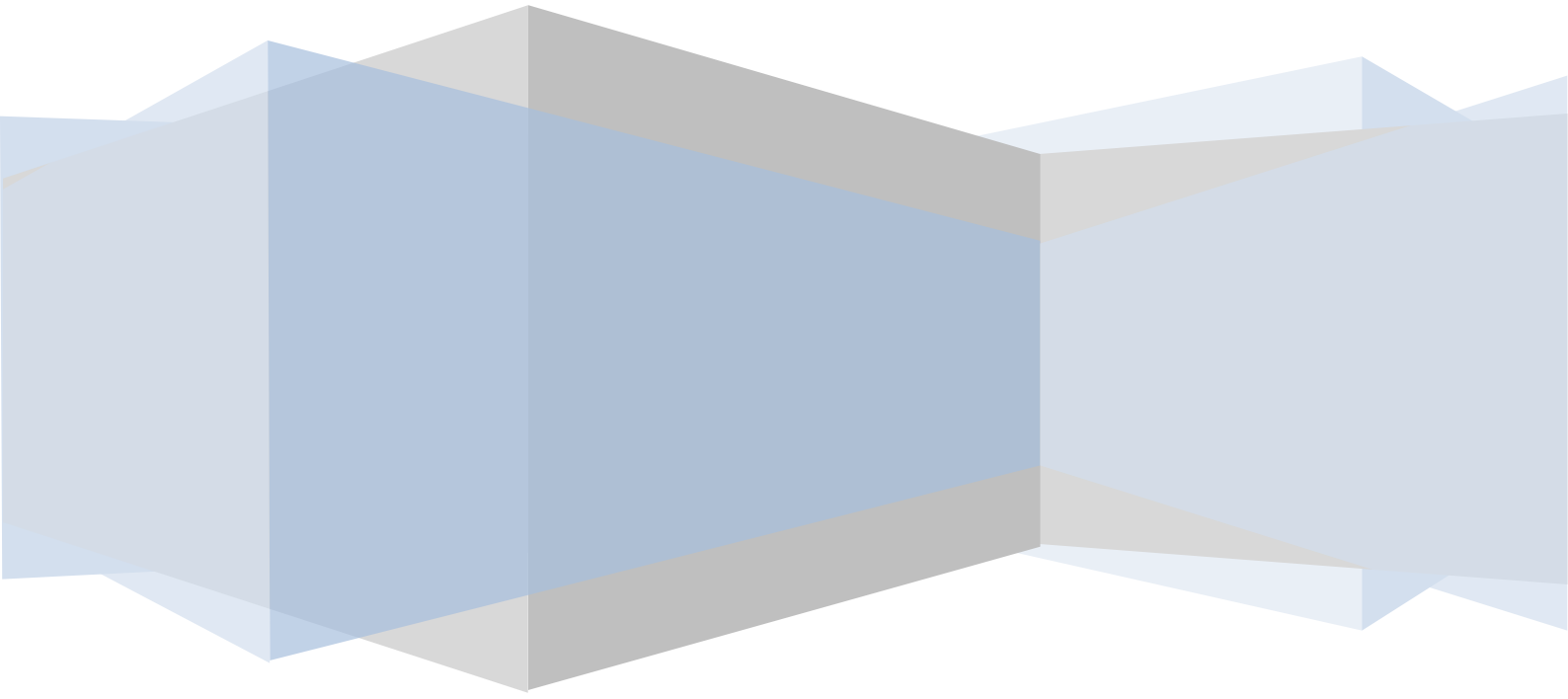
Seminario di Calcolo Scientifico

Confronto di tre diversi algoritmi per la risoluzione di un problema di GeneRank

Giulio Del Corso



Indice

Introduzione:

In questo breve seminario andremo a confrontare tre diversi algoritmi per lo studio del GeneRank.
Il primo algoritmo proposto sfrutta il metodo delle potenze ed è la semplice trasposizione di quanto visto durante il corso svolto.
Il secondo algoritmo invece sfrutta il metodo di Arnoldi andando ad individuare mediante l'algoritmo di ortonormalizzazione di Gram – Schmidt una base ortonormale dello spazio di Krylov.
Il terzo infine sceglierà in maniera distinta il nuovo vettore per iterare il metodo ottenendo grazie ai risultati teorici dimostrati durante il seminario una velocità di convergenza decisamente maggiore.

L'obiettivo sarebbe mostrare la maggiore efficienza dell'algoritmo proposto dall'articolo sia come tempi di esecuzione sia come velocità di convergenza.

A causa della implementazione poco efficiente per matrici di basse dimensioni ($\cong 4000$ geni) è possibile verificare la maggiore velocità di convergenza ma non i tempi più brevi di esecuzione.

