UNIVERSITÀ DEGLI STUDI DI CAGLIARI
FACOLTÀ DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Elettrica ed Elettronica

Risoluzione di sistemi lineari tramite il metodo Global Lanczos

Seminario di Matematica Applicata Avanzata

Edoardo Cannas - Anna Concas

Anno Accademico 2015/2016

Contents

Introduzione

La presenza di numerose applicazioni pratiche che prevedono la risoluzione di sistemi lineari con molteplici termini noti, ha portato nei recenti anni di ricerca nel campo della Matematica Applicata all'elaborazione di algoritmi che sfruttano la decompozione a blocchi di Lanczos. Tra di essi va inserito il metodo di decomposizione globale a blocchi di Lanczos, o più comunemente detto, Global Lanczos.
In questa tesina svilupperemo un algoritmo per la risoluzione di sistemi lineari, basato su questa decomposizione.
Ne elaboreremo due varianti, differenti tra loro per l'uso di una tecnica di riortogonalizzazione, che dovrebbe garantire una maggiore precisione della soluzione ricercata. Analizzeremo le prestazioni di entrambi, cercando, ove possibile, di mettere in luce vantaggi e svantaggi di entrambe le soluzioni.
Nel primo capitolo ci occuperemo di introdurre la tecnica di decomposizione globale a blocchi di Lanczos, mostrandone le caratteristiche rispetto alla sua versione standard, e illustrando una sua possibile implementazione sotto forma di algoritmo.
Nel secondo capitolo, applicheremo tale tecnica alla risoluzione di un sistema lineare $AX = B$ con:

- A matrice quadrata e simmetrica, di dimensione n;
- X e B matrici $n \times k$, con k dimensione del blocco.

Sostanzialmente, si tratta di risolvere un insieme di k sistemi lineari che condividono la stessa matrice dei coefficienti A.
Nel terzo capitolo, analizzeremo le prestazioni del nostro algoritmo in termini di comparazione di residui ed errori normalizzati illustrando i test effettuati. Esamineremo, inoltre, l'effetto della tecnica di riortogonalizzazione sulle prestazioni globali dell'algoritmo in termini di tempo di esecuzione e numero di iterazioni dello stesso (pertanto, di precisione della soluzione).

Chapter 1

Global Lanczos

1.1 Global Lanczos decomposition

La *global Lanczos decomposition* è una tecnica recente sviluppata a partire dalla decomposizione a blocchi di Lanczos. Una trattazione esaustiva può essere consultata in [2] e [3].
Partendo da una matrice A simmetrica di dimensione n, e da una matrice W di dimensione $n \times k$, con $k \ll n$, è possibile ottenere una decomposione di A del tipo:

$$A[V_1, V_2, \cdots, V_l] = [V_1, V_2, \cdots, V_l, V_{l+1}]T_{l+1,l} \tag{1.1}$$

ottenendo una matrice $\mathcal{V}$ i cui j-esimi blocchi $V_j \in \mathbb{R}^{n\times k}$ costituiscono una base ortonormale per il sottospazio di Krylov generato da A e W definito come

$$\mathcal{K}_l := \text{span}(W, AW, A^2W, \ldots, A^{l-1}W) \subseteq \mathbb{R}^{n\times k}.$$

.
La decomposizione 1.1 è ottenuta tramite una semplice iterazione di Lanczos a partire dalla matrice A e da una generica matrice W (chiamata anche blocco iniziale), con le uniche differenze che:

- il prodotto interno dello spazio è definito come $\langle W_1, W_2\rangle := \text{trace}(W_1^T W_2)$;
- la norma indotta dal prodotto scalare è quella di Frobenius, ossia

$$\|W_1\|_F := \langle W_1, W_1\rangle^{1/2}.$$

La matrice T_l è una matrice simmetrica e tridiagonale della forma:

$$T_l = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_l \\ & & & \beta_l & \alpha_l \end{bmatrix};$$

mentre la matrice $T_{l+1,l}$ è costituita dalla matrice tridiagonale T_l e da una riga di elementi nulli tranne l'ultimo pari a β_{l+1}, ovvero della forma

$$T_{l+1,l} = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_l \\ & & & \beta_l & \alpha_l \\ & & & & \beta_{l+1} \end{bmatrix}. \tag{1.2}$$

Osserviamo che la matrice T_l, pur di dimensione l pari al numero delle iterazioni dell'algoritmo, è decisamente inferiore in dimensione rispetto alla matrice A di partenza, e caratterizzata inoltre dal possedere degli autovalori che sono una buona approssimazione degli autovalori estremali di A.
Per quanto riguarda la matrice $\mathcal{V}$, è importante osservare che l'ortonormalità è richiesta solamente tra i generici blocchi componenti la base, e non tra le k colonne di ogni singolo blocco. Questo fa sì che il global Lanczos richieda uno sforzo computazionale minore rispetto alla generica decomposizione a blocchi di Lanczos, che invece impone l'ortonormalità anche tra le generiche colonne di ciascun blocco.
L'algoritmo 1, da noi elaborato facendo riferimento all'articolo [1], prende in ingresso un generico blocco W e considera la possibilità che alcuni coefficienti presentino un valore inferiore ad una soglia di tolleranza da noi fissata, portando ad una situazione cosìddetta di *breakdown*.
Il verificarsi di un breakdown nell'applicazione dell'algoritmo alla risoluzione di molteplici sistemi lineari, indica sostanzialmente che la soluzione trovata al j-esimo passo dell'algoritmo è esattamente la soluzione ricercata dell'insieme dei sistemi. Tratteremo questo caso più dettagliatamente nel secondo capitolo.

Algoritmo 1 Metodo decomposizione globale di Lanczos

Input: $A \in \mathbb{R}^{n\times n}$ simmetrica, $W \in \mathbb{R}^{n\times k}$
numero iterazioni algoritmo l, tau_β tolleranza breakdown
$\beta_1 = \|W\|_F, \quad V_1 = W/\beta_1$
for $j = 1, \ldots, l$
  $\tilde{V} = AV_j - \beta_j V_{j-1}, \quad \alpha_j = \langle V_j, \tilde{V}\rangle$
  $\tilde{V} = \tilde{V} - \alpha_j V_j$
  $\beta_{j+1} = \|\tilde{V}\|_F$
  if $\beta_{j+1} < \tau_\beta$ **then** uscita per breakdown **end**
  $V_{j+1} = \tilde{V}/\beta_{j+1}$
end for
Output: decomposizione globale di Lanczos (1.1)

Chapter 2

Global Lanczos per i sistemi lineari

2.1 Global Lanczos e problemi ai minimi quadrati

L'algoritmo di fattorizzazione global Lanczos appena esaminato, permette di sviluppare una variante a blocchi del metodo **GMRES** (*Generalized Minimal RESiduals*), utilizzato per la risoluzione di sistemi lineari tramite proiezioni in sottospazi di Krylov, nel caso di matrici dei coefficienti simmetriche.
Questo metodo, sostanzialmente, prevede la costruzione iterativa di una base ortonormale per un sottospazio di Krylov $\mathcal{K}_l$, con $l = 1, \ldots, n$, sul quale è possibile proiettare la soluzione di un sistema lineare $AX = B$ (già esaminato in precedenza, con A simmetrica e non singolare), risolvibile nel senso di un problema ai minimi quadrati.
Sia $S \subset H$ un sottospazio di dimensioni finite dello spazio di Hilbert H, con $\dim(S) = l$, prodotto interno $\langle v, w \rangle$, $\forall v, w \in H$, e norma indotta $\|w\| = \sqrt{\langle w, w \rangle}$.
Consideriamo il seguente

Lemma 2.1. *Supponiamo che $w_1, \ldots, w_l$ sia una base ortonormale per lo spazio S, e quindi sia $w = \sum_{j=1}^{l} y_j w_j \in S$ con $y_j = \langle w, w_j \rangle$.*
Allora $\|w\| = \|\mathbf{y}\|_2$, con $\mathbf{y} = (y_1, \ldots, y_l) \in \mathbb{R}^l$.

