Progetto per l'esame di Calcolo Scientifico

Renato Budinich

July 19, 2013

1 Descrizione del progetto

Il metodo di Lanczos, applicato a una matrice hermitiana $A \in \mathbb{C}^{n\times n}$ con autovalori $\lambda_1 \leq \ldots \leq \lambda_n$, fornisce al generico passo $k = 1, \ldots, n$ una matrice tridiagonale $T_k \in \mathbb{C}^{k\times k}$ con autovalori $\mu_1^{(k)} \leq \ldots \leq \mu_k^{(k)}$. Gli autovalori estremi di T_k convergono a quegli estremi di A, con velocità via via peggiore come ci si allontana dalle estremità ($\mu_{k-1}^{(k)}$ e $\mu_2^{(k)}$ convergono a λ_{n-1} e λ_2 più lentamente di quanto $\mu_k^{(k)}$ e $\mu_1^{(k)}$ convergano a λ_n e λ_1).

Lo scopo del progetto era verificare questa convergenza sulla matrice

$$A \stackrel{def}{=} \begin{bmatrix} 5 & 4 & 1 & & 0 \\ 4 & 6 & \ddots & \ddots & \\ 1 & \ddots & \ddots & \ddots & 1 \\ & \ddots & \ddots & 6 & 4 \\ 0 & & 1 & 4 & 5 \end{bmatrix}$$

che è il quadrato della matrice

$$\begin{bmatrix} 2 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & 2 \end{bmatrix}$$

e della quale si conosce quindi l'espressione esplicita per gli autovalori

$$\lambda_j = (2 + 2\cos(\pi \frac{n+1-j}{n+1}))^2 \quad j = 1, 2, \ldots, n \tag{1.0.1}$$

che rende facile la verifica del metodo.

2 L'algoritmo e l'implementazione

Il metodo di Lanczos fornisce delle relazioni ricorsive per gli autovettori q_k, $k = 1, \ldots, n$ di A, per il vettore diagonale e per quello sopradiagonale (rispettivamente α e β) di T_n:

$$\begin{cases} q_{k+1} & = \frac{Aq_k - \alpha_k q_k - \beta_{k-1} q_{k-1}}{\beta_k} \\ q_1 & = q \\ q_0 & = 0 \end{cases} \qquad \begin{cases} \alpha_k & = q_k \cdot Aq_k \\ \beta_k & = ||Aq_k - \alpha_k q_k - \beta_{k-1} q_{k-1}|| \end{cases} \quad k = 1, \ldots, n \tag{2.0.2}$$

dove q è un vettore qualunque di norma 1, e nel caso β_k sia uguale a 0, per q_{k+1} basta scegliere un vettore ortogonale a $q_1, \ldots, q_k$. Gli autovalori della matrice

$$T_n = \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ 0 & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

così ottenuta sono in teoria esattamente gli autovalori di A; in realtà a causa degli errori numerici saranno delle approssimazioni. Solitamente il metodo viene usato come metodo iterativo, sfruttando la proprietà di convergenza già accennata: ci si ferma all'iterazione K e si usano gli autovalori estremi di T_K come approssimazione di quelli di A.

Dalle relazioni (2.0.2) si ricava l'algoritmo (1), il cui costo dominante è la moltiplicazione matrice per vettore Aq_k, che nel nostro caso ha complessità $\mathcal{O}(n)$, quindi l'algoritmo globalmente ha complessità temporale $\mathcal{O}(nK)$, dove K è il numero di iterazioni effettuate.

La complessità in memoria dell'algoritmo è $\mathcal{O}(nK)$, perchè bisogna memorizzare la matrice Q degli autovettori; siccome questi non interessavano, ho usato invece l'algoritmo (2) che ha complessità in memoria $\mathcal{O}(K)$.

