

# Metodo del gradiente coniugato con precondizionamento applicato al problema del GeneRank

---

Francesco Caporali

18 maggio 2021

## 1 Introduzione

Questa relazione ha lo scopo di analizzare diversi possibili metodi risolutivi al problema della ricerca del vettore *soluzione* del modello di GeneRank.

Tale modello nasce nel 2005 come possibile schema di *ordinamento* di liste di geni sulla base della loro *connessione* e del valore ad essi attribuito in seguito a determinate sperimentazioni.

L'idea è quella di sfruttare un sistema preesistente, il PageRank, utilizzato da Google per creare un ordine di priorità all'interno delle pagine web, per poter automatizzare, almeno parzialmente, l'ordinamento di liste genetiche, utilizzando tutte le informazioni disponibili. Operativamente questo metodo prevede la costruzione della matrice di adiacenza del grafo del web e la ricerca di un vettore risultato che attribuisca un valore maggiore alle pagine che si considerano *più importanti*, in un senso che sarà chiarito nel seguito.

In un primo momento sono stati implementati algoritmi che permettono di risalire al vettore di *GeneRank* applicando le stesse tecniche utilizzate da Google (Julie L. Morrison [2]). Successivamente, partendo dall'idea che al crescere della taglia delle liste genetiche si dovesse cercare di rendere computazionalmente accettabile il tempo impiegato per la ricerca del risultato, si è pensato di poter sfruttare la particolare struttura dei dati per migliorare le prestazioni. Infatti i grafi geneticici in questione hanno la notevole proprietà di essere non orientati, il che lascia in eredità alle loro matrici di adiacenza una struttura simmetrica, fatto che le rende molto più trattabili rispetto a quelle analizzate nel modello di PageRank. Sulla base di questa idea è stata sviluppata la ricerca di Gang Wu, Wei Xu, Ying Zhang e Yimin Wei nell'articolo *A preconditioned conjugate gradient algorithm for GeneRank with application to microarray data mining* [1] e nelle seguenti pagine si cerca di ripercorrere i risultati da loro ottenuti al fine di eseguire sperimentazioni numeriche che ne confermino la correttezza.

## 2 Problema

L'individuazione di una priorità nelle liste genetiche sulla base di dati ed informazioni pregresse è una questione molto studiata negli ultimi anni.

Nell'articolo di riferimento (Gang Wu [1]) viene presa in analisi una specifica metodologia di priorizzazione basata sulle connessioni dei geni che mira a produrre una *classifica* genetica, da usare per gli esperimenti con microarray di DNA, che sia stabile rispetto ad errori nella raccolta dei dati. Con questo modello viene generato un sistema di equazioni lineari sparso di grandi dimensioni la cui risoluzione può essere approssimata in diversi modi.

La costruzione del grafo genetico è piuttosto naturale e viene fatta sulla base di database di informazioni (si veda *GO database*); in particolare in relazione alle *annotazioni* condivise dai diversi geni in esame all'interno dei suddetti DB vengono stabilite delle connessioni bilaterali.

Così facendo si possono ottenere matrici  $W = (w_{ij})_{i,j} \in (0, 1)^{n \times n}$  così definite<sup>1</sup>

$$w_{ij} = \begin{cases} 1 & \text{se } g_i \text{ e } g_j \text{ condividono una annotazione in GO } (g_i \neq g_j) \\ 0 & \text{altrimenti} \end{cases} \quad (1)$$

Come già detto  $W$  è simmetrica in quanto il grafo da cui discende è non orientato.

In maniera del tutto analoga a quanto previsto dal metodo *PageRank*, a partire da tale matrice se ne costruisce un'altra che ci permetta di classificare i nodi in relazione alla quantità di connessioni formate tra di essi. A tale fine vengono dapprima isolati i cosiddetti *dangling nodes* (per maggiori dettagli si veda il codice 25), nel caso in questione si tratta di quei geni che non presentano annotazioni in comune con nessun altro gene<sup>2</sup>, ed eliminati, rimpiazzando gli 0 nella riga e colonna corrispondenti con degli 1<sup>3</sup>. Quindi viene costruita  $D^{-1}$  a partire da  $D = (d_{ij})_{i,j}$  diagonale, ottenuta come segue

$$d_{ij} = \begin{cases} \sum_{k=1}^n w_{ik} = \sum_{k=1}^n w_{ki} & \text{se } i = j \\ 0 & \text{altrimenti} \end{cases} \quad (2)$$

La ricerca del vettore di *GeneRank* è, a questo punto, eseguibile per mezzo di due metodologie distinte, ma del tutto analoghe. Si può affrontare un problema di ricerca di un autovettore sinistro di una matrice di termini positivi oppure si può passare per la risoluzione di un sistema lineare. In entrambi i casi la soluzione non può essere trovata con approcci diretti in quanto la taglia di  $W$  rende il problema intrattabile, bensì vanno utilizzati dei metodi iterativi.

Si utilizzano, oltre a  $W$  e  $D^{-1}$ , sopra introdotte, delle variabili di supporto con le quali ci si può assicurare di trovare in ogni caso una soluzione. Si introducono un *damping factor*  $\alpha \in (0, 1)$  ed un *vettore di personalizzazione*  $\mathbf{ex}$ , strettamente positivo e normalizzato in

---

<sup>1</sup>dove  $n$  è il numero di geni in considerazione

<sup>2</sup>per convenzione si è imposto che la diagonale di  $W$  sia nulla

<sup>3</sup>nei codici nella sezione 7 questo non avviene in quanto è stato fatto nello script principale che richiama le funzioni

norma 1<sup>4</sup>.

Il problema di ricerca di autovettori che ci permette di stabilire la classifica tra i geni è il seguente

$$x^T(\alpha D^{-1}W + (1 - \alpha)e \mathbf{ex}^T) = x^T \quad (3)$$

con  $\mathbf{ex}^T e = 1$  ed  $e$  il vettore con 1 in tutte le componenti, di taglia  $n \times 1$ ,  $e = (1 \dots 1)^T$ . Si può affermare che esiste sempre un tale  $x$  ed è unico a meno di normalizzazione grazie al Teorema 2.1.

**Teorema di Perron-Frobenius 2.1.** *Sia  $A = (a_{ij})_{ij} \in \mathcal{M}(\mathbb{R}, n)$  valgono le seguenti affermazioni*

- se  $A \geq 0$  (ovvero se  $\forall i, j = 1 \dots n$   $a_{ij} \geq 0$ ) allora  $\exists \lambda \in Sp(A)$  tale che  $\lambda = \rho(A)$  ed in corrispondenza un autovettore destro e uno sinistro, rispettivamente  $x$  e  $y^T$  tali che  $\forall i = 1 \dots n$   $x_i \geq 0$  e  $y_i \geq 0$
- se  $A \geq 0$  ed irriducibile allora  $\rho(A)$  è semplice e i rispettivi autovettori destro e sinistro sono strettamente positivi
- se  $A > 0$ <sup>5</sup> allora  $\rho(A)$  è l'unico autovalore di modulo massimo

Si noti, infatti, che la matrice  $A = \alpha D^{-1}W + (1 - \alpha)e \mathbf{ex}^T$  è tale che  $\forall i, j$   $A_{ij} > 0$  e  $\rho(A) = 1$ , inoltre  $Ae = e$  dunque l'unico autovalore  $\lambda$  tale che  $|\lambda| = \rho(A)$  è proprio  $\lambda = 1$  ed è semplice.

La soluzione  $x$  di tale problema è un vettore di *pesi* attribuibili a ciascun gene che permettono di denotarne l'importanza.

Il sistema lineare che si può prendere in analisi alternativamente

$$(I - \alpha W^T D^{-1})x = (1 - \alpha) \mathbf{ex}, \quad \|\mathbf{ex}\|_1 = 1 \quad (4)$$

è del tutto analogo. Si noti infatti che denotando  $A = \alpha D^{-1}W + (1 - \alpha)e \mathbf{ex}^T$  il problema di ricerca dell'autovettore (3) sopra esposto è riscrivibile come  $A^T x = x$

$$\begin{aligned} &\text{se si scrive } A^T = \alpha W^T D^{-1} + (1 - \alpha) \mathbf{ex} e^T \text{ allora} \\ &A^T x = x \iff (I - \alpha W^T D^{-1})x = (1 - \alpha) \mathbf{ex} (e^T x) \\ &\iff (I - \alpha W^T D^{-1})x = (1 - \alpha) \mathbf{ex} \end{aligned}$$

dove per l'ultimo *se e solo se* si è usato che si può cercare  $x$  normalizzato in norma 1 in quanto è unico a meno di scalare.

Per mezzo di questo secondo metodo risolutivo e utilizzando la fondamentale proprietà  $W = W^T$  è possibile riscrivere il sistema 4 con una matrice simmetrica definita positiva (*SPD*) e su tale base si può provare ad approssimare una soluzione utilizzando il noto algoritmo del gradiente coniugato (*CG*).

---

<sup>4</sup>quest'ultimo nel caso di *GeneRank* è tale che  $\mathbf{ex}_i$  corrisponde al valore assoluto del cambiamento dell'espressione del gene  $g_i$

<sup>5</sup>chiaramente  $A > 0 \implies A \geq 0$  ed irriducibile

### 3 Metodo del gradiente coniugato

Partendo dal sistema  $(I - \alpha W^T D^{-1})x = (1 - \alpha) \mathbf{e}$  l'idea è piuttosto semplice: sia  $x$  soluzione del sistema

$$(I - \alpha W^T D^{-1})x = (1 - \alpha) \mathbf{e} \iff (D - \alpha W^T)\bar{x} = (1 - \alpha) \mathbf{e} \wedge \bar{x} = D^{-1}x \quad (5)$$

Dunque se si riesce a risolvere  $(D - \alpha W)\bar{x} = (1 - \alpha) \mathbf{e}$  per (5) basta prendere la soluzione  $\bar{x}$  e moltiplicarla per  $D$ , operazione con costo computazionale basso<sup>6</sup>.

Il seguente Teorema 3.1 permette di affermare che quanto appena detto è degno di nota in quanto il sistema ottenuto gode di un'importante proprietà.