Proof.

$$\|w\|^2 = \langle w, w \rangle = \sum_{i,j=1}^{l} y_i y_j \langle w_i, w_j \rangle = \|\mathbf{y}\|_2^2. \tag{2.1}$$

□

Consideriamo le matrici $A \in \mathbb{R}^{n\times n}$, B e $X \in \mathbb{R}^{n\times k}$ $(k \ll n)$.
Considerando A simmetrica e non singolare, vogliamo risolvere il sistema $AX = B$. Realizziamo, a partire dalle matrici A e B, la parziale decomposizione globale di Lanczos, già vista in (1.1), la quale può essere scritta in forma più compatta come

$$A\mathcal{V}_l = \mathcal{V}_{l+1}\tilde{T}_{l+1,l} \tag{2.2}$$

dove:

- $\mathcal{V}_l = [V_1, \ldots, V_l], \quad V_i \in \mathbb{R}^{n\times k}, \quad V_1 = B/\|B\|$
- la matrice $\tilde{T}_{l+1,l} \in \mathbb{R}^{(l+1)k\times lk}$ può essere espressa come

$$T_{l+1,l} \otimes I_k \tag{2.3}$$

  dove $I_k \in \mathbb{R}^{k\times k}$ è la matrice identità e $T_{l+1,l} \in \mathbb{R}^{l+1,l}$ è la matrice (1.2).

Osserviamo che nella (2.3) il simbolo $\otimes$ rappresenta il prodotto di Kronecker. Esso è un prodotto tensoriale matriciale sempre eseguibile, a differenza del classico prodotto righe per colonne, definito come segue:

Definizione 2.1. Data A una matrice $m \times n$ e B una matrice $p \times q$ si ha che il loro prodotto di Kronecker $A \otimes B$ è una matrice di dimensione $mp \times nq$ definita a blocchi nel seguente modo:

$$A \otimes B = \begin{bmatrix} a_{11}B & \ldots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \ldots & a_{mn}B \end{bmatrix}.$$

Quindi nel caso in cui A e B abbiano dimensione 2×3 e 2×2 rispettivamente si ha che il loro prodotto di Kronecker è una matrice 4×6 scritta, esplicitando ogni termine, come:

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \end{bmatrix}.$$

Riferendoci al lemma precedente, assumiamo $H = \mathbb{R}^{n\times k}$, con prodotto interno $\langle V, W\rangle := \text{trace}(V^T W)$ e norma indotta quella di Frobenius $(\|V\|_F)$; consideriamo il sottospazio di Krylov K_l generato dalle matrici A e B, ovvero:

$$\mathcal{K}_l(A, B) := \text{span}(B, AB, A^2B, \ldots, A^{l-1}B) = \text{span}(V_1, \ldots, V_l) \subseteq \mathbb{R}^{n\times k}.$$

Assumiamo di proiettare X nel sottospazio di Krylov appena creato. Questo implica che

$$X = \sum_{j=1}^{l} V_j y_j = \mathcal{V}_l(\mathbf{y} \otimes I_k)$$

con $\mathbf{y} = (y_1, \ldots, y_l)^T \in \mathbb{R}^l$. Pertanto

$$\begin{aligned} \|AX - B\|_F &= \|A\mathcal{V}_l(\mathbf{y} \otimes I_k) - B\|_F = \|\mathcal{V}_{l+1}\tilde{T}_{l+1,l}(\mathbf{y} \otimes I_k) - B\|_F \\ &= \|\mathcal{V}_{l+1}(T_{l+1,l}\mathbf{y} \otimes I_k) - B\|_F = \|Z\|_F \end{aligned} \tag{2.4}$$

dove $Z = \mathcal{V}_{l+1}(T_{l+1,l}\mathbf{y} \otimes I_k) - B \in \mathcal{K}_{l+1}$, ed abbiamo usato la proprietà $(C \otimes D)(E \otimes F) = (CE \otimes DF)$.
Il lemma 1 mostra che $\|Z\|_F = \|\mathbf{z}\|_2$, con

$$z_i = \langle V_i, Z\rangle = \langle V_i, \mathcal{V}_{l+1}(T_{l+1,l}\mathbf{y} \otimes I_k) - B\rangle = u_i - \langle V_i, B\rangle$$

e $\mathbf{u} = (u_1, \ldots, u_{l+1})^T = T_{l+1,l}\mathbf{y}$. Dato che

$$\langle V_i, B\rangle = \begin{cases} \|B\|_F, & i = 1 \\ 0, & i \neq 1 \end{cases}$$

ponendo $\beta_1 = \|B\|_F$, otteniamo

$$\min_{X \in \mathcal{K}_l} \|AX - B\|_F = \min_{\mathbf{y} \in \mathbb{R}^l} \|T_{l+1,l}\mathbf{y} - \beta_1 \mathbf{e}_1\|_2. \tag{2.5}$$

Si può notare come $\mathbf{y}^{(l)}$ sia la soluzione ai minimi quadrati del problema alla destra dell'equazione precedente, ed otteniamo che la l-esima approssimazione della soluzione $X^{(l)} = \mathcal{V}_l(\mathbf{y}^{(l)} \otimes I_k)$.

2.2 glanczosLS.m

Partendo dalla base teorica descritta nella sezione precedente, abbiamo sviluppato un algoritmo in ambiente di programmazione MATLAB, implementato in una funzione **glancozLs.m**, per la risoluzione di sistemi lineari nella forma vista di $AX = B$. L'algoritmo è stato sviluppato in due varianti, con e senza riortogonalizzazione dei blocchi componenti la base del sottospazio, ed è illustrato nell'algoritmo 2.
Esso prevede una semplice iterazione di Lanczos per la creazione di una base ortonormale per un sottospazio di Krylov a partire dalle matrici A e B, sul quale proiettare la soluzione X. Questo compito è eseguito nelle righe a partire dalla 4 alla 8, e dalla 13 alla 24 nel ciclo principale.