Algorithm 1

$q_0 \leftarrow 0$
$q_1 \leftarrow$ vettore random di norma 1
for $k = 1, \ldots, K$ **do**
    $t \leftarrow Aq_k$
    $\alpha_k \leftarrow q_k \cdot t$
    $r \leftarrow t - \alpha_k q_k - \beta_{k-1} q_{k-1}$
    if $k < n$ **then**
        $\beta_k \leftarrow ||r||$
        if $\beta_k \neq 0$ **then**
            $q_{k+1} \leftarrow \frac{r}{\beta_k}$
        else
            break
        end if
    end if
end for

Algorithm 2

$w \leftarrow 0$
$v \leftarrow$ vettore random di norma 1
for $k = 1, \ldots, K$ **do**
    $t \leftarrow Av$
    $\alpha_k \leftarrow v \cdot t$
    $r \leftarrow t - \alpha_k v - \beta_{k-1} w$
    if $k < n$ **then**
        $\beta_k \leftarrow ||r||$
        if $\beta_k \neq 0$ **then**
            $z \leftarrow \frac{r}{\beta_k}$
            $w \leftarrow v$
            $v \leftarrow z$
        else
            break
        end if
    end if
end for

2.1 Codice sorgente

Segue l'implementazione in Fortran dell'algoritmo. Ho usato la subroutine DSTEVR di LAPACK per calcolare ad ogni passo gli autovalori di T_k. Nell'algoritmo ad ogni iterazione controllo se $\beta(k) < \epsilon$, costante fissata inizialmente, per evitare problemi di instabilità numerica; si veda ad esempio la figura (1), dove ϵ è stato fissato abbastanza piccolo da non entrare mai nel ramo dell' `if` che interrompe il ciclo iterativo - vengono svolte quindi tutte le K iterazioni, e si vede che da una certa iterazione in poi gli errori di alcuni autovalori più interni cominciano ad aumentare. Invece ponendo $\epsilon = 1$ l'iterazione si ferma prima di tale fenomeno (che ho riscontrato sperimentalmente in molte altre istanze) fornendo un'approssimazione migliore di quegli autovalori.

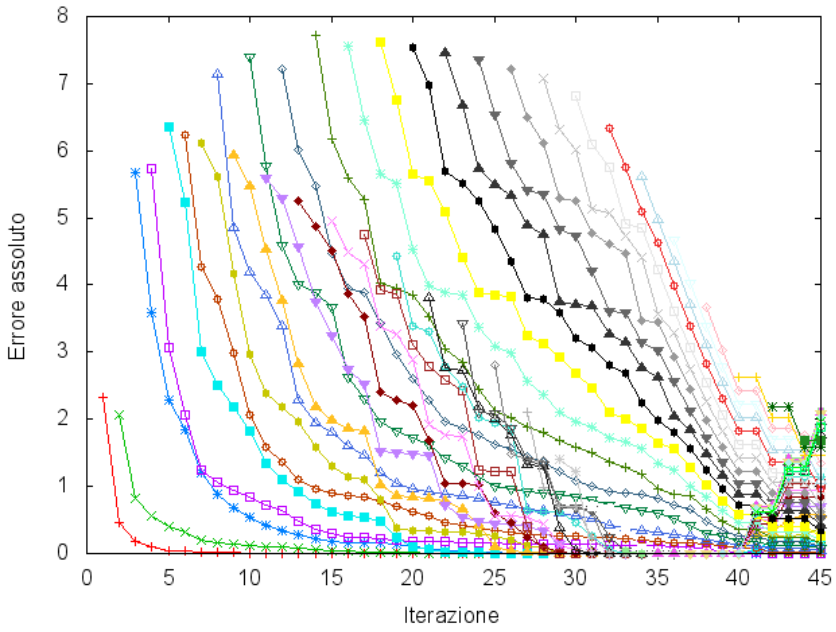

Figure 1: errori per $n = 45, K = 45, \epsilon = 10^{-3}$

3 Utilizzo del programma e interpretazione dell'output

Il programma chiede in input n e il numero di iterazioni K da effettuare, e fornisce in output tre file:

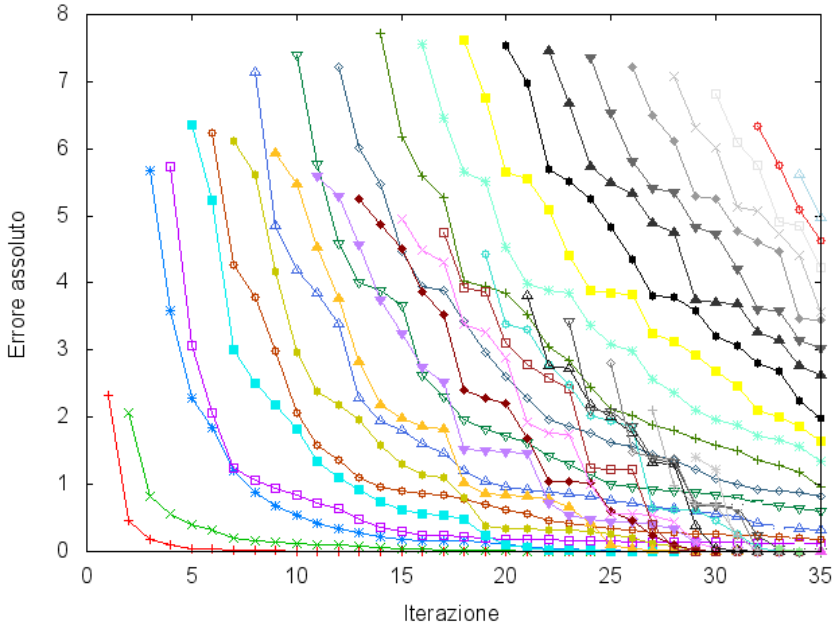

Figure 2: errori per $n = 45, K = 45, \epsilon = 1$

correcteigv.txt contiene gli n autovalori di A in ordine crescente, uno per riga, calcolati usando la (1.0.1)

eigv.txt c'è una riga per ogni iterazione effettuata; la prima colonna è l'indice k dell'iterazione, quelle successive sono gli autovalori di T_k, ordinati in questo modo: $\mu_k^{(k)}, \mu_1^{(k)}, \mu_{k-1}^{(k)}, \mu_2^{(k)}, \ldots$.

errors.txt anche qui la prima colonna è l'indice k dell'iterazione, quelle successive invece sono le differenze in valore assoluto tra gli autovalori corretti e quelli approssimati, con lo stesso ordine del file precedente, ovvero: $|\lambda_n - \mu_k^{(k)}|, |\lambda_1 - \mu_1^{(k)}|, |\lambda_{n-1} - \mu_{k-1}^{(k)}|, |\lambda_2 - \mu_2^{(k)}|, \ldots$

Questi file vengono poi utilizzati dagli script `gnuplot plot-eigv.gp` e `plot-err.gp` per produrre dei grafici rispettivamente della convergenza e degli errori commessi, al crescere delle iterazioni.

Per eseguire il programma e visualizzare il grafico degli errori si può ad esempio dare il commando:

```
$ gfortran lanczos.f90 -llapack -o lanczos && ./lanczos && gnuplot plot-err.gp
```

Per il grafico della convergenza degli autovalori (che è utile solo per valori piccoli di n) basta sostituire `plot-eigv.gp` a `plot-err.gp` nel commando sopra.

Nelle figure sono riportati alcuni esempi dei grafici prodotti per diversi valori di n e K.