**Teorema 3.1.**  *$D - \alpha W$  è una matrice simmetrica definita positiva.*

*Dimostrazione.* Essendo  $D$  e  $W$  simmetriche anche  $D - \alpha W$  lo sarà; rimane da verificare che sia definita positiva ovvero che  $Sp(D - \alpha W) \subset \mathbb{R}^+$ . Si ricorda che la segnatura di una matrice è invariante per congruenza e quindi in particolare lo è la definitezza. Nella ricerca di una matrice congruente a  $D - \alpha W$  si nota che

$$D^{-\frac{1}{2}}(D - \alpha W)D^{-\frac{1}{2}} = I - \alpha(D^{-\frac{1}{2}}WD^{-\frac{1}{2}})$$

e

$$D^{-\frac{1}{2}}WD^{-\frac{1}{2}} = D^{-\frac{1}{2}}(WD^{-1})D^{\frac{1}{2}}$$

da cui si ottiene che  $D^{-\frac{1}{2}}(D - \alpha W)D^{-\frac{1}{2}} = I - \alpha(D^{-\frac{1}{2}}WD^{-\frac{1}{2}})$  è simile a  $I - \alpha(WD^{-1})$  e che in particolare hanno lo stesso spettro. Con le seguenti osservazioni si conclude:

$$\begin{aligned} Sp(I - \alpha(WD^{-1})) &= 1 - \alpha Sp(WD^{-1}) \text{ e} \\ \rho(WD^{-1}) &\leq \|WD^{-1}\|_1 = 1 \end{aligned}$$

Infatti dato  $\lambda \in Sp(WD^{-1})$  si ha  $1 - \alpha\lambda \geq 1 - \alpha > 0 \implies Sp(I - \alpha(WD^{-1})) \subset \mathbb{R}^+$ .

La prima osservazione è immediata, mentre per la seconda basta notare che

$$\|WD^{-1}\|_1 = \|D^{-1}We\|_\infty = \|e\|_\infty = 1$$

dove  $\|WD^{-1}\|_1 = \|D^{-1}We\|_\infty$  è giustificata dal fatto che  $WD^{-1} \geq 0$  e dunque, denotando  $WD^{-1} = A$  e  $(WD^{-1})_{ij} = a_{ij}$ , si ha

$$\|WD^{-1}\|_1 := \max_{j=1 \dots N} \sum_{i=1}^N a_{ij} = \max_{j=1 \dots N} e^T A^j = \max_{j=1 \dots N} (e^T A)^j = \max_{j=1 \dots N} (A^T e)_j = \|D^{-1}We\|_\infty$$

□

---

<sup>6</sup>il costo è esattamente  $n$  essendo  $D$  diagonale

### 3.1 Gradiente Coniugato ( $CG$ )

Dato che  $D - \alpha W$  è una  $SPD$  allora si può risolvere il sistema 5 con il metodo del gradiente coniugato (Codice 12), la cui convergenza è assicurata dal seguente Teorema 3.2

**Teorema 3.2.** *Data  $A(:= D - \alpha W)$  matrice simmetrica definita positiva ed il sistema  $Ax = b$ , denotando con  $\bar{x}$  la soluzione esatta e con  $x_k$  l'approssimazione data da  $CG$  al passo  $k$ , vale*

$$\|\bar{x} - x_k\|_A \leq 2 \left( \frac{\sqrt{\mu_2(A)} - 1}{\sqrt{\mu_2(A)} + 1} \right)^k \|\bar{x} - x_0\|_A \quad (6)$$

dove  $\|v\|_A = \sqrt{v^T A v}$  e  $\mu_2(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$  in quanto  $A$  è simmetrica.

In realtà questo teorema mostra non solo la convergenza, ma fornisce una stima della sua velocità. Segue immediatamente che restringendo lo spettro si ha una forte accelerazione della convergenza che aumenta con velocità esponenziale nel numero  $k$  di passi da eseguire prima di raggiungere la soglia prestabilita. Un modo per migliorare il  $CG$  è dunque cercare di avvicinare  $\lambda_{max}(D - \alpha W)$  e  $\lambda_{min}(D - \alpha W)$ . Questo obiettivo viene raggiunto per mezzo di un precondizionamento.

### 3.2 Gradiente Coniugato con Precondizionamento ( $PCG$ )

Un precondizionamento tipicamente usato per il metodo del gradiente coniugato è della forma  $M = CC^T$ ; trasforma il sistema  $Ax = b$  in un nuovo sistema equivalente  $(C^{-1}AC^{-T})(C^T x) = C^{-1}b$ .

Questa struttura è fondamentale poichè un sistema  $SPD$  con buone proprietà, deve essere trasformato in un nuovo sistema che le preservi: usando un precondizionatore  $M = C_1 C_2$ <sup>7</sup> si dovrebbe garantire che  $C_1 A C_2$  sia ancora  $SPD$ , che  $\mu_2(C_1 A C_2) < \mu_2(A)$  e che sia facile risalire alla vera soluzione conoscendo  $C_2 x$ .

In questo particolare caso viene scelta  $C = C^T = D^{\frac{1}{2}}$ <sup>8</sup>. In tal modo data  $\bar{x}$  soluzione del sistema, vale

$$(D - \alpha W^T)\bar{x} = (1 - \alpha) \mathbf{ex} \iff \begin{aligned} (I - \alpha D^{-\frac{1}{2}} W D^{-\frac{1}{2}})\tilde{x} &= (1 - \alpha)(D^{-\frac{1}{2}} \mathbf{ex}) \\ \wedge \tilde{x} &= D^{\frac{1}{2}}\bar{x} \end{aligned} \quad (7)$$

Dunque invece di utilizzare l' $SPD$   $D - \alpha W^T$  si sfrutta la nuova matrice  $I - \alpha D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ , il cui spettro è notevolmente appiattito, come dimostrato nel Teorema 3.3.

---

<sup>7</sup>in generale precondizionare con  $C_1 C_2$  significa passare da  $Ax = b$  al nuovo sistema  $(C_1^{-1} A C_2^{-1})(C_2 x) = C_1^{-1}b$

<sup>8</sup>è semplice verificare che tale precondizionatore possiede la prima e la terza proprietà tra quelle sopra elencate. Per la seconda proprietà è necessario il Teorema 3.3

**Teorema 3.3.** *Gli autovalori della matrice  $I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}$  soddisfano le seguenti disuguaglianze:*

$$\lambda_{\max}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) \leq 1 + \alpha$$

$$\lambda_{\min}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) \geq 1 - \alpha$$

*Dimostrazione.* Si osservi anzitutto che  $\text{Sp}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 1 - \alpha \text{Sp}(D^{-\frac{1}{2}} WD^{-\frac{1}{2}})$  e che  $D^{-\frac{1}{2}} WD^{-\frac{1}{2}}$  è simile a  $WD^{-1}$ , dunque gli autovalori di  $D^{-\frac{1}{2}} WD^{-\frac{1}{2}}$  sono gli stessi di  $WD^{-1}$ .

Ricordando che  $\rho(WD^{-1}) \leq 1$ <sup>9</sup> e che  $\text{Sp}(WD^{-1}) = \text{Sp}(D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) \subset \mathbb{R}^+$  (poichè  $D^{-\frac{1}{2}} WD^{-\frac{1}{2}}$  è simmetrica), valgono le seguenti disuguaglianze

$$\lambda_{\max}(WD^{-1}) \leq \rho(WD^{-1}) \leq 1 \text{ e } \lambda_{\min}(WD^{-1}) \geq -\rho(WD^{-1}) \geq -1$$

e ciò implica la tesi per l'osservazione iniziale

$$\lambda_{\max}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 1 - \alpha \lambda_{\min}(WD^{-1}) \leq 1 + \alpha$$

$$\lambda_{\min}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 1 - \alpha \lambda_{\max}(WD^{-1}) \geq 1 - \alpha$$

□

Questo teorema mostra che la matrice precondizionata ha un numero di condizionamento piccolo e particolarmente legato al *damping factor*. Scegliendo  $\alpha$  opportunamente si può controllare  $\mu(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}})$ ; per esempio per  $\alpha = 0.85$ <sup>10</sup>, si ha  $\mu(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) \leq \frac{1+\alpha}{1-\alpha} = 12.3$ .

Il processo per applicare il *PCG*, sopra esposto, è solitamente applicato in maniera implicita poichè se non lo si facesse ad ogni iterazione dell'algoritmo si avrebbero 2 moltiplicazioni per  $D^{-\frac{1}{2}}$  (James W. Demmel [3, p. 317-318]). Il metodo che ne risulta è quello presente nel Codice 14; in maniera del tutto analoga a come è stato fatto per il *CG* si riporta una dimostrazione della convergenza dell'algoritmo in questione.

**Teorema 3.4.** *Date  $D - \alpha W$  matrice simmetrica definita positiva e  $B := I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}$  matrice precondizionata, i sistemi  $(D - \alpha W)x = (1 - \alpha)v$  e  $(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}})x = (1 - \alpha)(D^{-\frac{1}{2}} v)$ , le rispettive soluzioni esatte  $\bar{x}$  e  $\tilde{x} = D^{-\frac{1}{2}} \bar{x}$  e  $x_k$  l'approssimazione di  $\tilde{x}$  data da *PCG* al passo  $k$ , vale*

$$\|\tilde{x} - x_k\|_B \leq 2 \left( \frac{\sqrt{1+\alpha} - \sqrt{1-\alpha}}{\sqrt{1+\alpha} + \sqrt{1-\alpha}} \right)^k \|\tilde{x} - x_0\|_B \quad (8)$$

dove  $\|v\|_B = \sqrt{v^T B v}$ .

---

<sup>9</sup>già dimostrato alla fine del Teorema 3.1

<sup>10</sup>è il valore utilizzato da Google

*Dimostrazione.* Si può considerare questo teorema come corollario del Teorema 3.2 in quanto il *PCG* non è altro che il *CG* applicato alla matrice  $B$ . Dunque, per quanto già visto (6), si ha che

$$\|\tilde{x} - x_k\|_B \leq 2 \left( \frac{\sqrt{\mu_2(B)} - 1}{\sqrt{\mu_2(B)} + 1} \right)^k \|\tilde{x} - x_0\|_B$$

dove  $\mu_2(B) = \frac{\lambda_{max}(B)}{\lambda_{min}(B)}$  essendo  $B$  simmetrica. Per il Teorema 3.3 si ha

$$\mu_2(B) = \frac{\lambda_{max}(B)}{\lambda_{min}(B)} \leq \frac{1+\alpha}{1-\alpha}$$

Essendo poi  $\left(\frac{\sqrt{x}-1}{\sqrt{x}+1}\right)^k$  con  $k \in \mathbb{N}$  monotona crescente si ha la tesi

$$\|\tilde{x} - x_k\|_B \leq 2 \left( \frac{\sqrt{\mu_2(B)} - 1}{\sqrt{\mu_2(B)} + 1} \right)^k \|\tilde{x} - x_0\|_B \leq 2 \left( \frac{\sqrt{1+\alpha} - \sqrt{1-\alpha}}{\sqrt{1+\alpha} + \sqrt{1-\alpha}} \right)^k \|\tilde{x} - x_0\|_B$$

□

## 4 Altri metodi

Un importante obiettivo del lavoro è mettere a confronto il *CG* con altri noti algoritmi solitamente utilizzati nella ricerca del vettore di *Pagerank*. Si utilizzeranno per tale confronto i seguenti metodi iterativi<sup>11</sup>: *Jacobi*, *Gauss-Seidel*, *Power method*, *Richardson* e *Arnoldi*.