Per matrici di dimensioni maggiori invece ($\cong 10^4$) i risultati da me ottenuti nella sperimentazione numerica coincidono con quanto mostrato dall'articolo.

In una seconda implementazione più accurata si riescono ad ottenere dei risultati apprezzabile anche sul fronte della velocità di convergenza, rimane sempre però (per valori bassi del microarray e damping factor inferiore a 0.7) conveniente l'utilizzo del metodo di Arnoldi non modificato.

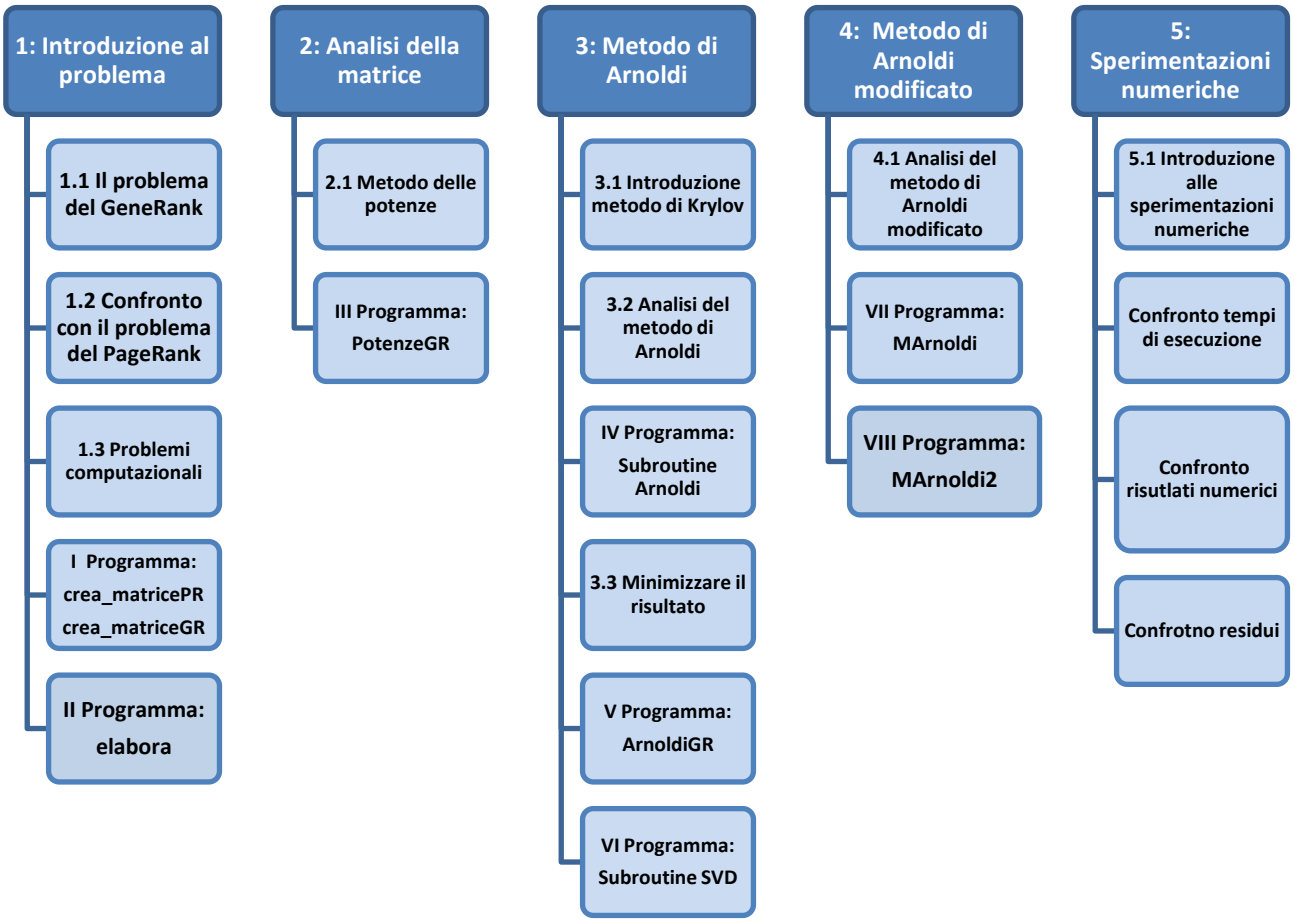
Materiale utilizzato:

Articoli:

Google's Page Rank Model (Page et al., 1998 ; Langville and Meyer, 2005)
GeneRank Model (Morrison et al., 2005; Yue et al., 2007)
Krylov Subspace Algorithms for Computing GeneRank for the Analysis of Microarray Data Mining (Gang Wu, Ying Zhang and Yimin Wei, 2010)

Libri di riferimento:

Iterative Method for Sparse Linear System (Yousef Saad)
Metodi numerici per l'algebra lineare (Bini Dario, Capovani Milvio, Menchi Ornella)


1: Introduzione al problema
1.1 Il problema del GeneRank
1.2 Confronto con il problema del PageRank
1.3 Problemi computazionali
I Programma: crea_matricePR crea_matriceGR
II Programma: elabora
2: Analisi della matrice
2.1 Metodo delle potenze
III Programma: PotenzeGR
3: Metodo di Arnoldi
3.1 Introduzione metodo di Krylov
3.2 Analisi del metodo di Arnoldi
IV Programma: Subroutine Arnoldi
3.3 Minimizzare il risultato
V Programma: ArnoldiGR
VI Programma: Subroutine SVD
4: Metodo di Arnoldi modificato
4.1 Analisi del metodo di Arnoldi modificato
VII Programma: MArnoldi
VIII Programma: MArnoldi2
5: Sperimentazioni numeriche
5.1 Introduzione alle sperimentazioni numeriche
Confronto tempi di esecuzione
Confronto risutlati numerici
Confrotno residui

1.1 Analisi matriciale del problema del GeneRank:

La formulazione del problema è analoga a quella fornita nello studio del PageRank per una rete internet.

Sia $G = \{g_1, \dots, g_N\}$ un insieme di N geni disposti su di un microarray.

La matrice di giacenza associata al problema si definisce come:

$$w_{ij} = \begin{cases} 1 \text{ se } g_i \text{ e } g_j \text{ condividono un}' \text{annotazionesu GO} \\ 0 \text{ altrimenti} \end{cases}$$

Osservazione:
La matrice W è simmetrica.

Valutare quale gene/nodo sia più importante è equivalente a risolvere il sistema lineare:

Esempio:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \rightarrow \begin{cases} Imp_1 = Imp_2 \\ Imp_2 = \frac{1}{2} Imp_1 + Imp_3 \\ Imp_3 = \frac{1}{2} Imp_1 \end{cases}$$

Esempio simmetrico:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \rightarrow \begin{cases} Imp_1 = Imp_2 + Imp_3 \\ Imp_2 = \frac{1}{2} Imp_1 \\ Imp_3 = \frac{1}{2} Imp_1 \end{cases} \rightarrow \begin{cases} Imp_1 = 1 \\ Imp_2 = \frac{1}{2} \\ Imp_2 = \frac{1}{2} \end{cases}$$

Questo può essere riscritto come un problema di autovalori:

Definiamo $\deg(i) = \sum_{j=1}^{N} w_{ij} = \sum_{j=1}^{N} w_{ji}$
Detto **grado** del gene i-esimo.

Poniamo $D = \text{diag}(\deg(1), \dots, \deg(N))$

Detto r^* vettore delle importanze:
$Mr^* = r^*$ con $M = WD^{-1}$

Esempio simmetrico:

$$D^{-1} = \begin{pmatrix} \frac{1}{2} & & \\ & 1 & \\ & & 1 \end{pmatrix} \rightarrow M = \begin{pmatrix} 0 & 1 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{pmatrix}$$

Infatti:

$$\begin{pmatrix} 0 & 1 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

Per evitare problemi se la i-esima riga (e di conseguenza colonna) è nulla la imponiamo uguale ad e^T (la colonna ad e, questo viene fatto nel programma *elaboraGR*) così da ridistribuire l'importanza sull'intero microarray.

Il vettore di GeneRank che stiamo cercando è dunque r^* tale che:
$Mr^* = r^*$

Teorema 1 (Perron – Frobenius):
Sia A una matrice $n \times n$ di elementi non negativi.

Allora esiste un autovalore λ di A tale che $\lambda = \rho(A) \geq 0$.
Esistono un autovettore destro x ed uno sinistro y corrispondenti a λ con componenti non negative.
Se inoltre A è irriducibile allora λ è semplice e gli autovettori x, y hanno componenti positive.
Se infine A ha elementi positivi allora λ è l'unico autovalore di modulo massimo.