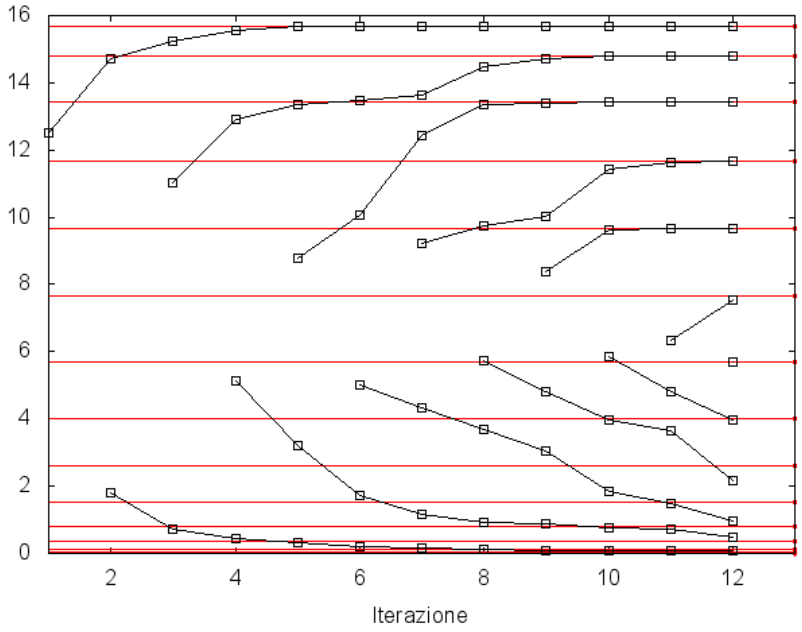

Figure 3: convergenza degli autovalori per $n = K = 15$

4 Commenti

Dai grafici degli errori si nota come la convergenza sia effettivamente più rapida sugli autovalori estremi e come le velocità di convergenza siano accoppiate, tra il primo e l'ultimo autovalore, il secondo e il penultimo e via così.

Per $n = 10^6$, $K = 200$ il mio computer desktop (processore intel i5 2500k, 8gb di RAM) impiega

```
$ time
```