### 4.1 Jacobi

In particolare *Jacobi* è utilizzato nel seguito per confrontare la convergenza degli angoli formati tra vettori approssimanti e quelli esatti, facendo da metro di paragone per attestare l'accuratezza di *CG* e *PCG*. Di seguito una breve descrizione del suo algoritmo riportato completamente nel Codice 16.

Il sistema originale, prima di essere portato in forma di *SPD*, si presentava come descritto nell'equazione 4. Approssimare la sua soluzione con il metodo di *Jacobi* consiste nell'eseguire i seguenti passi:

- data  $A = I - \alpha WD^{-1}$ , siano  $M = \text{diag}(A)$ <sup>12</sup> ed  $N = -(A - M)$  in modo che  $A = M - N$
- partendo dal risultato al passo  $k - 1$  si consideri l'approssimazione al passo  $k$ :  $x_k = M^{-1}(Nx_{k-1} + (1 - \alpha) \mathbf{ex})$

Nel nostro caso per convenzione si è imposta  $W$  con la diagonale nulla, dunque tale proprietà è ereditata anche da  $\alpha WD^{-1}$ , segue  $M = I$  ed il metodo si riduce a

$$x_k = \alpha WD^{-1}x_{k-1} + (1 - \alpha) \mathbf{ex}$$

Come accennato nella sezione 2 si ricorda che gli approcci risolutivi al problema di *GeneRank* sono due, tra loro equivalenti: si può procedere con la ricerca di un autovettore sinistro come in (3) oppure si può approssimare la soluzione di un sistema lineare come in (4). Il metodo di *Jacobi* così come *Gauss-Seidel*, *Richardson*, *CG*, *PCG* e *Arnoldi* sono del secondo tipo, invece il *Power method* è del primo tipo.

### 4.2 Power method

Partendo da  $x^T(\alpha D^{-1}W + (1 - \alpha)e \mathbf{ex}^T) = x^T$ , l'idea alla base di questa strategia è eseguire un buon numero di volte la seguente procedura: conoscendo  $x_{k-1}^T$  si calcola  $x_k^T = x_{k-1}^T A$  con  $A = \alpha D^{-1}W + (1 - \alpha)e \mathbf{ex}^T$  e poi lo si normalizza nella norma opportuna  $x_k = \frac{x_k}{\|x_k\|}$ <sup>13</sup>. Operativamente, come riportato nel Codice 20, non è conveniente eseguire un

---

<sup>11</sup>tutti gli algoritmi sono presenti nella sezione 7

<sup>12</sup>per  $\text{diag}(A)$  si intende la matrice che ha per diagonale la diagonale di  $A$

<sup>13</sup>in realtà quest'ultimo passaggio nel nostro caso si può omettere, scegliendo infatti la norma-1, partendo da un  $x_0$  già normalizzato, si ha  $\|x_1\|_1 = x_1 e = x_0 A e = x_0 e = \|x_0\|_1 = 1$

prodotto matrice vettore ad ogni iterazione, si semplifica dunque la procedura sfruttando la forma di  $A$

$$\begin{aligned}x_k^T = x_{k-1}^T A &\iff x_k = A^T x_{k-1} = \alpha W D^{-1} x_{k-1} + (1 - \alpha) \mathbf{e} \mathbf{x} e^T \cdot x_{k-1} \\&= \alpha W D^{-1} x_{k-1} + (1 - \alpha) \mathbf{e} \mathbf{x}\end{aligned}$$

Risulta evidente che questa riscrittura del *Power method* rende l'algoritmo 20 del tutto equivalente al 16, ovvero al metodo di *Jacobi*, tuttavia sono riportate entrambe le versioni poichè sono rielaborazioni di codici provenienti da fonti diverse.

## 5 Angoli tra i vettori approssimati

Uno dei possibili appropcci per lo studio della convergenza dei vettori approssimati dai metodi iterativi è analizzare i seni degli angoli compresi tra le varie stime e la soluzione esatta. Infatti più piccolo è il seno di tali angoli più è precisa l'approssimazione<sup>14</sup>. Valendo la seguente disuguaglianza

**Proposizione 5.1.**  $\sin \angle(x_1, x_2) \leq \sin \angle(x_1, x) + \sin \angle(x_2, x)$ .

è sufficiente stimare le quantità  $\sin \angle(x_k, x)$  per poter quantificare il *gap* che intercorre tra i diversi algoritmi. Il seguente Teorema 5.1 ci permette di approssimare queste quantità per alcuni dei metodi oggetto di studio.

**Teorema 5.1.** *Data  $D - \alpha W$  matrice simmetrica definita positiva ed il sistema  $(D - \alpha W)x = (1 - \alpha)x$ , denotando con  $x$  la soluzione esatta,  $\bar{x} = D^{-1}x$ ,  $\tilde{x} = D^{-\frac{1}{2}}\bar{x}$  e con  $x_k^{CG}$ ,  $x_k^{PCG}$ ,  $x_k^J$  le rispettive approssimazioni fornite dai metodi CG, PCG e Jacobi, valgono le seguenti disuguaglianze*

$$\begin{aligned}\sin \angle(x_k^{CG}, \bar{x}) &\leq \frac{1}{\sqrt{\lambda_{\min}(B)}} \|x_k^{CG} - \bar{x}\|_A \\ \sin \angle(x_k^{PCG}, \tilde{x}) &\leq \frac{1}{\sqrt{1-\alpha}} \|x_k^{PCG} - \tilde{x}\|_B \\ \sin \angle(x_k^J, x) &\leq \alpha^k \sqrt{n} \|x_0^J - x\|_2\end{aligned}$$

con  $A = D - \alpha W$  e  $B = I - \alpha D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ .

*Dimostrazione.* Nella dimostrazione viene usato più volte un importante risultato preliminare

$$\sin \angle(x_k, x) = \min_{\beta \in \mathbb{C}} \|x_k - \beta x\|_2 \leq \|x_k - x\|_2$$

CG. Essendo  $A$  una matrice simmetrica è ortogonalmente diagonalizzabile per il teorema spettrale, dunque  $A = U^T DU$  con  $U^T U = I$  (si denota  $D = (d_i)_{i=1}^N$ ).

$$\begin{aligned}\|x_k^{CG} - \bar{x}\|_A^2 &= (x_k^{CG} - \bar{x})^T A (x_k^{CG} - \bar{x}) \\ &= (U(x_k^{CG} - \bar{x}))^T D (U(x_k^{CG} - \bar{x})) \\ &= \sum_{i=1}^N (U(x_k^{CG} - \bar{x}))_i^2 d_i \\ &\geq \min_{i=1 \dots N} d_i \sum_{i=1}^N (U(x_k^{CG} - \bar{x}))_i^2 \\ &= \lambda_{\min}(A) \|x_k^{CG} - \bar{x}\|_2^2 \\ &\geq \lambda_{\min}(A) \sin \angle(x_k^{CG}, \bar{x})^2\end{aligned}$$

---

<sup>14</sup>si ricorda che la soluzione è unica a meno di normalizzazione

dove l'ultimo passaggio è vero per il risultato preliminare.  
 Segue la tesi dividendo per  $\lambda_{min}(A)$  e passando alla radice

$$\sin \angle(x_k^{CG}, \bar{x}) \leq \frac{1}{\sqrt{\lambda_{min}(A)}} \|x_k^{CG} - \bar{x}\|_A$$

*PCG.* In maniera del tutto analoga a quanto fatto per *CG* si può ottenere  
 $\sin \angle(x_k^{PCG}, \tilde{x}) \leq \frac{1}{\sqrt{\lambda_{min}(B)}} \|x_k^{PCG} - \tilde{x}\|_B$  ed essendo  $\lambda_{min}(B) \geq 1 - \alpha$  si ha

$$\sin \angle(x_k^{PCG}, \tilde{x}) \leq \frac{1}{\sqrt{1 - \alpha}} \|x_k^{PCG} - \tilde{x}\|_B$$

*Jacobi.* Partendo dalla scrittura esplicita del termine  $x_k^J = \alpha WD^{-1}x_{k-1}^J + (1 - \alpha)x$  si ottiene

$$\begin{aligned} \|x_k^J - x\|_2 &\leq \|(\alpha WD^{-1})^k(x_0^J - x)\|_2 \\ &\leq \|(\alpha WD^{-1})^k\|_2 \|(x_0^J - x)\|_2 \\ &\leq \sqrt{N} \|(\alpha WD^{-1})^k\|_1 \|(x_0^J - x)\|_2 \\ &\leq \sqrt{N} \alpha^k \|(x_0^J - x)\|_2 \end{aligned}$$

dove si è usata la proprietà delle norme di matrice indotte  $\|Mx\|_2 \leq \|M\|_2 \|x\|_2$ , che  $\forall M \in \mathcal{M}(\mathbb{R}, N) \|M\|_2 \leq \sqrt{N} \|M\|_1$  e che  $\|(\alpha WD^{-1})^k\|_1 \leq \|WD^{-1}\|_1^k = 1$ . Segue dal risultato preliminare che

$$\sin \angle(x_k^J, x) \leq \alpha^k \sqrt{N} \|(x_0^J - x)\|_2$$

□

## 6 Sperimentazioni numeriche

In questa sezione si mostrano i risultati di alcune sperimentazioni numeriche effettuate con **Matlab R2020a**.

Nella prima è stato utilizzato il set di dati **data\_GR\_Morrison.mat** (Julie L. Morrison [2]), contenente tre matrici di adiacenza di grafi genetici ed i tre rispettivi vettori di cambiamento dell'espressione dei geni, costruite usando il database *GO*: (**w\_All**, **expr\_data**), (**w\_Down**, **expr\_dataDown**) e (**w\_Up**, **expr\_dataUp**).

Successivamente si è utilizzata una matrice binaria simmetrica  $W \in \mathcal{M}(\{0, 1\}, 300000)$  generata aleatoriamente per verificare se il comportamento degli algoritmi fosse il medesimo anche per matrici di grandi dimensioni.

Infine si è cercato di verificare il risultato derivante dall'applicazione del Teorema 5.1 usando la matrice **w\_All**. Sono stati messi a confronto gli angoli formati tra le soluzioni approssimate dai vari metodi e la soluzione *esatta* calcolata risolvendo direttamente il sistema lineare. Il confronto ha coinvolto tutti i metodi anche se *CG*, *PCG* e *Jacobi* sono i soli sui quali si sono dimostrati risultati teorici.

