il valore dell'errore assoluto si attesta dopo 200-300 iterazioni su una soglia da cui non si discosta più. I dati numerici di confronto tra i due metodi (e il metodo `eig` di Matlab) sono mostrati in Tabella 8. L'algoritmo `eig` di Matlab ha prestazioni (e velocità) di gran lunga migliori.

Il secondo test è stato effettuato su una matrice con autovalori con lo stesso valore assoluto: $(1, 2, 3, 4, 5, -4, -3, -2, -1)$. I risultati del metodo QR tradizionale e modificato sono mostrati nelle Figure 9 e 10. Come si vede, il metodo tra-

Tabella 8: Confronto tra i vari metodi per una matrice con autovalori distinti

Metodo	`tol`	`toltype`	`k`	Errore	Tempo
`cn2_qreig`	10^{-7}	1	147	$5,330 \cdot 10^{-8}$	785 ms
		2	125	$7,114 \cdot 10^{-7}$	663 ms
	10^{-9}	1	177	$1,550 \cdot 10^{-9}$	934 ms
		2	164	$7,191 \cdot 10^{-9}$	891 ms
`cn2_eig`	10^{-7}	-	125	$7,114 \cdot 10^{-7}$	673 ms
	10^{-9}	-	164	$7,191 \cdot 10^{-9}$	871 ms
`eig`	-	-	-	$5,988 \cdot 10^{-11}$	0,2 ms

Figura 9: Andamento dell'errore assoluto per il metodo `cn2_qreig(...)` in presenza di autovalori con lo stesso modulo



dizionale non può essere utilizzato in questo caso, mentre il metodo modificato converge. I risultati comparati sono mostrati in Tabella 9.

Il test successivo mostra il comportamento dei metodi su una matrice con autovalori multipli *non* difettivi: $(1, 2, 3, 4, 5, 4, 3, 2, 1)$. In questo caso, il metodo QR tradizionale converge, ma inizialmente presenta vistose oscillazioni, ha una convergenza molto più lenta e non raggiunge una precisione elevata (Figura 11). Il metodo QR modificato, invece, converge rapidamente e la precisione è dello stesso ordine di τ o migliore (Figura 12). Il confronto in Tabella 10.

Ora mostriamo i risultati su una matrice con autovalori difettivi: $(1, 2, 3, 4, 4, 4, 5, 6, 7)$. 4 ha molteplicità algebrica 3 e geometrica 1. In questo caso, come si può vedere

Tabella 9: Confronto tra i vari metodi in presenza di autovalori con lo stesso modulo

Metodo	`tol`	`toltype`	`k`	Errore	Tempo
`cn2_qreig`	10^{-7}	1	500	$3,886$	2635 ms
		2	500	$3,886$	2844 ms
	10^{-9}	1	500	$3,886$	2788 ms
		2	500	$3,886$	2648 ms
`cn2_eig`	10^{-7}	-	76	$7,968 \cdot 10^{-8}$	465 ms
	10^{-9}	-	97	$3,774 \cdot 10^{-10}$	526 ms
`eig`	-	-	-	$9,504 \cdot 10^{-14}$	0,1 ms

Figura 10: Andamento dell'errore assoluto per il metodo `cn2_eig(...)` in presenza di autovalori con lo stesso modulo

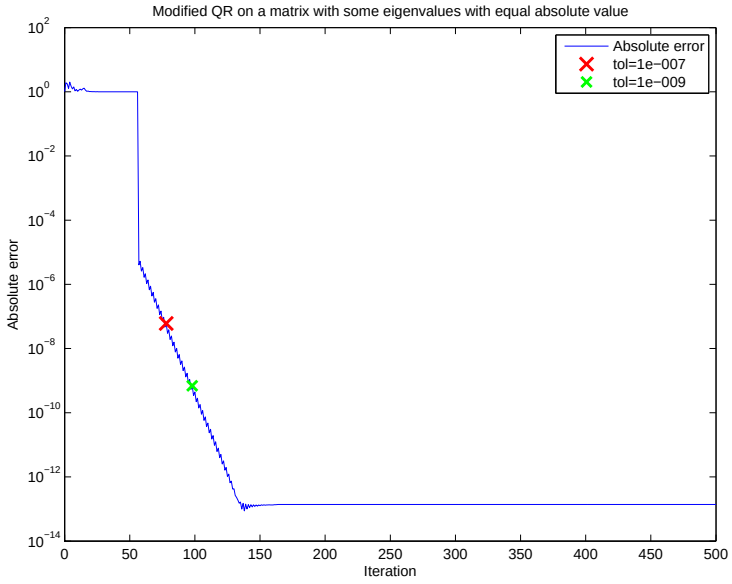


Figura 11: Andamento dell'errore assoluto per il metodo `cn2_qreig(...)` in presenza di autovalori con molteplicità 2