Algoritmo 2 Global Lanczos for Linear Systems

1: **Input:** $A \in \mathbb{R}^{n\times n}$ simmetrica, $B \in \mathbb{R}^{n\times k}$
2: tol tolleranza soluzione, tau_β tolleranza breakdown
3: $reorth$, variabile booleana di controllo riortogonalizzazione
4: $\beta_1 = \|B\|_F, \quad V_1 = B/\beta_1$ // passo 0
5: $\tilde{V} = AV_1, \quad \alpha_1 = \langle V_1, \tilde{V}\rangle$ // passo 1
6: $\tilde{V} = \tilde{V} - \alpha_1 V_1$
7: $\beta_2 = \|\tilde{V}\|_F$
8: $T_{1,1} = \alpha_1, \quad T_{2.1} = \beta_2$ // sistemazione elementi matrice T
9: $y = T_1/\|B\|_F \mathbf{e}_1$ // calcolo soluzione al passo 1
10: $j = 1$
11: **repeat** // ciclo principale
12: $j = j + 1, y_0 = y$
13: $T_{j-1,j} = \beta_j$
14: $\tilde{V} = AV_j - \beta_j V_{j-1}, \quad \alpha_j = \langle V_j, \tilde{V}\rangle$
15: $\tilde{V} = \tilde{V} - \alpha_j V_j$
16: **if** reorth==true
17: **for** $i = 0, \ldots, j$
18: $\tilde{V} = \tilde{V} - \langle \tilde{V}, V_i\rangle V_i$
19: **end for**
20: **end if**
21: $\beta_{j+1} = \|\tilde{V}\|_F$
22: $T_{j+1,j} = \beta_{j+1}, \quad T_{j,j} = \alpha_j$
23: **if** $\beta_{j+1} < tau_\beta$ **then** uscita per breakdown **end**
24: $V_{j+1} = \tilde{V}/\beta_{j+1}$
25: $y = T_j/\|B\|_F \mathbf{e}_1$
26: **until** $\|y - y_0\| < tol\|y\|$ or $j < n$
27: $Y = y \otimes I_k$
28: $X = \mathcal{V}Y$ // soluzione riportata nello spazio originario
29: **Output:** soluzione sistema

Al passo 9 ed al passo 25, viene invece calcolata la soluzione ai minimi

quadrati $\mathbf{y} \in \mathbb{R}^l$, come illustrato nella sezione precedente.
È importante notare come $\mathbf{y} \in \mathbb{R}^l$, rappresenti la soluzione di un problema non vincolato, mentre X è la soluzione del problema di minimo vincolato a sinistra dell'equazione 2.5.
Nelle righe 16-20, è stato previsto l'inserimento di una procedura di riortogonalizzazione dei blocchi della matrice $\mathcal{V}$, poiché al crescere del numero delle iterazioni è possibile che i blocchi tendano a perdere la reciproca ortonormalità per il propagarsi di errori di macchina. In questo caso, abbiamo optato per una riortogonalizzazione completa piuttosto che selettiva, poiché dai test noi effettuati non abbiamo notato cali prestazionali di interesse tali da far preferire la seconda rispetto alla prima.
L'ultimo importante aspetto da evidenziare del nostro algoritmo, è il controllo nella riga 23 sull'elemento β_{j+1}, confrontato con una tolleranza inserita da input, al fine di gestire in modo corretto un'eventuale situazione di *breakdown*.
Se i blocchi $A^iB, \quad i = 0, 1, \ldots, n-1$ risultassero essere tutti indipendenti, avremmo che $\mathcal{K}_n \equiv R^{n \times k}$, e l'algoritmo terminerebbe in n passi. Se non si avesse questa indipendenza, potrebbe accadere che per un certo $l < n$, il blocco A^lB risulti essere *linearmente dipendente* dai blocchi precedenti. In questo caso, esisterebbero degli scalari $\alpha_i, \quad i = 0, \ldots, l$, non tutti nulli, tali che:

$$\sum_{i=0}^{l} \alpha_i A^i B = 0.$$

Supponendo $\alpha_0 \neq 0$, dalla precedente relazione è possibile esplicitare la matrice B ottenendo

$$AX^* = B = -\frac{1}{\alpha_0} \sum_{i=1}^{l} \alpha_i A^i B = A\Big(\frac{1}{\alpha_0} \sum_{i=1}^{l} \alpha_i A^{i-1} B\Big) \tag{2.6}$$

dove X^* indica la soluzione **esatta** del sistema lineare $AX = B$, il che implica

$$X^* = \frac{1}{\alpha_0} \sum_{i=1}^{l} \alpha_i A^i B \in \mathcal{K}_l$$

ed il metodo terminerebbe in l passi (vedi [5]).
In questa eventualità, l'algoritmo 2 dal passo 23 in poi restituirebbe la soluzione esatta del sistema, come calcolata nella riga 28, e le matrici $\mathcal{V}$ e T della fattorizzazione di Lanczos opportunamente "ritagliate".

Chapter 3

Test numerici

In questo capitolo ci siamo occupati del test dell'algoritmo 2 da noi elaborato su matrici di interesse. In particolare, abbiamo preso in esame due aspetti principali:

- accuratezza nel calcolo della soluzione;
- velocità di esecuzione.

Questo ci ha portato pertanto ad elaborare due test differenti, illustrati nelle sezioni sottostanti, entrambi realizzati in ambiente MATLAB, tramite script e funzioni opportune. Tutti i dati sono stati elaborati partendo da due matrici particolari:

- una matrice ad entrate casuali, generata con la funzione **rand** di MATLAB, resa poi successivamente simmetrica;
- una matrice di Lanczos, ottenuta tramite una parziale riortogonalizzazione di una approssimazione agli elementi finiti, di un operatore biarmonico su un dominio rettangolare, con un lato tenuto fisso e gli altri liberi.

La seconda matrice, di dimensione 960, è simmetrica e definita positiva, ed è stata scaricata dal sito [4].
Premettiamo che per portare a convergenza l'algoritmo, abbiamo dovuto modificare leggermente la struttura delle due matrici per evitare l'applicazione di un precondizionatore. Tale modifica, ha interessato gli elementi della diagonale di entrambe le matrici, che sono stati resi proporzionali di un fattore **fact**, alla somma degli elementi della riga di appartenenza.