Tutti i test sono stati effettuati su un computer che monta un processore **AMD Ryzen 7 3800X**, 8-Core 3.90 GHz e che dispone di 16,0 GB di RAM con sistema operativo **Windows 10**, 64-bit.

### 6.1 Sperimentazione 1: **w\_All**, **w\_Up** e **w\_Down**

A seguire vengono prese come matrici di riferimento **w\_All**, **w\_Up** e **w\_Down** con i vettori ad esse associati e, fissando l'errore di approssimazione a  $10^{-14}$  e utilizzando diversi valori per il damping factor  $\alpha$ , si ottengono i risultati riportati nelle successive tabelle. Sono evidenziati i tempi di esecuzione dei singoli metodi ed il numero di iterazioni effettuate prima di raggiungere la soglia fissata per l'errore.

Tabella 1: **w\_All**: n. di iterazioni / tempo in secondi

$\alpha$	0.50	0.70	0.85	0.99
<b>CG</b>	318 / 0.280s	366 / 0.315s	418 / 0.351s	574 / 0.468s
<b>PCG</b>	23 / 0.018s	31 / 0.024s	42 / 0.032s	67 / 0.052s
<b>Jacobi</b>	40 / 0.010s	72 / 0.019s	152 / 0.039s	2171 / 0.599s
<b>Arnoldi</b>	28 / 0.015s	40 / 0.021s	60 / 0.025s	112 / 0.046s
<b>Gauss-Seidel</b>	48 / 0.012s	84 / 0.018s	170 / 0.036s	2342 / 0.405s
<b>Power</b>	40 / 0.010s	72 / 0.019s	152 / 0.040s	2171 / 0.560s
<b>Richardson</b>	62 / 0.017s	106 / 0.029s	217 / 0.059s	3016 / 0.810s

Tabella 2: `w_Down`: n. di iterazioni / tempo in secondi

$\alpha$	0.50	0.70	0.85	0.99
CG	228 / 0.067s	261 / 0.074s	300 / 0.082s	411 / 0.101s
PCG	23 / 0.004s	32 / 0.006s	45 / 0.009s	73 / 0.013s
Jacobi	42 / 0.002s	76 / 0.003s	160 / 0.007s	2293 / 0.083s
Arnoldi	32 / 0.004s	44 / 0.004s	60 / 0.006s	132 / 0.011s
Gauss-Seidel	50 / 0.002s	90 / 0.03s	184 / 0.010s	2620 / 0.082s
Power	42 / 0.002s	76 / 0.003s	160 / 0.006s	2293 / 0.083s
Richardson	67 / 0.003s	113 / 0.004s	230 / 0.010s	3228 / 0.148s

Tabella 3: `w_Up`: n. di iterazioni / tempo in secondi

$\alpha$	0.50	0.70	0.85	0.99
CG	261 / 0.103s	305 / 0.118s	353 / 0.130s	466 / 0.168s
PCG	23 / 0.007s	31 / 0.009s	43 / 0.013s	68 / 0.020s
Jacobi	40 / 0.003s	75 / 0.006s	160 / 0.015s	2511 / 0.212s
Arnoldi	32 / 0.006s	40 / 0.007s	60 / 0.009s	112 / 0.015s
Gauss-Seidel	50 / 0.004s	90 / 0.008s	184 / 0.011s	2606 / 0.158s
Power	40 / 0.003s	75 / 0.006s	160 / 0.012s	2510 / 0.220s
Richardson	64 / 0.005s	111 / 0.009s	228 / 0.019s	3197 / 0.267s

Risulta che i due metodi più rapidi al variare del *damping factor* sono Arnoldi e PCG, con quest'ultimo che impiega qualche millesimo di secondo in più per convergere ma esegue meno iterazioni rispetto al primo.

Di seguito sono riportate le matrici `w_All`, `w_Up` e `w_Down` ottenute con il comando `spy(·)` di Matlab.

w\_All / w\_Up / w\_Down

---

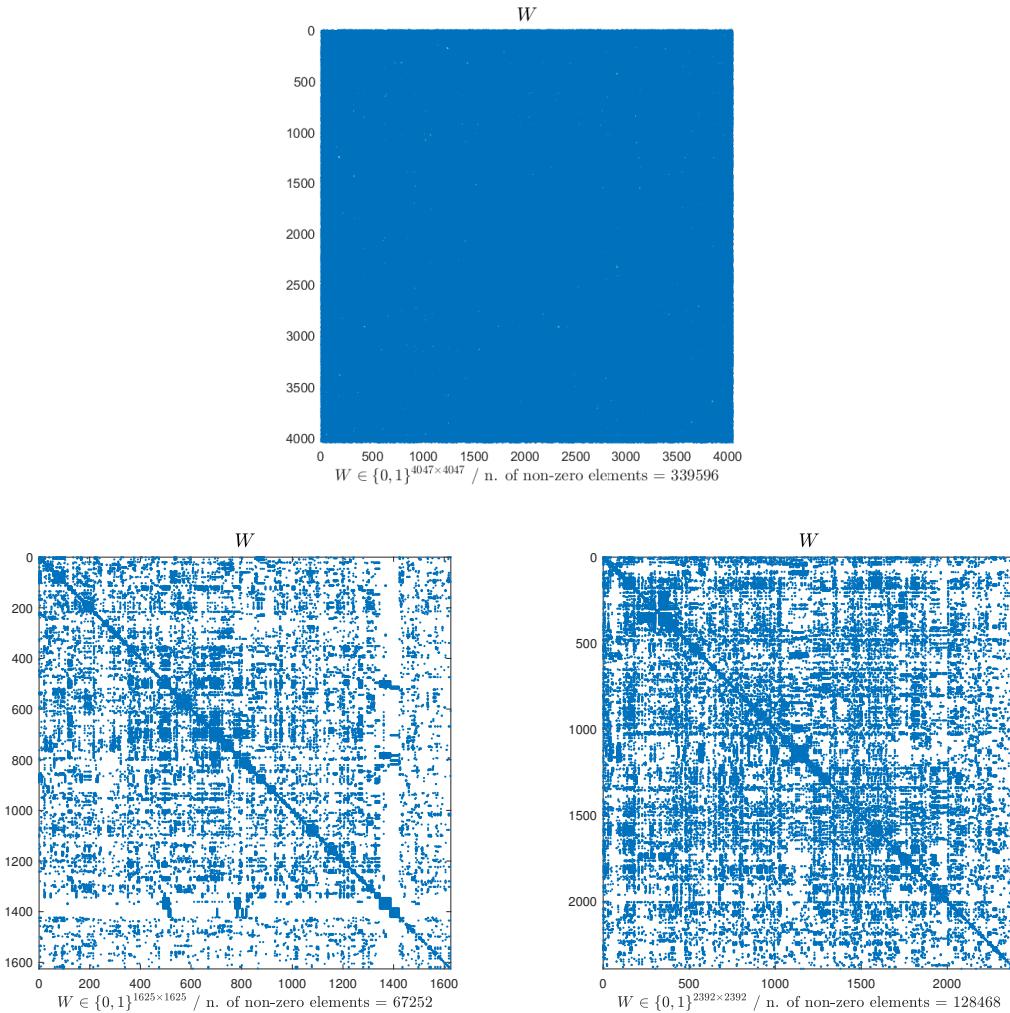


Figura 1: w\_All in alto, w\_Down a sinistra e w\_Up a destra

Per poter studiare al meglio l'andamento degli algoritmi si analizza la variazione della norma del residuo per i vari metodi in funzione del numero di iterazioni. Più precisamente  $\|r\|_1$  viene messa in funzione del numero di prodotti matrice-vettore ad ogni iterazione, trattandosi delle operazioni più *costose*<sup>15</sup>. I grafici seguenti mostrano proprio questi andamenti per tutte le matrici e per i vari valori del *damping factor*  $\alpha$ .

<sup>15</sup>si precisa che per il metodo di *Gauss-Seidel* in realtà l'operazione più costosa non è data solo dal prodotto matrice vettore che vale  $O(n^2)$ , ma anche dalla risoluzione di un sistema lineare con una matrice triangolare superiore, che costa anch'esso  $O(n^2)$