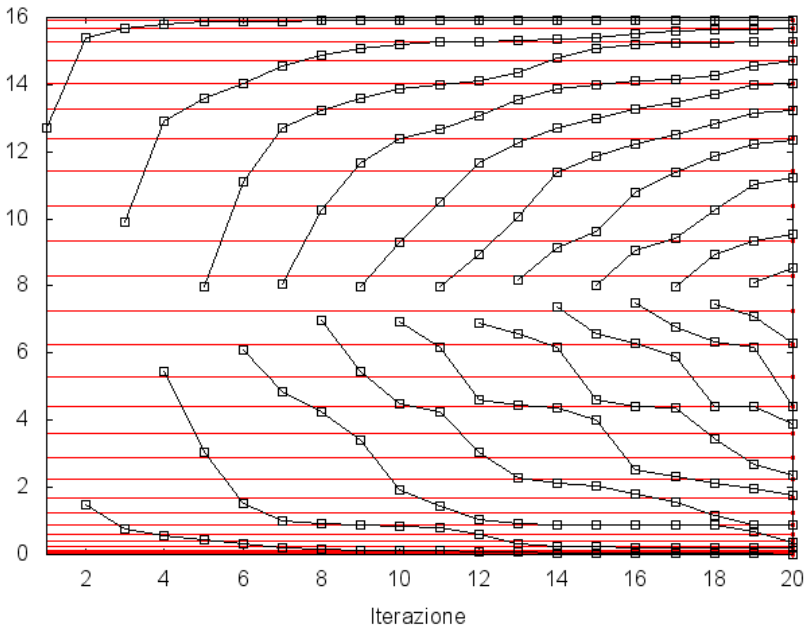

Figure 4: convergenza degli autovalori per $n = 30, K = 20$

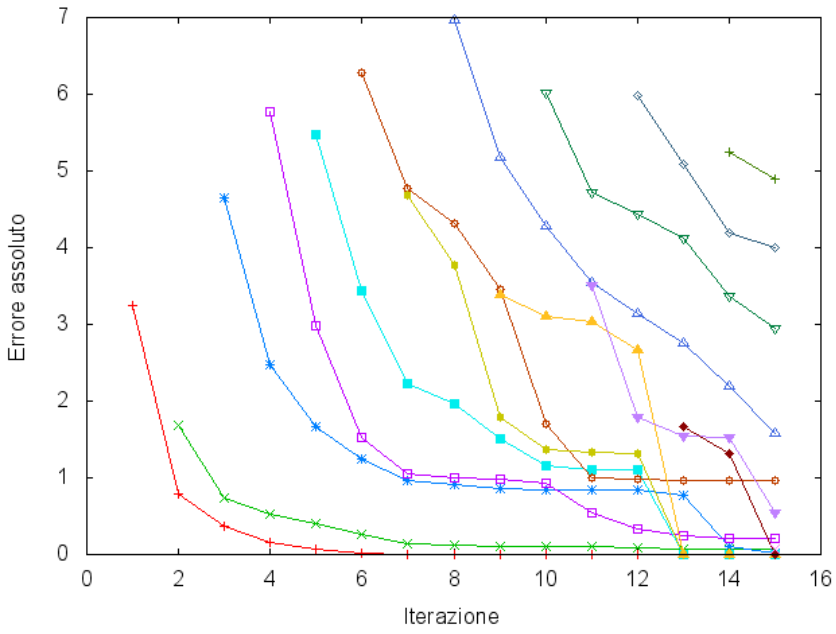

Figure 5: errori per $n = 20, K = 20$

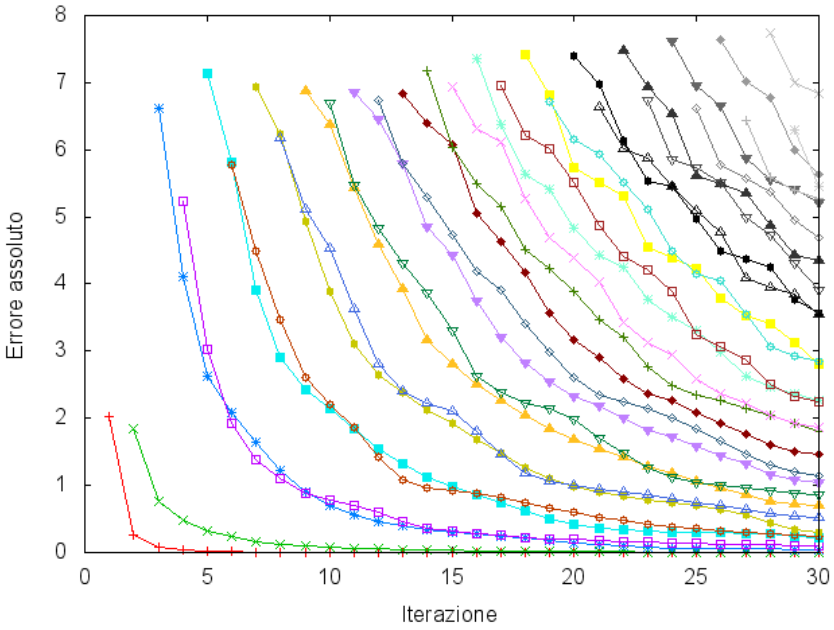

Figure 6: errori per $n = 100, K = 30$