3.1 Test 1

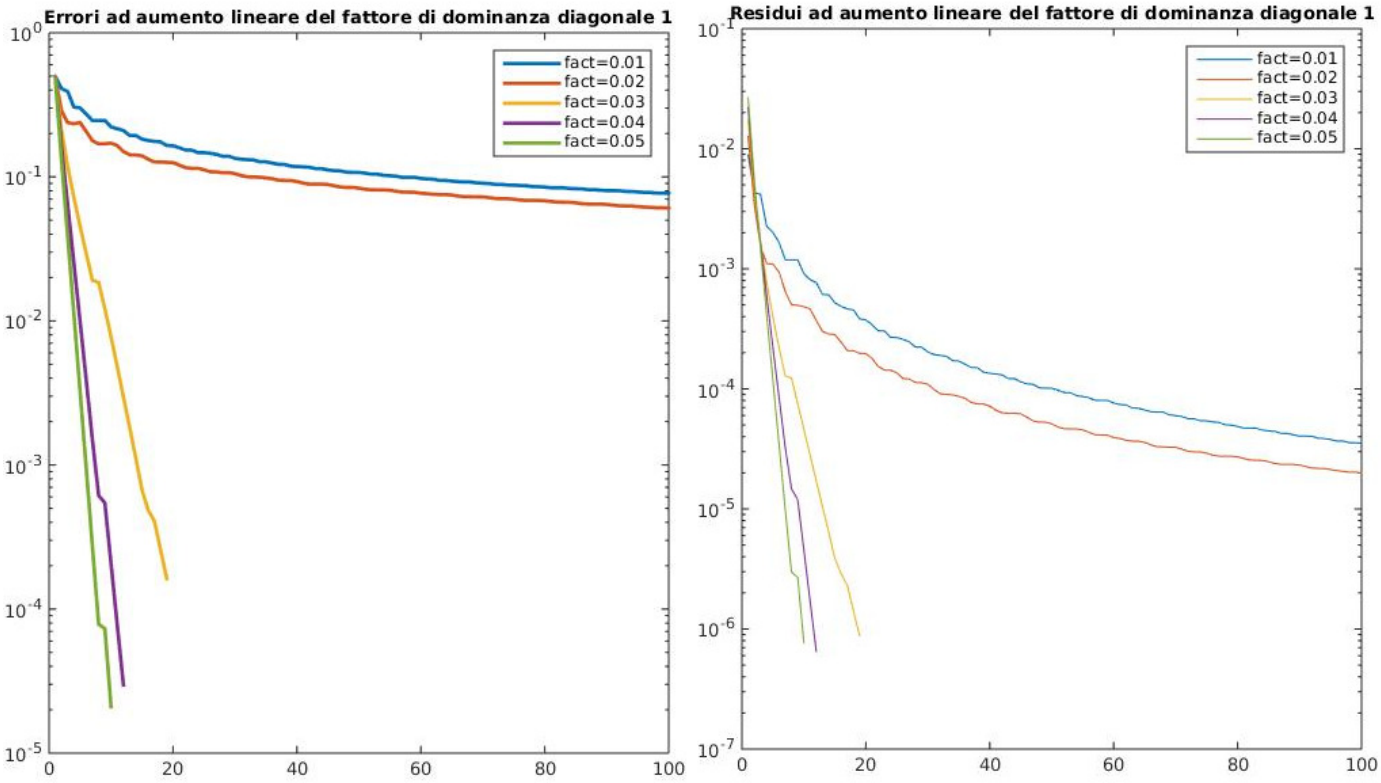

Figure 3.1: *Andamento errori e residui normalizzati matrice random*

Il primo test effettuato analizza l'andamento degli errori e dei residui normalizzati, nel caso di un incremento lineare del fattore di proporzionalità **fact**, considerando come matrice di partenza una matrice di dimensione 1000 ad entrate casuali generata tramite la funzione *rand* di MATLAB.
In entrambe le figure 3.1 e 3.2, l'andamento di errori e residui è mostrato rispetto al numero di iterazioni dell'algoritmo nella risoluzione del sistema lineare $AX = B$ (dove A è la matrice random citata precedentemente), per i valori da noi utilizzati del **fact**.
La matrice B, è stata invece calcolata a partire dalla matrice A e da una soluzione nota a priori denominata *truesol*, rispetto alla quale sono stati calcolati anche i residui. L'output dei test è stato separato in due figure per renderlo maggiormente comprensibile.

Risulta evidente come per i primi due valori del **fact**, ovvero 0.01 e 0.02, l'algoritmo non vada a convergenza.
Per un **fact** pari a 0.03 invece, l'algoritmo giunge a convergenza con 21 iterazioni. Per incrementi successivi, le prestazioni dell'algoritmo continuano a migliorare, con una precisione sempre maggiore per un numero di iterazioni sempre minore.
A questo proposito, come si può osservare nella figura 3.2, per valori di **fact** superiori a 0.05, il numero di iterazioni è minore di 10, e giunge al valore minimo di 7 negli ultimi 3 test effettuati.