### Plot dell'andamento dell'errore per w\_All

---

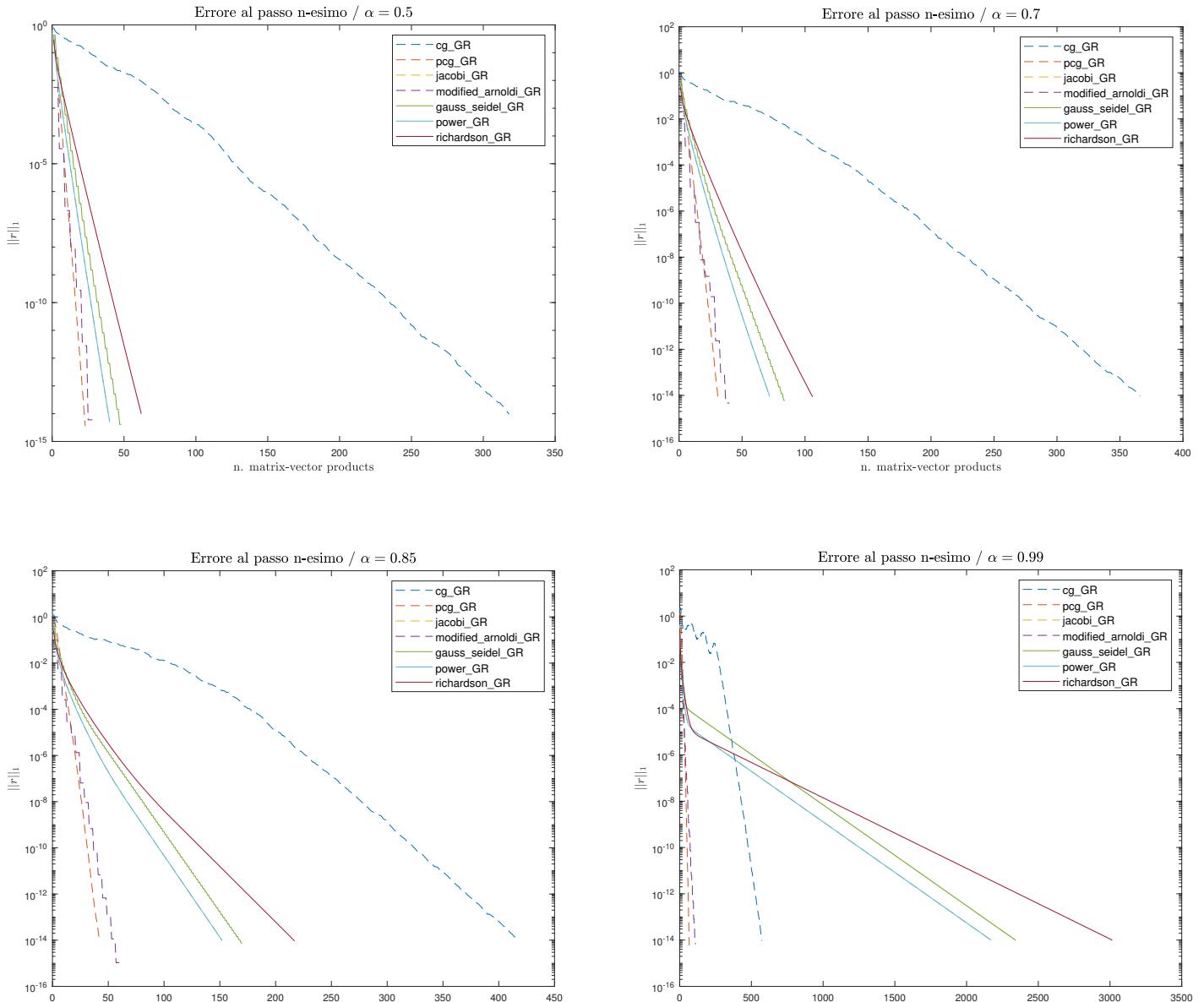


Figura 2: ↙  $\alpha = 0.50$  / ↗  $\alpha = 0.70$  / ↘  $\alpha = 0.85$  / ↘  $\alpha = 0.99$

### Plot dell'andamento dell'errore per w\_Down

---

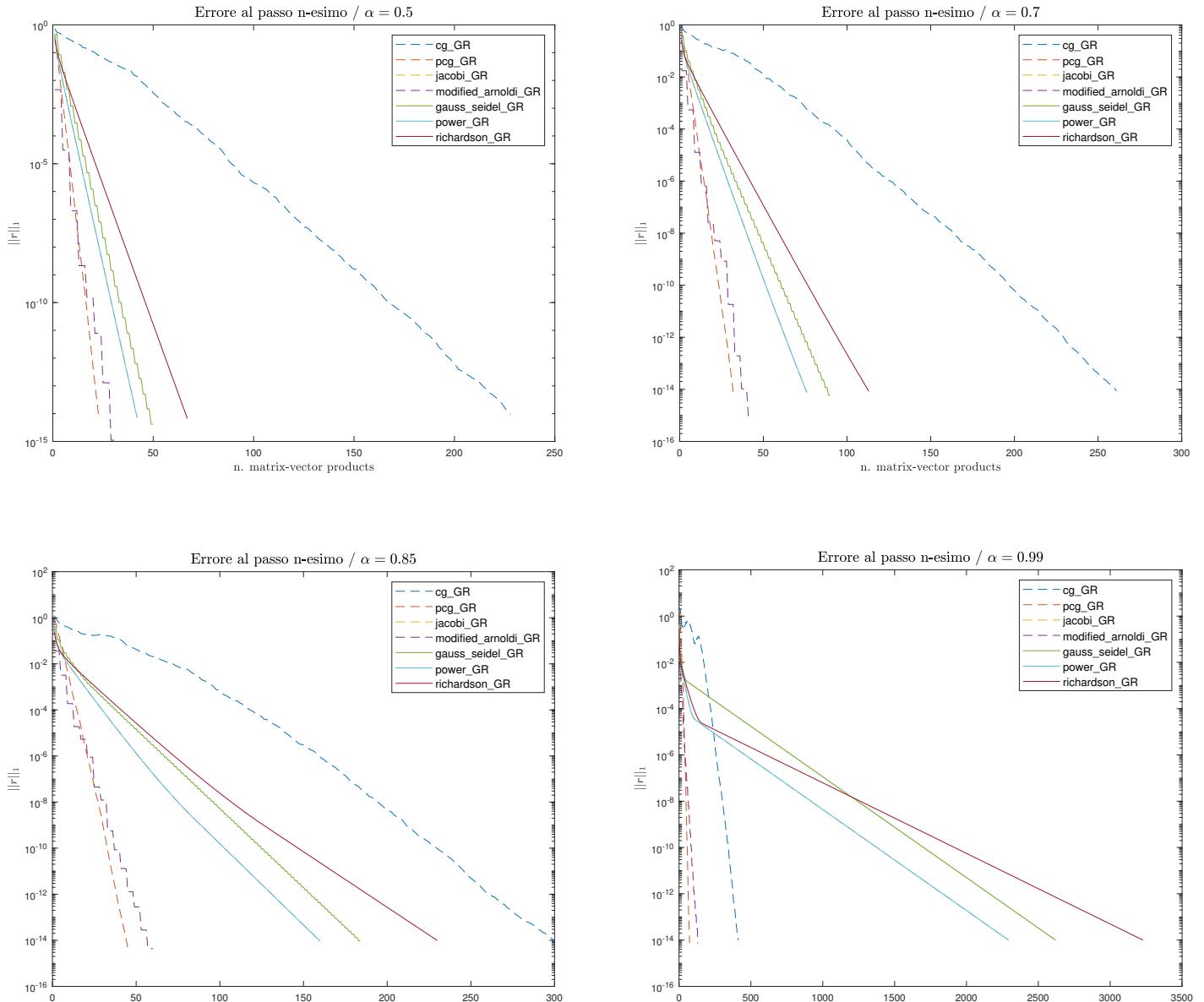


Figura 3: ↙  $\alpha = 0.50$  / ↗  $\alpha = 0.70$  / ↘  $\alpha = 0.85$  / ↘  $\alpha = 0.99$

### Plot dell'andamento dell'errore per w\_Up

---

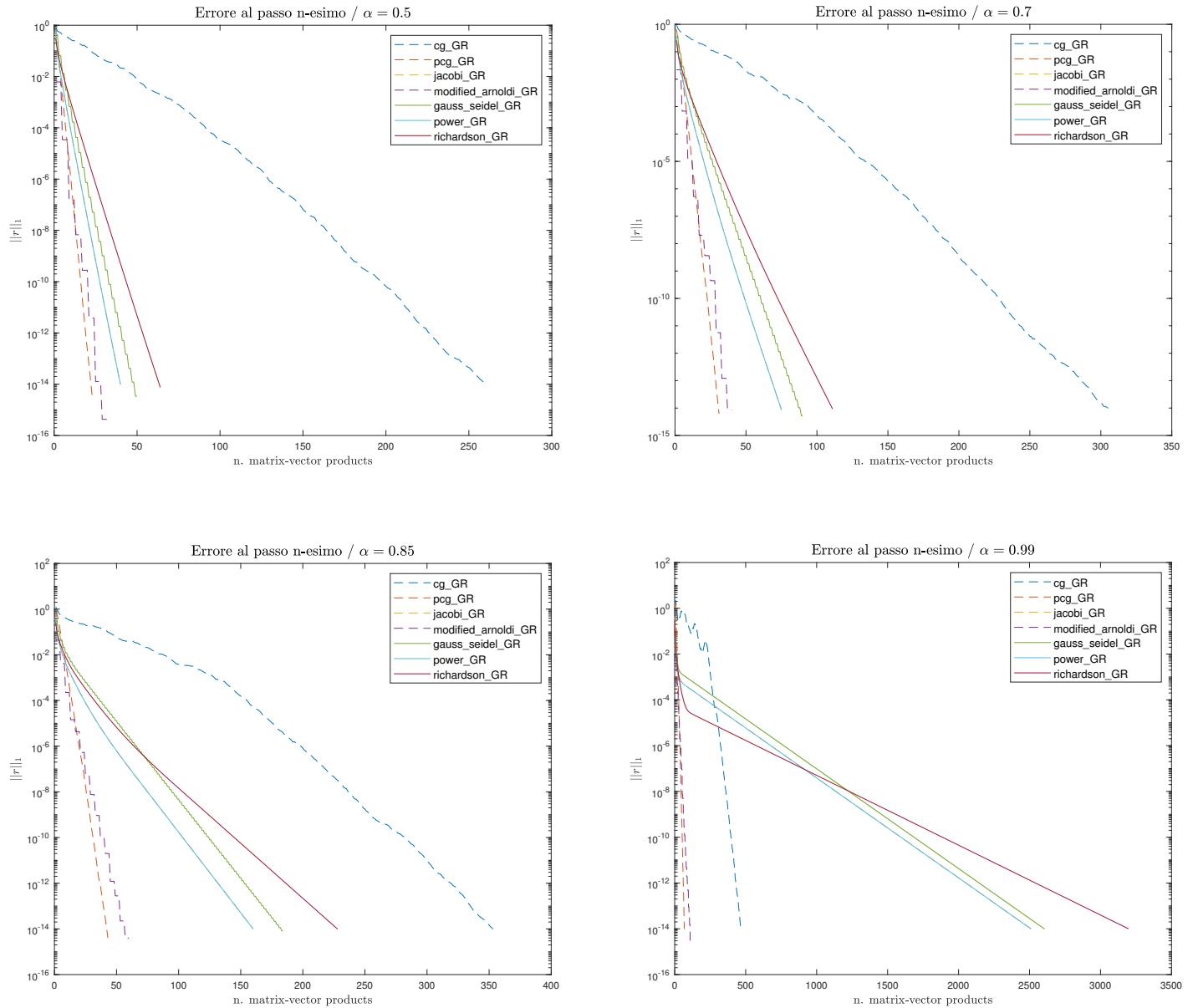


Figura 4: ↙  $\alpha = 0.50$  / ↗  $\alpha = 0.70$  / ↘  $\alpha = 0.85$  / ↘  $\alpha = 0.99$

Il miglioramento apportato dal precondizionamento al metodo del *gradiente coniugato* è considerevole e risulta subito evidente dai risultati riportati nelle tabelle 1, 2 e 3 e dai plot 2, 3 e 4. L'efficacia del Teorema 3.3 è dimostrata in maniera ancora più esplicita nelle figure seguenti. Si è preso come valore di riferimento  $\alpha = 0.85$ , dunque si ha che  $\forall \lambda \in Sp(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}})$  vale  $\lambda \in [0.15, 0.85]$ . In particolare

-  $W = w\_All$

$$\lambda_{max}(D - \alpha W) = 493.86 \quad \text{e} \quad \lambda_{min}(D - \alpha W) = 0.15$$

$$\lambda_{max}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 1.85 \quad \text{e} \quad \lambda_{min}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 0.15$$