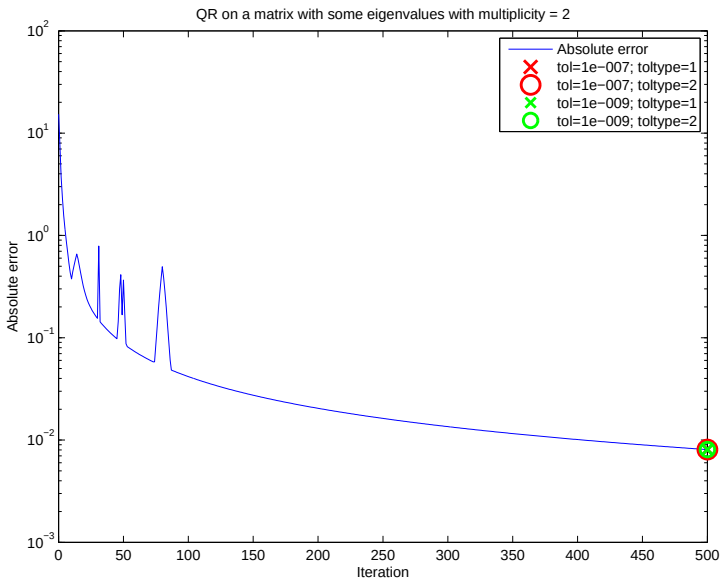


Figura 12: Andamento dell'errore assoluto per il metodo `cn2_eig(...)` in presenza di autovalori con molteplicità 2

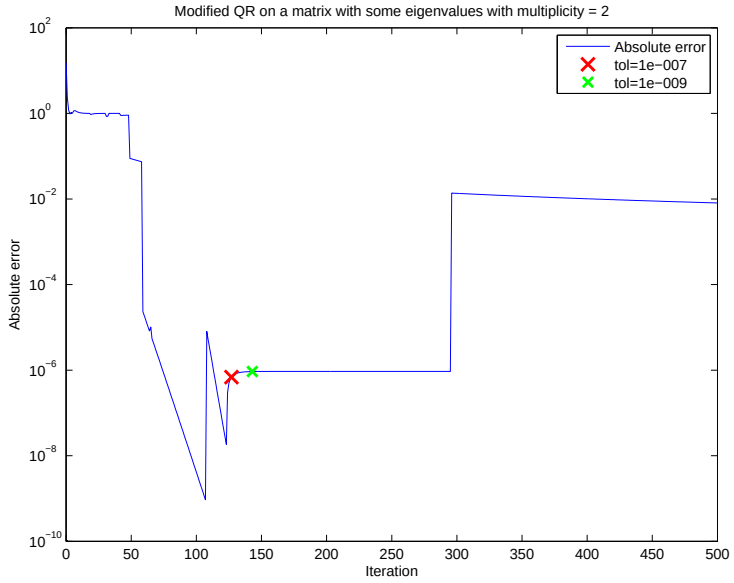
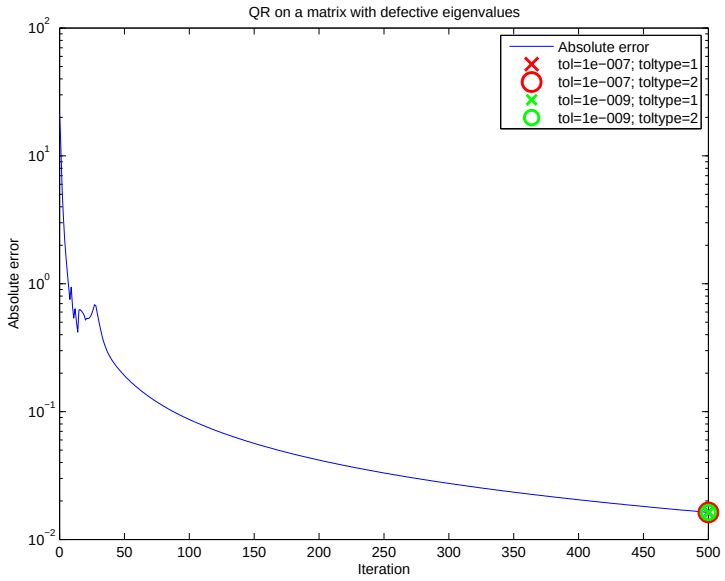