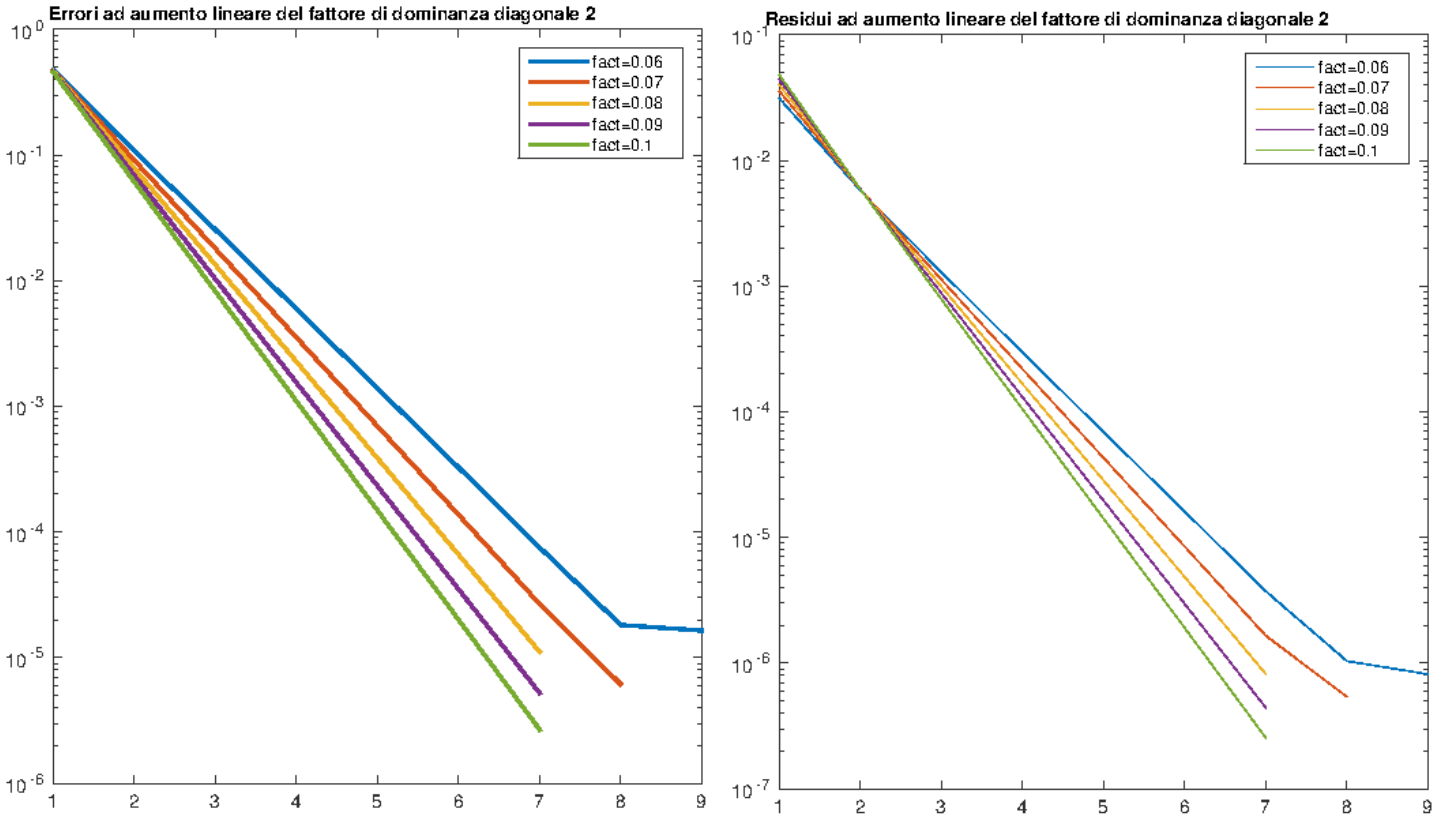

Figure 3.2: *Andamento errori e residui normalizzati matrice random*

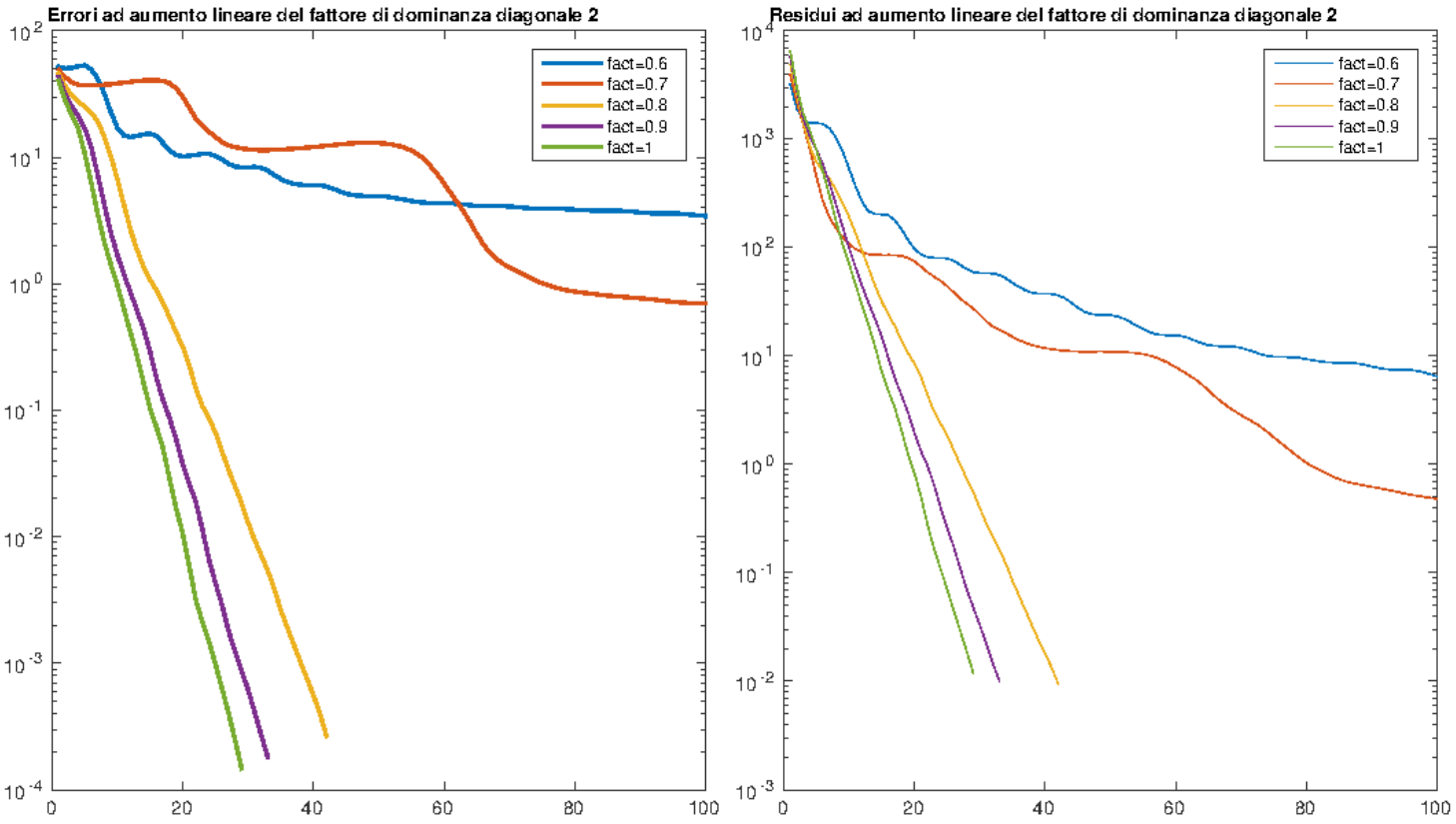

Figure 3.3: *Andamento errori e residui normalizzati matrice sperimentale*

Per i test eseguiti sulla matrice sperimentale di dimensione 960, abbiamo dovuto aumentare il range di incremento del **fact** di un fattore 10 rispetto ai test eseguiti con la matrice rand.
Come si evince difatti dalla figura 3.3, per i primi 5 test l'algoritmo non riesce a giungere a convergenza. A partire da valori del **fact** maggiori di 0.7, l'algoritmo converge giungendo al minimo numero di iterazioni pari a 29, in corrispondenza del valore del **fact** pari a 1, in una condizione per la matrice di dominanza diagonale debole per righe (figura 3.4).