-  $W = w\_Down$

$$\lambda_{max}(D - \alpha W) = 194.07 \quad \text{e} \quad \lambda_{min}(D - \alpha W) = 0.15$$

$$\lambda_{max}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 1.85 \quad \text{e} \quad \lambda_{min}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 0.15$$

-  $W = w\_Up$

$$\lambda_{max}(D - \alpha W) = 289.89 \quad \text{e} \quad \lambda_{min}(D - \alpha W) = 0.15$$

$$\lambda_{max}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 1.85 \quad \text{e} \quad \lambda_{min}(I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}) = 0.15$$

Autovalori di  $D - \alpha W$  e  $I - \alpha D^{-\frac{1}{2}} WD^{-\frac{1}{2}}$  con  $W = w\_All$

---

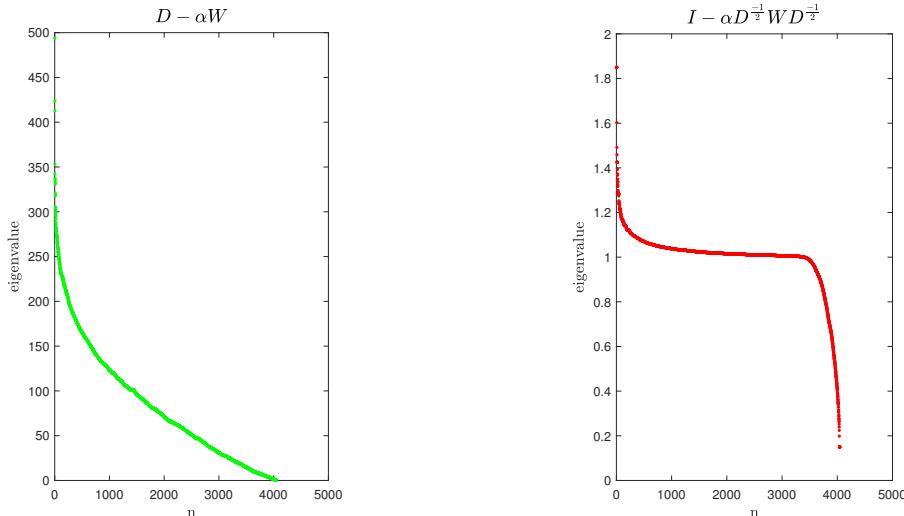


Figura 5:  $\leftarrow$  autovalori prima di applicare il precondizionamento /  $\rightarrow$  autovalori dopo aver applicato il precondizionamento

---

**Autovalori di  $D - \alpha W$  e  $I - \alpha D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$  con  $W = w_{Down}$**

---

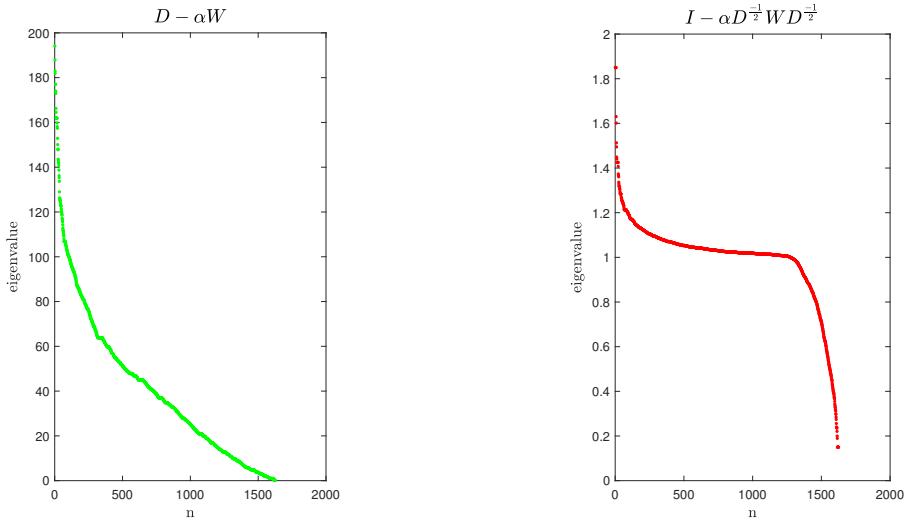


Figura 6:  $\leftarrow$  autovalori prima di applicare il precondizionamento /  $\rightarrow$  autovalori dopo aver applicato il precondizionamento

---

**Autovalori di  $D - \alpha W$  e  $I - \alpha D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$  con  $W = w_{Up}$**

---

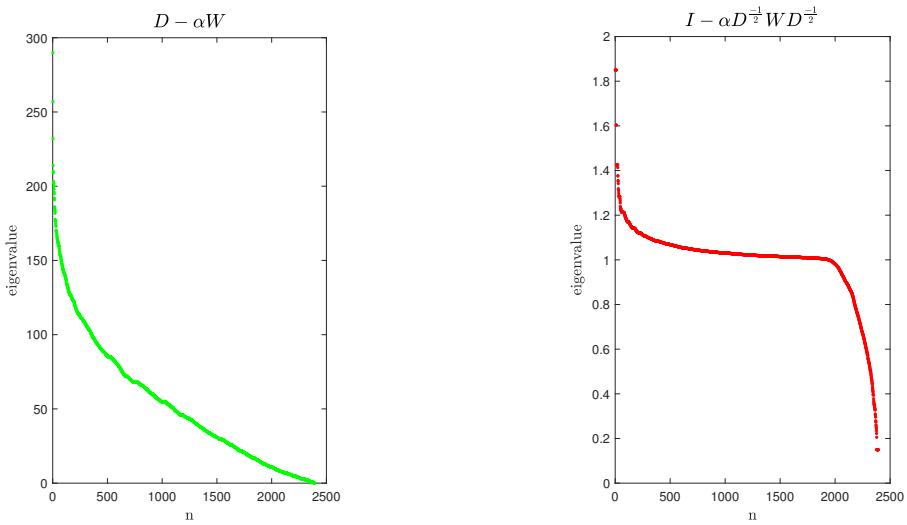


Figura 7:  $\leftarrow$  autovalori prima di applicare il precondizionamento /  $\rightarrow$  autovalori dopo aver applicato il precondizionamento

## 6.2 Sperimentazione 2: matrice binaria di grandi dimensioni

Il modo più diretto per verificare la stabilità degli algoritmi che si stanno utilizzando è utilizzare matrici di taglia grande. Con i seguenti codici si è prodotta una matrice  $W$ ,  $300000 \times 300000$  simmetrica con densità<sup>16</sup>  $\frac{7}{300000}$  ed un vettore *di personalizzazione*  $\mathbf{ex}$  neutrale  $\mathbf{ex} = \frac{\mathbf{e}}{n}$ , la si è salvata nel file `random_300000.mat` e successivamente con il comando `load(·)` la si è caricata per poter essere utilizzata nello stesso script della sperimentazione 6.1. Si è scelto di usare una tolleranza di  $10^{-8}$ , più grande della precedente  $10^{-14}$ , in quanto altrimenti i tempi di elaborazione sarebbero stati eccessivi.

---

```

1 function A = random_GR_matrix(n,k)
2 % Random GeneRank matrix
3 % Generate a boolean symmetric matrix of size n and density k/n. Possible input of
4 % GeneRank problem
5 %
6 %      n := size of the matrix
7 %      k := density (k/n)
8 %
9 %      output:
10 %          A := matrix for GeneRank
11 %
12 %      example:
13 %          A = random_GR_matrix(10000,7);
14
15 A = sprandsym(n,k/n);
16 A = A ~= 0;
17 A = A - diag(diag(A));

```

---

Figura 8: intput/random\_GR\_matrix

---

```

1 % build the matrix and the vector
2 n = 300000;
3 W = random_GR_matrix(n,7);
4 ex = ones(n, 1)/n;
5 W = sparse(W);
6 ex = sparse(ex);

```

---

Figura 9: salvataggio e caricamento di  $W$  ed  $\mathbf{ex}$  (comandi estratti dallo script `main.m`, non presente nella relazione)

Nella seguente tabella 4 sono riportati i risultati dell'esperimento, seguiti dall'output di `spy(W)` e dal grafico che mostra la relazione tra le norme dei residui ed il numero di prodotti matrice-vettore.

Anche questa sperimentazione conferma che *CG* e *PCG* sono tra i metodi più efficienti

<sup>16</sup>si è scelto di avere circa 7 posizioni non nulle per ciascuna riga di  $W$  in quanto questa è solitamente la densità delle matrici del web

assieme ad *Arnoldi*; sorprendentemente anche il metodo di *Richardson* risulta molto stabile al crescere della taglia della matrice. In particolare il *PCG* risulta essere l'algoritmo più rapido e più efficiente (esegue un numero minore di prodotti matrice vettore) anche al crescere di  $\alpha$  e di `size(W)`.

Tabella 4: `random_300000`: n. di iterazioni / tempo in secondi

$\alpha$	0.50	0.70	0.85	0.99
CG	7 / 2.319s	8 / 2.635s	7 / 2.319s	7 / 2.298s
PCG	4 / 1.331s	4 / 1.312s	5 / 1.639s	5 / 1.643s
Jacobi	27 / 3.499s	50 / 6.309s	100 / 12.988s	473 / 62.242s
Arnoldi	4 / 1.412s	8 / 1.960s	8 / 1.930s	8 / 1.735 s
Gauss-Seidel	26 / 3.362s	44 / 5.674s	86 / 11.106s	1062 / 136.053s
Power	27 / 3.481s	50 / 6.462s	100 / 13.161s	473 / 59.946s
Richardson	8 / 1.062s	12 / 1.576s	16 / 2.069s	20 / 2.462s

$W = \text{random\_}300000$

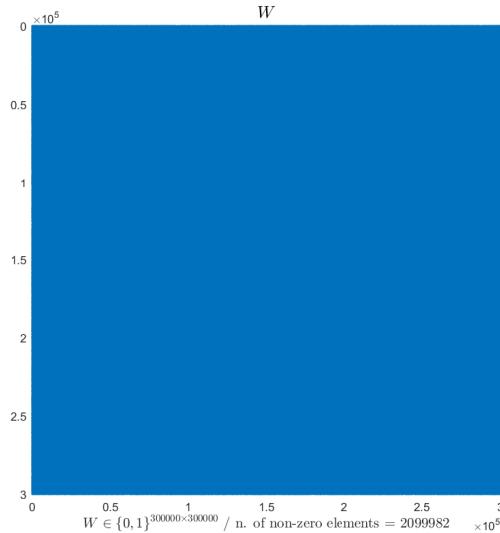


Figura 10:  $W$ ,  $300000 \times 300000$  simmetrica con densità  $\frac{7}{300000}$