Tabella 10: Confronto tra i vari metodi in presenza di autovalori con molteplicità 2

Metodo	`tol`	`toltype`	`k`	Errore	Tempo
`cn2_qreig`	10^{-7}	1	500	$8,062 \cdot 10^{-3}$	2635 ms
		2	500	$8,062 \cdot 10^{-3}$	2844 ms
	10^{-9}	1	500	$8,062 \cdot 10^{-3}$	2788 ms
		2	500	$8,062 \cdot 10^{-3}$	2648 ms
`cn2_eig`	10^{-7}	-	123	$5,094 \cdot 10^{-8}$	709 ms
	10^{-9}	-	145	$6,803 \cdot 10^{-12}$	799 ms
`eig`	-	-	-	$6,448 \cdot 10^{-13}$	0,1 ms

Figura 13: Andamento dell'errore assoluto per il metodo `cn2_qreig(...)` in presenza di autovalori difettivi



nelle Figure 13 e 14, l'algoritmo QR converge, ma molto lentamente e l'algoritmo QR modificato converge con un ordine di grandezza superiore. Questo scenario è il peggiore anche per l'algoritmo `eig` di Matlab. La comparativa è nella Tabella 11.

L'ultimo test riguarda una matrice con autovalori complessi e coniugati. L'algoritmo QR classico ha un andamento decisamente fluttuante, evidente sintomo della non convergenza. L'algoritmo modificato converge nel giro di 100 iterazioni, come negli altri casi, mantenendo un errore assoluto basso. I risultati nelle Figure 15 e 16 e nella Tabella 12.

Tabella 11: Confronto tra i vari metodi in presenza di autovalori difettivi

Metodo	`tol`	`toltype`	`k`	Errore	Tempo
`cn2_qreig`	10^{-7}	1	500	$1,627 \cdot 10^{-2}$	2726 ms
		2	500	$1,627 \cdot 10^{-2}$	2715 ms
	10^{-9}	1	500	$1,627 \cdot 10^{-2}$	2748 ms
		2	500	$1,627 \cdot 10^{-2}$	3016 ms
`cn2_eig`	10^{-7}	-	500	$8,127 \cdot 10^{-3}$	2721 ms
	10^{-9}	-	500	$8,127 \cdot 10^{-3}$	2942 ms
`eig`	-	-	-	$1,762 \cdot 10^{-5}$	0,1 ms

Figura 14: Andamento dell'errore assoluto per il metodo `cn2_eig(...)` in presenza di autovalori difettivi

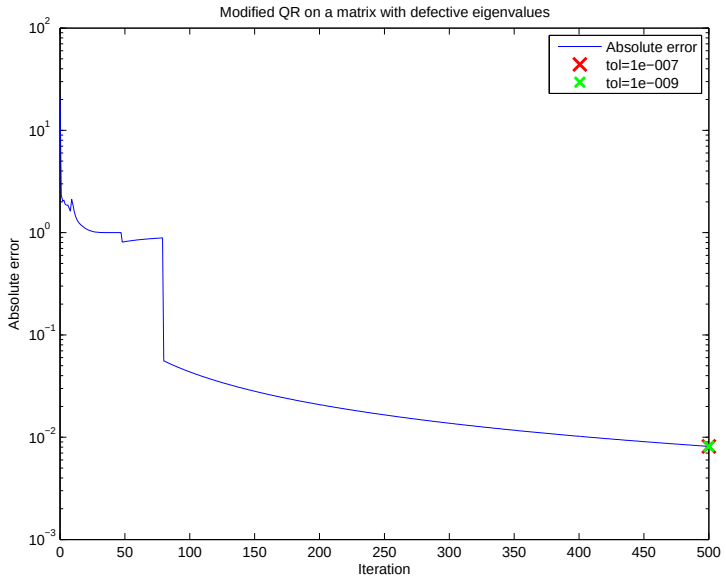


Figura 15: Andamento dell'errore assoluto per il metodo `cn2_qreig(...)` in presenza di autovalori complessi e coniugati