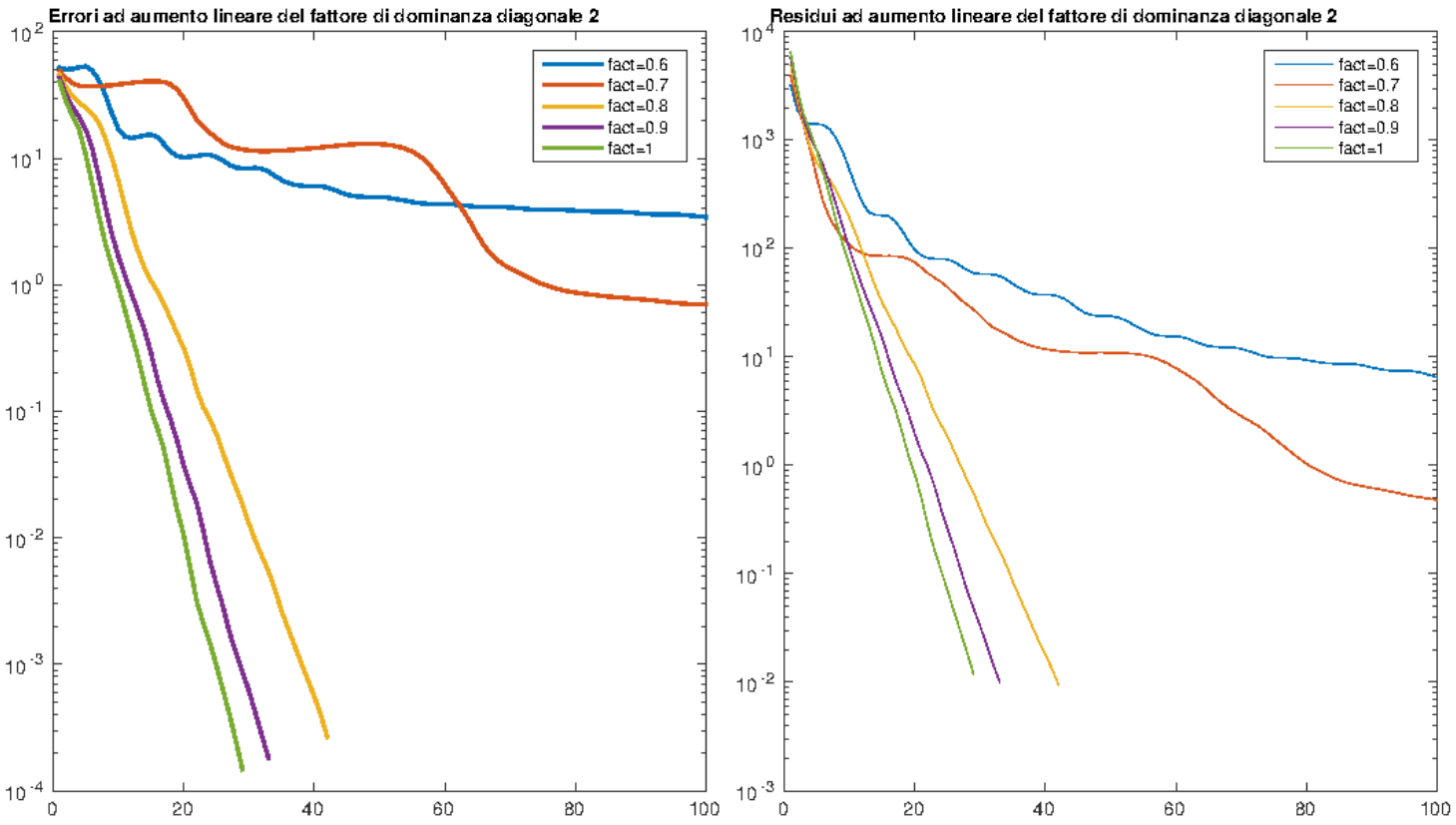

Figure 3.4: *Andamento errori e residui normalizzati matrice sperimentale*

3.2 Test 2

In questa sezione prenderemo in esame i test da noi eseguiti sull'efficacia in termini di prestazioni della riortogonalizzazione dei blocchi generati nella fattorizzazione globale di Lanczos, per la risoluzione di sistemi lineari del tipo $AX = B$ visti in precedenza.
Suddetti test sono stati eseguiti su delle matrici ad entrate casuali di dimensione crescente (da 1000 a 10000), alle quali è stato applicato un **fact** di proporzionalità pari 0.03, per portare a convergenza l'algoritmo.
Il test si è suddiviso in:

1. un test in cui la dimensione dei blocchi viene lasciata fissa, e pertanto, al crescere della dimensione della dimensione della matrice, aumenta linearmente il numero di blocchi;

2. un test in cui il rapporto tra le dimensioni della matrice e dei blocchi, viene mantenuto fisso, e pertanto il numero dei blocchi della fattorizzazione rimane costante, aumentando tuttavia la dimensione di quest'ultimi.

Scopo dei test è verificare se, a parità di precisione della soluzione (numero di iterazioni eseguite dall'algoritmo), la riortogonalizzazione dei blocchi, che richiede uno sforzo computazionale maggiore, comporti dei tempi di esecuzione sensibilmente più lunghi rispetto alla versione dell'algoritmo privo di essa.

Nella figura 3.5, in cui viene illustrato il test 1, notiamo 3 situazioni principali.
A parità di numero di iterazioni, l'algoritmo dotato di riortogonalizzazione, si dimostra sempre leggermente più lento. Esistono comunque dei casi in cui suddetto algoritmo si dimostra leggermente più veloce: in tali situazioni, la motivazione da noi rintracciata consiste nel fatto che l'algoritmo, essendo più preciso, esegue meno iterazioni.
Notiamo, inoltre, che la crescita del numero dei blocchi (al crescere della dimensione della matrice) influisce sui tempi di esecuzione generale dell'algoritmo, com'era lecito aspettarsi, ma non sui tempi necessari alla riortogonalizzazione, visto che la differenza a parità di iterazioni si mantiene più o meno costante in tutti i casi.
Gli unici due casi che si discostano da quelli esaminati precedentemente sono esaminabili nella figura 3.6, dove, nell'esecuzione dell'algoritmo con matrici di dimensione 9000 e 10000, a parità di numero di iterazioni, la riortogonalizzazione si dimostra rispettivamente sensibilmente più lenta, e più veloce.

Imputiamo questa situazione alla natura casuale delle entrate delle due matrici.

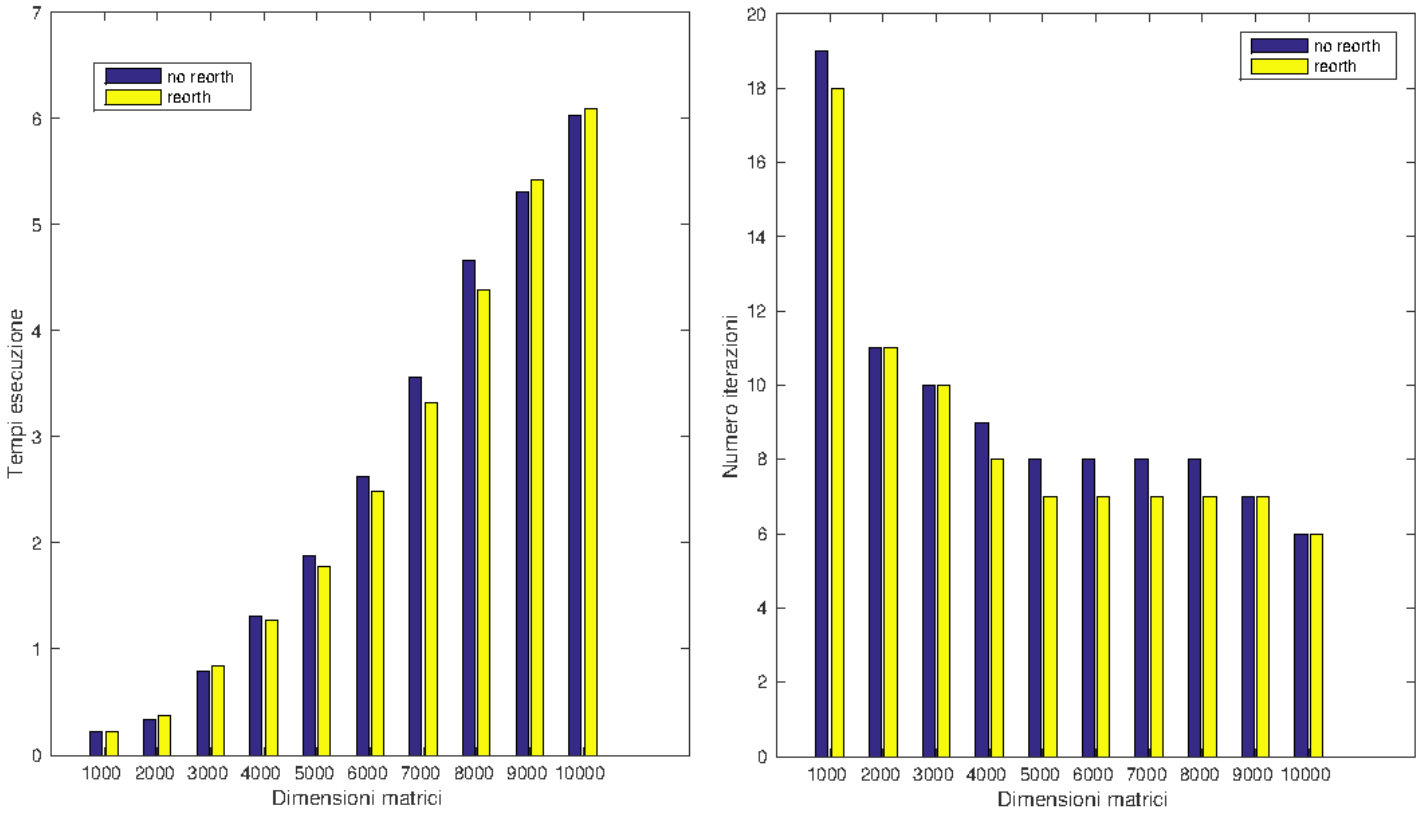

Figure 3.5: *Tempi di esecuzione e numero di iterazioni algoritmo al crescere della dimensione della matrice A, dimensione blocchi fissa.*