### Plot dell'andamento dell'errore per random\_300000

---

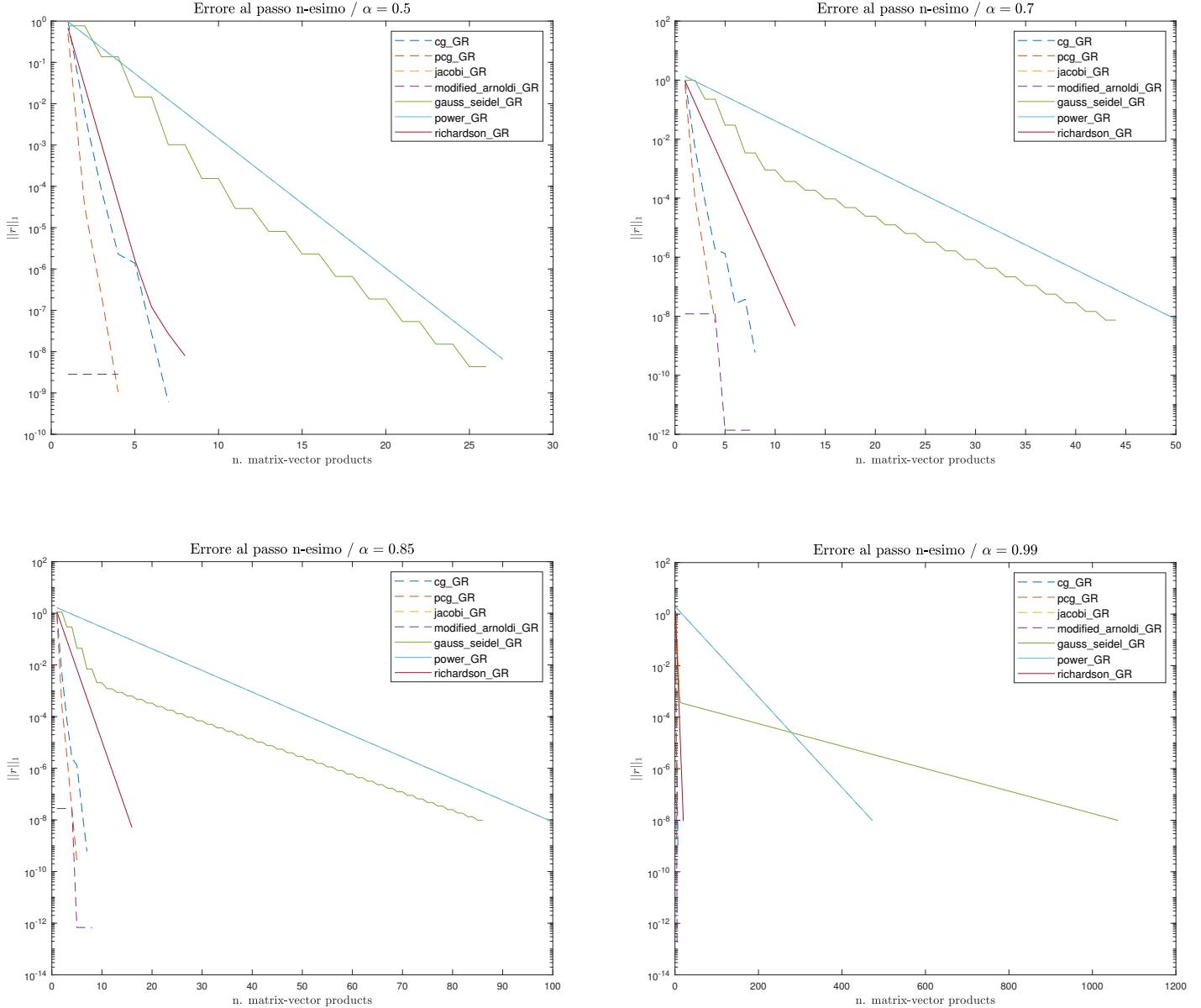


Figura 11: ↖  $\alpha = 0.50$  / ↗  $\alpha = 0.70$  / ↙  $\alpha = 0.85$  / ↘  $\alpha = 0.99$

### 6.3 Sperimentazione 3: angoli tra vettori approssimati ed esatti

Per quanto detto nella sezione 5, per analizzare la *correlazione* tra i vettori approssimati dai vari metodi iterativi e la soluzione *esatta* è sufficiente avere a disposizione i valori  $\sin \angle(x_k, x)$  per ciascun  $x_k$  dato dagli algoritmi considerati. Si ricorda che più è piccolo ciascuno di questi numeri, più è alta la correlazione con il risultato esatto del problema. Come fatto nell'articolo preso a riferimento (Gang Wu [1]) si procede al calcolo di  $\sin \angle(x, y)$  nel modo seguente (per maggiori dettagli si veda il codice 24):

$$\sin \angle(x, y) = \left\| \left( I - \frac{xx^T}{\|x\|_2 \|x^T\|_2} \right) \frac{y}{\|y\|_2} \right\|_2 = \left\| \frac{y}{\|y\|_2} - \frac{x}{\|x\|_2} \left( \frac{x^T y}{\|x^T\|_2 \|y\|_2} \right) \right\|_2$$

Ne segue, guardando i risultati nella tabella 5, che all'aumentare di  $\alpha$  *CG* e *PCG* approssimano *meglio* la soluzione corretta del sistema dato da `w_All` e `expr_data` di quanto non facciano tutti gli altri metodi.

Tabella 5:  $\sin \angle(y, x)$  al variare di  $y$  soluzione approssimata dai vari metodi

$\alpha$	0.50	0.70	0.85	0.99
$\sin \angle(x^{CG}, x)$	$6.936 \times 10^{-15}$	$4.761 \times 10^{-15}$	$2.370 \times 10^{-15}$	$1.564 \times 10^{-15}$
$\sin \angle(x^{PCG}, x)$	$3.191 \times 10^{-15}$	$4.816 \times 10^{-15}$	$4.213 \times 10^{-15}$	$8.766 \times 10^{-15}$
$\sin \angle(x^J, x)$	$2.716 \times 10^{-14}$	$2.088 \times 10^{-13}$	$5.439 \times 10^{-13}$	$9.077 \times 10^{-12}$
$\sin \angle(x^A, x)$	$3.923 \times 10^{-14}$	$3.972 \times 10^{-14}$	$2.835 \times 10^{-14}$	$6.021 \times 10^{-13}$
$\sin \angle(x^{GS}, x)$	$1.347 \times 10^{-15}$	$5.635 \times 10^{-15}$	$3.561 \times 10^{-14}$	$1.636 \times 10^{-12}$
$\sin \angle(x^P, x)$	$2.716 \times 10^{-14}$	$2.088 \times 10^{-13}$	$5.439 \times 10^{-13}$	$9.077 \times 10^{-12}$
$\sin \angle(x^R, x)$	$7.480 \times 10^{-14}$	$4.148 \times 10^{-13}$	$1.020 \times 10^{-12}$	$1.709 \times 10^{-11}$

## 7 Codice

In questa sezione è riportata la parte sostanziale del codice sorgente utilizzato per effettuare le sperimentazioni numeriche. Non sono stati aggiunti gli script per generare i plot e raccogliere i risultati per non appesantire troppo la relazione.

La prima porzione di codici di che segue è la parte centrale del progetto: si tratta delle trascrizioni in Matlab dei metodi iterativi sopra citati.

### Metodo del gradiente coniugato

---

```
1 function [x, err] = cg_GR(W,ex,alpha,tol)
2 % CG for GeneRank
3 % Conjugate gradient algorithm applied to a GeneRank problem
4 %
5 % input:
6 %     W := GeneRank matrix
7 %     ex := constant term of the system
8 %     alpha := damping factor
9 %     tol := tolerance factor
10 % output
11 %     x := estimate solution
12 %
13 % example
14 %     x = cg_GR(W, ex, 0.85, 1e-14);
15 %
16 % Source:
17 % "A preconditioned conjugate gradient algorithm for GeneRank with application to
18 % microarray data mining"
19 % Gang Wu, Wei Xu, Ying Zhang, Yimin Wei
20 % (important changes have been made)
21 fprintf("--- \n1. cg_GR \n");
22
23 n = size(W,1);
24
25 ex = abs(ex);
26 ex = ex/norm(ex, 1);
27
28 e = ones(n,1);
29 d = W*e;
30
31 % inizializations: k, x, r, p, res, err
32 k = 0;
33 x = zeros(n, 1);
34 r = ex;
35 p = ex;
36 res = 1;
37 err = [];
38
39 % short definitions
40 W_complete = alpha*W;
```

---

Figura 12: methods/cg\_GR (parte 1)

---

```
42 % loop
43 tic;
44 while (res > tol)
45     k = k + 1;
46
47     r_old = r;
48
49     % first loop, inizializations: z, nu, mu
50     z = d.*p - W_complete*p;
51     nu = (r_old'*r_old)/(p'*z);
52     x = x + nu*p;
53     r = r_old - nu*z;
54     mu = (r'*r)/(r_old'*r_old);
55     p = r + mu*p;
56
57     res = norm(r,1);
58     err = [err res];
59 end
60
61 x = d.*x;
62 x = x/norm(x,1);
63
64 t = toc;
65
66 plot_function(alpha,1,err,k,t,x);
```

---

Figura 13: methods/cg\_GR (parte 2)

## Metodo del gradiente coniugato con precondizionamento

```
1 function x = pcg_GR(W,ex,alpha,tol)
2 % PCG for GeneRank
3 % Preconditioned conjugate gradient algorithm applied to a GeneRank problem
4 %
5 % input:
6 %     W := GeneRank matrix
7 %     ex := constant term of the system
8 %     alpha := damping factor
9 %     tol := tolerance factor
10 % output
11 %     x := estimate solution
12 %
13 % example
14 %     x = pcg_GR(W,ex,0.85,1e-14);
15 %
16 % Source:
17 % "A preconditioned conjugate gradient algorithm for GeneRank with application to
18 % microarray data mining"
19 % Gang Wu, Wei Xu, Ying Zhang, Yimin Wei
20 % (important changes have been made)
21 fprintf("--- \n2. pcg_GR \n");
22
23 n = size(W,1);
24
25 ex = abs(ex);
26 ex = ex/norm(ex, 1);
27
28 e = ones(n,1);
29 d = W*e;
30 d_inv = 1./d;
31
32 % inizializations: k, x, r, p, y, res, err
33 k = 0;
34 x = zeros(n, 1);
35 r = ex;
36 p = ex.*d_inv;
37 y = p;
38 res = 1;
39 err = [];
40
41 % short definitions
42 W_complete = alpha*W;
```