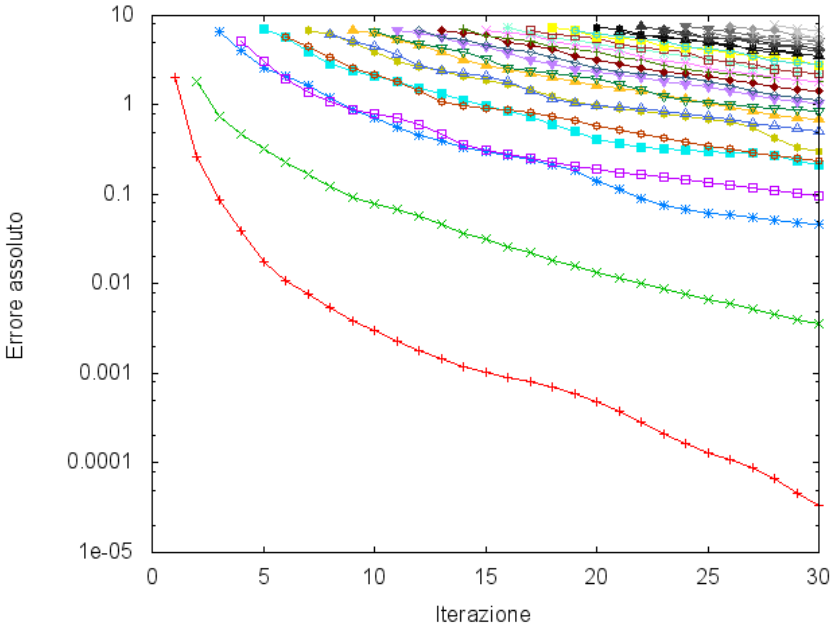

Figure 7: errori per $n = 100, K = 30$ in scala logaritmica

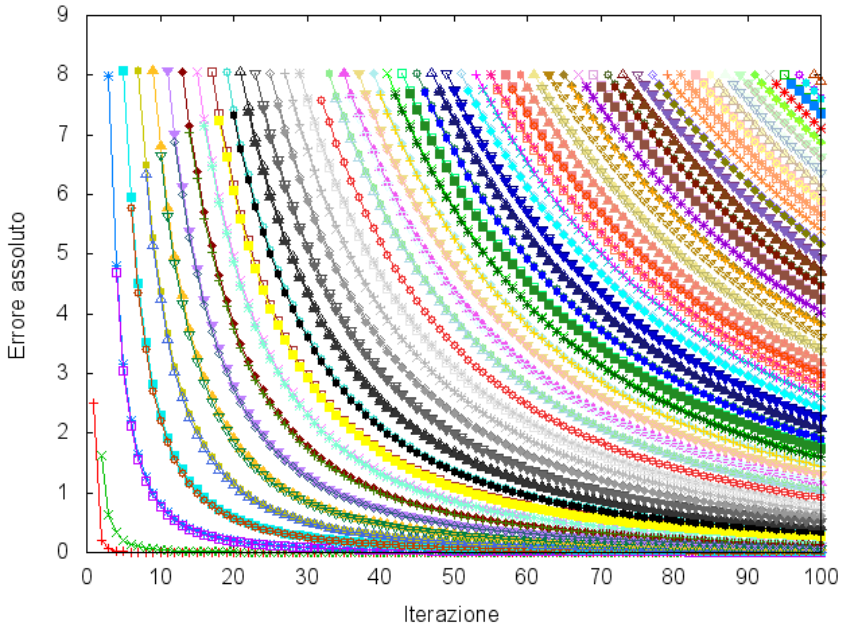

Figure 8: errori per $n = 10^6, K = 100$

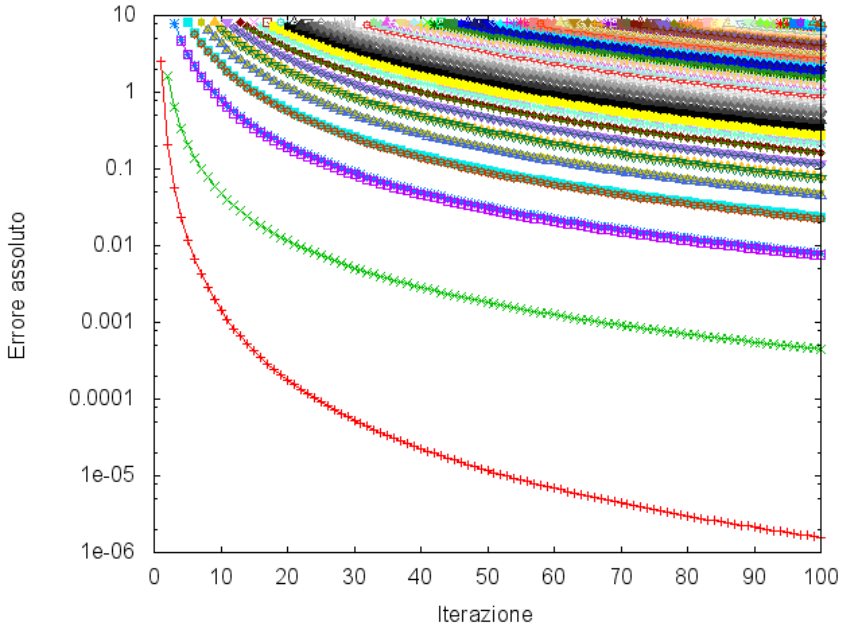

Figure 9: errori per $n = 10^6, K = 100$ in scala logaritmica