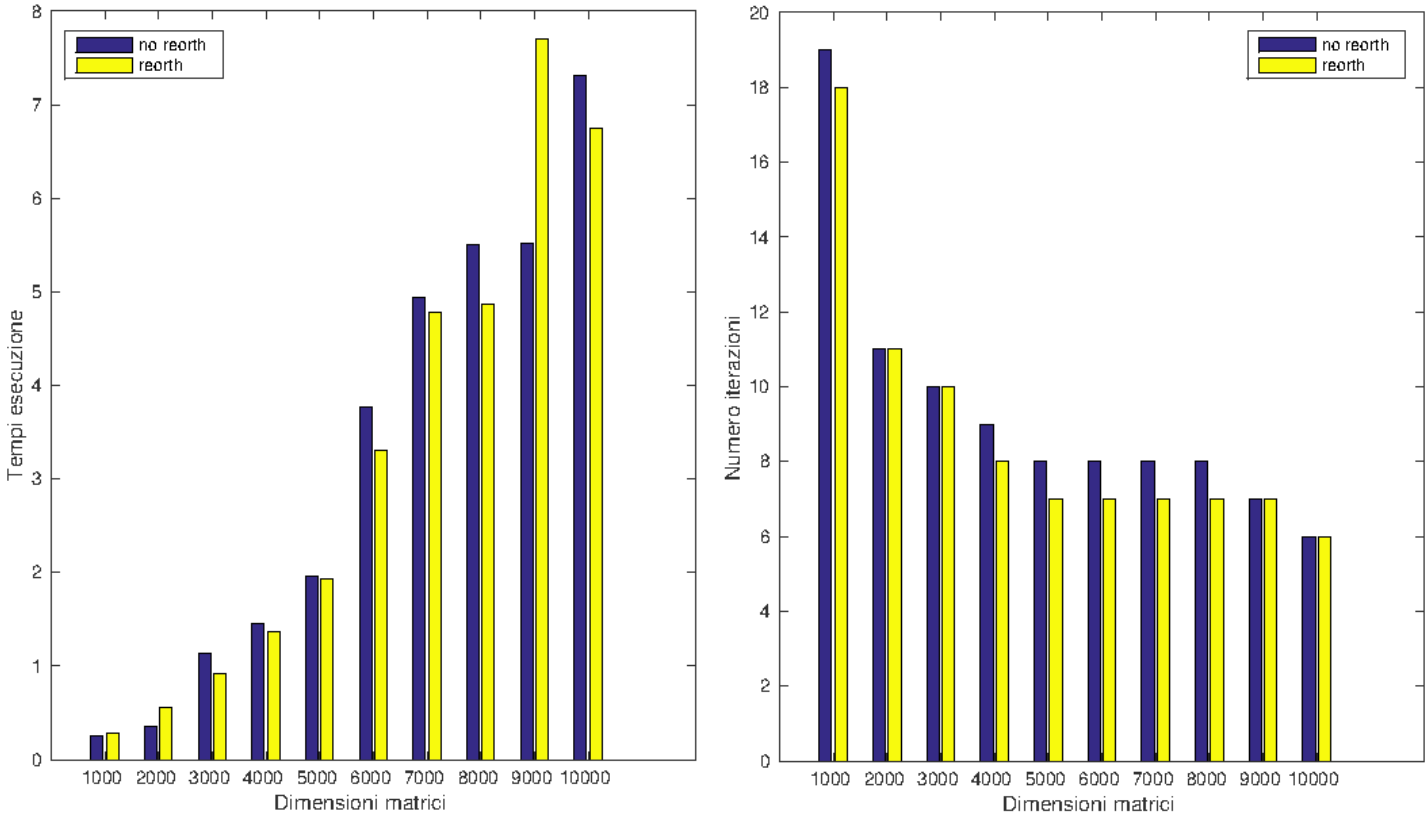

Figure 3.6: *Tempi di esecuzione e numero di iterazioni algoritmo al crescere della dimensione della matrice A, dimensioni blocchi fissa*

I risultati del test numero 2, sono rappresentati nelle figure 3.7 e 3.8.
Nella prima figura, notiamo che i tempi di esecuzione rispettano più o meno la casistica osservata nel test precedente. Difatti, a parità di numero di iterazioni, la riortogonalizzazione si dimostra più lenta, con un ritardo maggiore al crescere della dimensione della matrice (e quindi della dimensione dei singoli blocchi). Si osservino a tal proposito, i casi per dimensione della matrice pari a 9000 e 10000.
La situazione in cui l'algoritmo con riortogonalizzazione si presenta più rapido, è nuovamente imputabile al fatto che esso, essendo più preciso, esegue meno iterazioni. Tale casistica è evidente nella figura 3.8, dove, per dimensione della matrice pari a 10000, l'algoritmo privo di riortogonalizzazione, si dimostra più lento, ma eseguendo un'iterazione in più.
Come già accennato in precedenza, rispetto al primo test, i ritardi dell'algoritmo con riortogonalizzazione rispetto a quello privo di essa, crescono con la dimensione della matrice in esame. Questo aspetto è imputabile al fatto che anche la dimensione dei blocchi cresce linearmente con essa, mantenendo un rapporto costante. Pertanto, combinando i risultati dei due test, si può concludere che l'aspetto più importante nel deterioramente delle prestazioni dell'algortmo con l'uso della tecnica di riortogonalizzazione, sia non tanto il numero di blocchi realizzato nella creazione del sottospazio di Krylov, quanto la dimensione degli stessi.