Figura 14: methods/pcg\_GR (parte 1)

---

```
44 % loop
45 tic;
46 while (res > tol)
47     k = k + 1;
48
49     y_old = y;
50     r_old = r;
51
52     % first loop, inizializations: z, nu, mu
53     z = d.*p - W_complete*p;
54     nu = (y_old'*r_old)/(p'*z);
55     x = x + nu*p;
56     r = r_old - nu*z;
57     y = r.*d_inv;
58     mu = (y'*r)/(y_old'*r_old);
59     p = y + mu*p;
60
61     res = norm(r,1);
62     err = [err res];
63 end
64
65 x = d.*x;
66 x = x/norm(x,1);
67
68 t = toc;
69
70 plot_function(alpha,2,err,k,t,x);
```

---

Figura 15: methods/pcg-GR (parte 2)

## Metodo di Jacobi

---

```
1 function x = jacobi_GR(W,ex,alpha,tol)
2 % Jacobi for GeneRank
3 % Jacobi algorithm applied to a GeneRank problem
4 %
5 %   input:
6 %     W := GeneRank matrix
7 %     ex := constant term of the system
8 %     alpha := damping factor
9 %     tol := tolerance factor
10 %   output
11 %     x := estimate solution
12 %
13 %   example
14 %     x = jacobi_GR(W,ex,0.85,1e-14);
15 %
16 % Source:
17 % "A preconditioned conjugate gradient algorithm for GeneRank with application to
18 % microarray data mining"
19 % Gang Wu, Wei Xu, Ying Zhang, Yimin Wei
20 % (important changes have been made)
21 fprintf("--- \n3. jacobi_GR \n");
22
23 n = size(W,1);
24
25 ex = abs(ex);
26 ex = ex/norm(ex, 1);
27
28 e = ones(n,1);
29 d = W*e;
30 d_inv = 1./d;
31
32 % inizializations: k, x, res, err
33 k = 0;
34 x = e/n;
35 res = 1;
36 err = [];
37
38 % short definitions
39 ex_complete = (1 - alpha)*ex;
40 W_complete = alpha*W;
```

---

Figura 16: methods/jacobi\_GR (parte 1)

---

```
42 % loop
43 tic;
44 while (res > tol)
45     k = k + 1;
46
47     x_old = x;
48
49     x = W_complete*(d_inv.*x) + ex_complete;
50
51     res = norm(x - x_old,1);
52     err = [err res];
53 end
54
55 x = x/norm(x,1);
56
57 t = toc;
58
59 plot_function(alpha,3,err,k,t,x);
```

---

Figura 17: methods/jacobi\_GR (parte 2)

## Metodo di Gauss-Seidel

---

```
1 function x = gauss_seidel_GR(W,ex,alpha,tol)
2 % Gauss-Seidel for GeneRank
3 % Gauss-Sidel algorithm applied to a GeneRank problem
4 %
5 %   input:
6 %     W := GeneRank matrix
7 %     ex := constant term of the system
8 %     alpha := damping factor
9 %     tol := tolerance factor
10 %   output
11 %     x := estimate solution
12 %
13 %   example
14 %     x = gauss_seidel_GR(W,ex,0.85,1e-14);
15 %
16 % Source:
17 % Appunti del corso di Calcolo Scientifico
18 % Dario Andrea Bini e Lidia Aceto
19 % (important changes have been made)
20
21 fprintf("--- \n5. gauss_seidel_GR \n");
22
23 n = size(W,1);
24
25 ex = abs(ex);
26 ex = ex/norm(ex, 1);
27
28 e = ones(n,1);
29 d = W*e;
30 % in case of non-symmetric matrix
31 % dang = d==0;
32 % d = d + dang*n;
33 d_inv = 1./d;
34
35 % inizializations: k, x, res, err
36 k = 0;
37 x = e/n;
38 res = 1;
39 err = [];
40
41 % computations: L, U
42 L = alpha*tril(W,-1);
43 U = speye(n) - triu(W)*diag(sparse(alpha*d_inv));
44
45 % short definitions
46 ex_complete = (1 - alpha)*ex;
```

---

Figura 18: methods/gauss\_seidel\_GR (parte 1)

---

```
48 % loop
49 tic;
50 while (res > tol)
51     k = k + 2;
52
53     x_old = x;
54
55     x = d_inv.*x;
56     x = L*x + ex_complete;
57     % in case of non-symmetric matrix
58     % x = L*x + alpha*sum(dang.*x) + ex_complete;
59     x = U\x;
60
61     res = norm(x - x_old,1);
62     err = [err res res];
63 end
64
65 x = x/norm(x,1);
66
67 t = toc;
68
69 plot_function(alpha,5,err,k,t,x);
```

---

Figura 19: methods/gauss\_seidel\_GR (parte 2)

## Metodo delle potenze

```
1 function x = power_GR(W,ex,alpha,tol)
2 % Power method for GeneRank
3 % Power algorithm applied to a GeneRank problem
4 %
5 %    input:
6 %        W := GeneRank matrix
7 %        ex := constant term of the system
8 %        alpha := damping factor
9 %        tol := tolerance factor
10 %   output
11 %        x := estimate solution
12 %
13 %   example
14 %       x = power_GR(W,ex,0.85,1e-14);
15 % Source:
16 % Appunti del corso di Calcolo Scientifico
17 % Dario Andrea Bini e Lidia Aceto
18 % (important changes have been made)
19
20 fprintf("--- \n6. power_GR \n");
21
22 n = size(W,1);
23
24 ex = abs(ex);
25 ex = ex/norm(ex, 1);
26
27 e = ones(n,1);
28 d = W*e;
29 % in case of non-symmetric matrix
30 % dang = d==0;
31 % d = d + dang*n;
32 d_inv = 1./d;
33
34 % inizializations: k, x, res, err
35 k = 0;
36 x = e/n;
37 res = 1;
38 err = [];
39
40 % short definitions
41 ex_complete = (1 - alpha)*ex;
```

Figura 20: methods/power\_GR (parte 1)

---

```
43 % loop ( A = alpha*W*D_inv + (1 - alpha)*ex*ones(n,1)' )
44 tic;
45 while res > tol
46     k = k + 1;
47
48     x_old = x;
49
50     x = d_inv.*x;
51     x = W*x;
52     % in case of non-symmetric matrix
53     % x = W*x + sum(dang.*x);
54     x = alpha*x + ex_complete;
55
56     res = norm(x - x_old, 1);
57     err = [err res];
58 end
59
60 x = x/norm(x,1);
61
62 t = toc;
63
64 plot_function(alpha,6,err,k,t,x);
```

---

Figura 21: methods/power\_GR (parte 2)

## Metodo di Richardson

---

```
1 function x = richardson_GR(W,ex,alpha,tol)
2 % Richardson for GeneRank
3 % Richardson algorithm applied to a GeneRank problem
4 %
5 %   input:
6 %     W := GeneRank matrix
7 %     ex := constant term of the system
8 %     alpha := damping factor
9 %     tol := tolerance factor
10 %   output
11 %     x := estimate solution
12 %
13 %   example
14 %     x = richardson_GR(W,ex,0.85,1e-14);
15 %
16 % Source:
17 % Appunti del corso di Calcolo Scientifico
18 % Dario Andrea Bini e Lidia Aceto
19 % (important changes have been made)
20
21 fprintf("--- \n7. richardson_GR \n");
22
23 % beta
24 beta = 0.7;
25
26 n = size(W,1);
27
28 ex = abs(ex);
29 ex = ex/norm(ex,1);
30
31 e = ones(n,1);
32 d = W*e;
33 d_inv = 1./d;
34
35 % inizializations: k, x, res, err
36 k = 0;
37 x = e/n;
38 res = 1;
39 err = [];
40
41 % short definitions
42 ex_complete = beta*(1 - alpha)*ex;
43 W_complete = beta*alpha*W;
```

---

Figura 22: methods/richardson\_GR (parte 1)

---

```
45 % loop
46 tic;
47 while (res > tol)
48     k = k + 1;
49
50     x_old = x;
51
52     x = (1 - beta)*x + W_complete*(d_inv.*x) + ex_complete;
53
54     res = norm(x - x_old,1);
55     err = [err res];
56 end
57
58 x = x/norm(x,1);
59
60 t = toc;
61
62 plot_function(alpha,7,err,k,t,x);
```

---

Figura 23: methods/richardson\_GR (parte 2)

Di seguito sono riportate alcune delle funzioni più rilevanti utilizzate per l'esecuzione delle sperimentazioni numeriche: `sin_angle.m` serve a calcolare il seno dell'angolo compreso tra la soluzione esatta e quella approssimata e `remove_dangling.m` è la funzione usata per rimuovere i *dangling nodes* dalle matrici di partenza  $W$ .

---

```

1 function sin_angle(x, xreal)
2 % Compute and print $\sin\angle(x, xreal)$
3 %
4 %   input:
5 %     x := estimate solution
6 %     xreal := solution of GeneRank problem
7 %
8 %   example
9 %     sin_angle(x, xreal);
10
11 xnorm = x/norm(x,2);
12 xrealnorm = xreal/norm(xreal,2);
13
14 tmp = xnorm' * xrealnorm;
15 tmp = xnorm * tmp;
16
17 s = norm(xrealnorm - tmp,2);
18
19 fprintf("    sin angle(x, xreal) = %e\n", s);

```

---

Figura 24: output/sin\_angle

```

1 function W = remove_dangling(W)
2 % Remove dangling nodes
3 %
4 %   input:
5 %     W := GeneRank matrix
6 %
7 %   output:
8 %     W := GeneRank matrix without dangling nodes
9 %
10 %   example
11 %     W = remove_dangling(W);
12
13 e = ones(size(W,1),1);
14 e = sparse(e);
15
16 d = W*e;
17 dang = d==0;
18 dang = sparse(dang);
19
20 Dang = dang*e';
21 Dang_t = Dang' - diag(diag(Dang'));
22
23 W = W + Dang + Dang_t;
24 W = sparse(W);

```

---

Figura 25: side/remove\_dangling

## Riferimenti bibliografici

- [1] Gang Wu, Wei Xu, Ying Zhang, Yimin Wei, *A preconditioned conjugate gradient algorithm for GeneRank with application to microarray data mining*, Data Min Knowl Disc 26, 2013, p. 27-56, <https://doi.org/10.1007/s10618-011-0245-7>
- [2] Julie L. Morrison, Rainer Breitling, Desmond J. Higham & David R. Gilbert, *GeneRank: Using search engine technology for the analysis of microarray experiments*, BMC Bioinformatics, 2005
- [3] James W. Demmel, *Applied numerical linear algebra*, SIAM, 1996