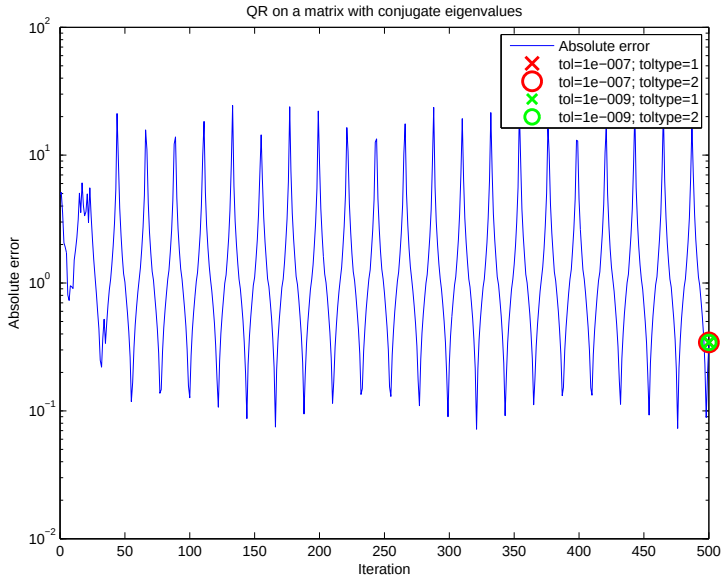


Figura 16: Andamento dell'errore assoluto per il metodo `cn2_eig(...)` in presenza di autovalori complessi e coniugati

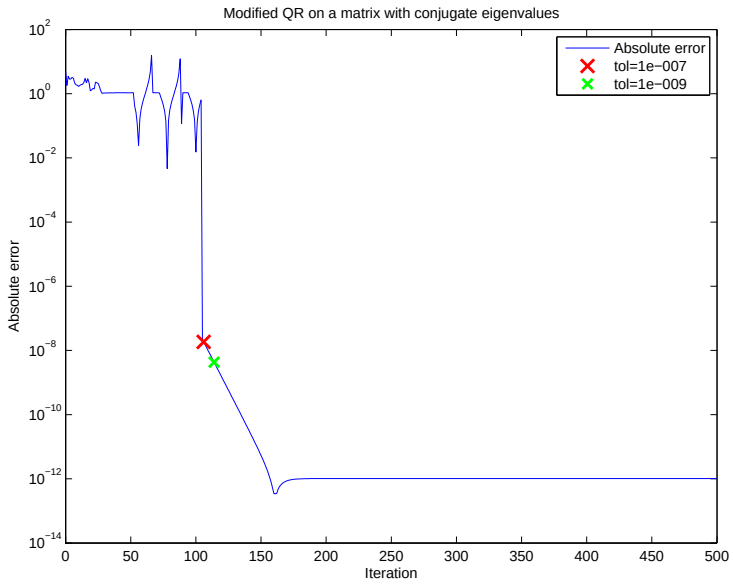


Tabella 12: Confronto tra i vari metodi in presenza di autovalori complessi e coniugati

Metodo	`tol`	`toltype`	`k`	Errore	Tempo
`cn2_qreig`	10^{-7}	1	500	$0,354$	2122 ms
		2	500	$0,354$	2123 ms
	10^{-9}	1	500	$0,354$	2147 ms
		2	500	$0,354$	2132 ms
`cn2_eig`	10^{-7}	-	108	$1,121 \cdot 10^{-8}$	544 ms
	10^{-9}	-	113	$4,522 \cdot 10^{-9}$	493 ms
`eig`	-	-	-	$5,005 \cdot 10^{-12}$	0,1 ms

6.1 Conclusioni

Risulta evidente dai test come il metodo QR mostri una convergenza estremamente lenta, e non può quindi essere utilizzato da solo per calcolare velocemente gli autovalori di una matrice, specie se molto grande: la funzione `eig` di Matlab ha sempre mantenuto prestazioni almeno 5000 volte superiori. Questo può essere dovuto in parte all'implementazione con un linguaggio ad alto livello, in parte alla mancanza di particolari accorgimenti per accelerare la convergenza, ad esempio non è stato fatto uso di *shift*.

Inoltre, l'algoritmo QR si dimostra estremamente instabile, lento e impreciso se gli autovalori non rispettano la condizione sul modulo. In particolare non è utilizzabile con autovalori uguali in modulo ma non in segno, e con autovalori complessi e coniugati. Abbiamo però visto che una semplice modifica renda l'algoritmo utilizzabile anche in questi casi.