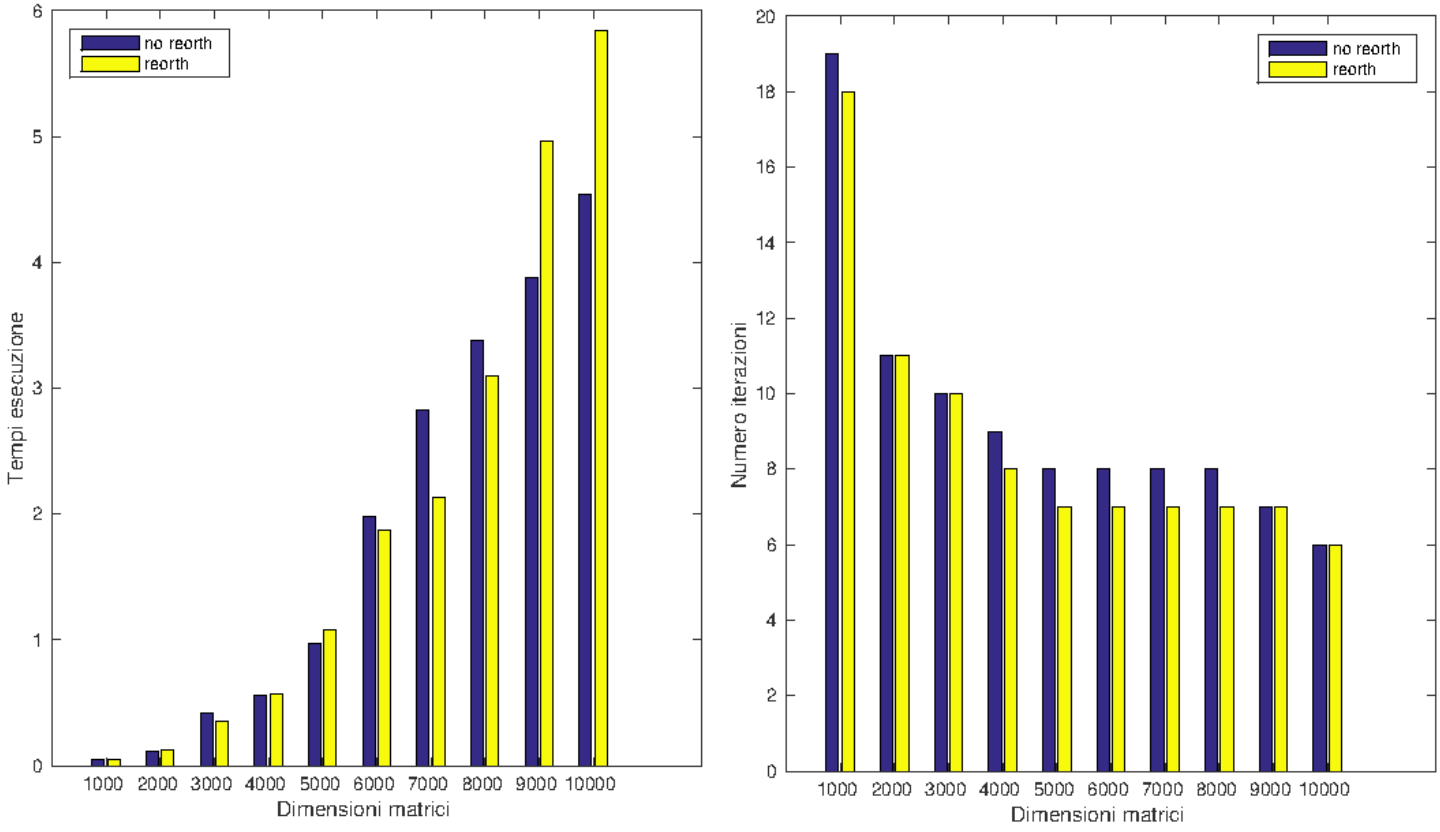

Figure 3.7: *Tempi di esecuzione e numero di iterazioni algoritmo al crescere della dimensione della matrice A, numero blocchi fisso*

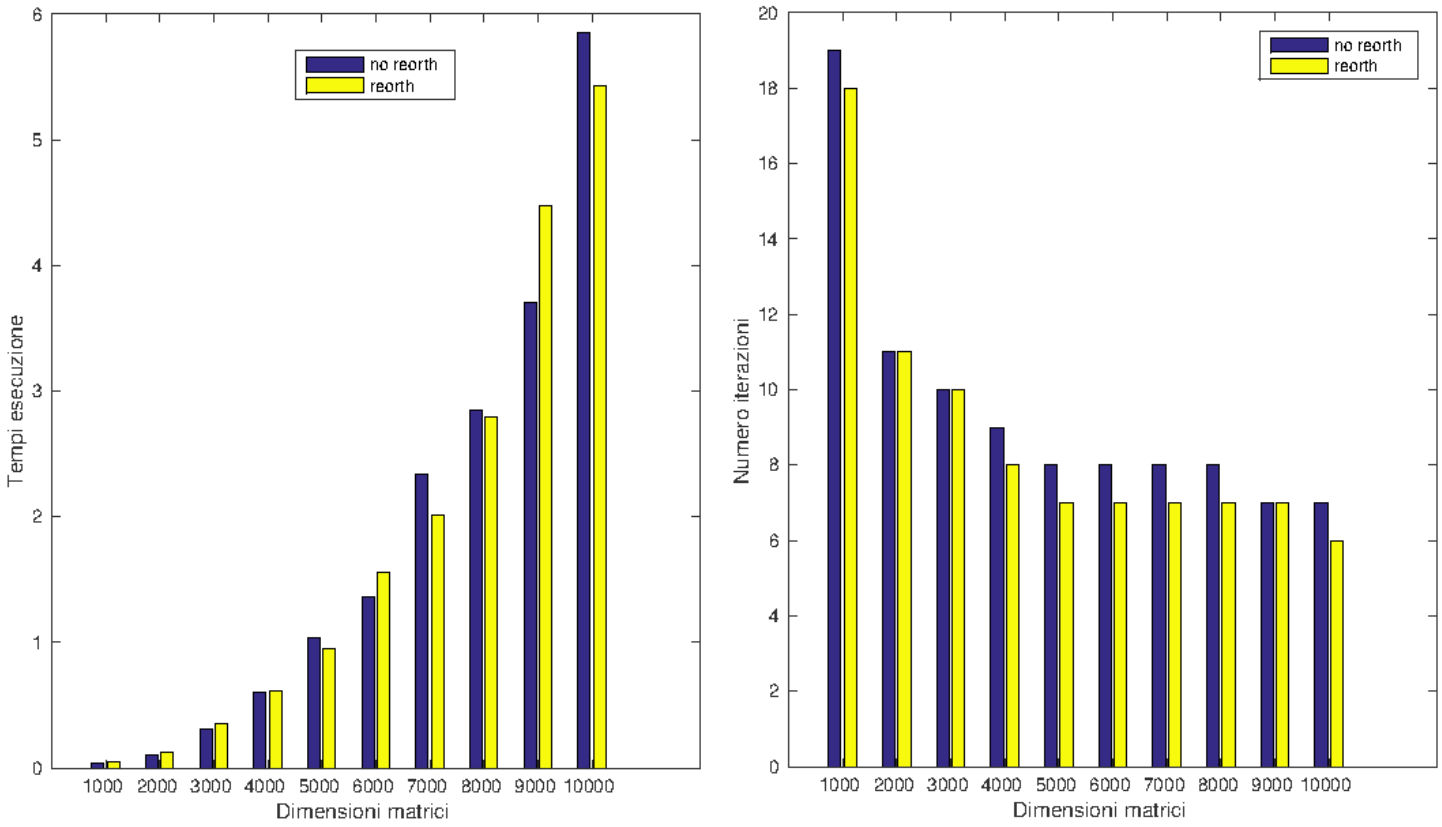

Figure 3.8: *Tempi di esecuzione e numero di iterazioni algoritmo al crescere della dimensione della matrice A, numero blocchi fisso*

Bibliography

[1] M. Bellalij, L. Reichel, G. Rodriguez, and H. Sadok. Bounding matrix functionals via partial global block Lanczos decomposition. *Applied Numerical Mathematics*, 94:127 – 139, 2015.

[2] A. El Guennouni, K. Jbilou, and H. Sadok. The block Lanczos method for linear systems with multiple right-hand sides. *Applied Numerical Mathematics*, 51(2–3):243–256, 2004.

[3] L. Elbouyahyaoui, A. Messaoudi, and H. Sadok. Algebraic properties of the block GMRES and block Arnoldi methods. *Electronic Transation on Numerical Analysis*, 33:207–220, 2009.

[4] US government. Matrix market, http://math.nist.gov/matrixmarket/.

[5] G. Rodriguez. *Algoritmi Numerici*. Pitagora Editrice Bologna, Bologna, 2008.