Problema:
Esistono grafi riducibili.
Si studia dunque la matrice:

$$A = d \cdot W^t D^{-1} + (1-d) \cdot ex \cdot e^T$$

Con d damping factor e ex un vettore arbitrario tale che $ex^T e = 1$; $ex = \begin{pmatrix} ex_1 \\ ex_2 \\ \vdots \\ ex_N \end{pmatrix}$ con $ex_i \geq 0$

Teorema 2:

Sia $L = Z^H (D^{-1}W) Z$ la decomposizione di Schur di $D^{-1}W$, dove $Z = \left[\frac{e}{\|e\|_2}, Z_\perp\right] \in \mathbb{C}^{n \times n}$ è unitaria e $L \in \mathbb{C}^{n \times n}$ sia triangolare superiore con gli autovalori $\{1, \lambda_2, \dots, \lambda_n\}$ di $D^{-1}W$ lungo la diagonale.
Allora:

$$Z^H A^t Z = d \cdot L + (1-d) \cdot \begin{pmatrix} 1 & \|e\|_2 \cdot ex^t \cdot Z_\perp \\ 0 & 0 \end{pmatrix} \coloneqq \tilde{L}$$

è la decomposizione di Schur di A^t, $\tilde{L}$ è triangolare superiore con elementi diagonali:
$\{1, d \cdot \lambda_2, \dots, d \cdot \lambda_n\}$

Conseguenza:
$Z^H A Z = \tilde{L}^t$ è una matrice triangolare inferiore con elementi diagonali $\{1, d \cdot \lambda_2, \dots, d \cdot \lambda_n\}$.
Inoltre detti $\{1, \lambda_2, \dots, \lambda_n\}$ gli autovalori di $W^t D^{-1}$ allora gli autovalori di A sono proprio $\{1, d \cdot \lambda_2, \dots, d \cdot \lambda_n\}$.
Osserviamo che 1 è l'autovalore di modulo massimo di A poiché $|d \cdot \lambda_i| \leq d < 1$.
Di conseguenza il corrispondente autovettore r^* di A è unico anche se A è riducibile.

Grazie a questa formulazione la soluzione esiste unica e $\rho(A)$ è l'unico autovalore di modulo 1.

Osservazione:
L'inserimento del damping factor serve ad eliminare il problema della matrici cicliche, ci si riconduce infatti al teorema di Perron – Frobenius che, in questo caso, garantisce la convergenza del metodo.

Attenzione:
Il damping factor va a modificare l'autovettore relativo all'autovalore 1. Per un damping factor prossimo ad 1 otteniamo valori vicini ma per d arbitrario non abbiamo garanzie sulla soluzione ottenuta.
Questo inoltre non modifica la scala di importanza dei geni ma restituisce valori numerici differenti, infatti:
$r^* \mid Mr^* = W^t D^{-1} r^* = r^*$ allora:

$$Ar^* = d \cdot M \cdot r^* + (1-d) \cdot e \cdot v^T \cdot r^* = \left(\mathrm{Id} \cdot d + (1-d) \begin{pmatrix} v_1 & \cdots & v_n \\ \vdots & & \vdots \\ v_1 & \cdots & v_n \end{pmatrix} \right) \cdot r^* \neq r^*$$

Osservazione:
Nell'articolo di Morrison et al. (2005) viene evidenziato come il valore di d dipenda dai dati.
La scelta del fattore di smorzamento ottimale rimane però un problema aperto (nelle sperimentazioni numeriche faremo tentativi con d diversi).

La soluzione r^* rappresenta l'importanza dei geni studiati. (Morrison et al., 2005)

1.2 Confronto con il PageRank:

Le differenze sostanziali con il metodo di PageRank riguardano la matrice di giacenza che, nel caso del GeneRank, risulta essere simmetrica.

In due dimostrazioni questo fatto ci è utile (Sfruttando il Teorema Spettrale abbiamo garantita la diagonalizzabilità della matrice) ma dal punto di vista implementativo non ci sono sostanziali differenze in quanto la matrice stocastica A su cui applicheremo i vari metodi non mantiene la struttura simmetrica.

Attenzione:
E' necessario prestare attenzione al fatto che rispetto alla normale formulazione del problema (ad esempio quella proposta dal Prof. Bini) la matrice A con cui stiamo lavorando risulti essere la trasposta.
Si è scelto di mantenere questa differenza in quanto gli algoritmi sono stati implementati in maniera autonoma.

1.3 Problemi computazionali:

Nello scrivere i programmi in Fortran per implementare gli algoritmi descritti si sono evidenziati i seguenti problemi.

Problema 1:
Il link fornito dagli autori dell'articolo contenete la matrice delle giacenze W e il vettore ex delle evidenze sperimentali non è più funzionante.

> **Risoluzione:**
> La matrice delle giacenze è stata ottenuta studiando il sito di riferimento di Gene Ontology Consortium e valutando dagli esempi forniti il livello di sparsità delle matrici di geni (approssimativamente il 99% dei geni non si influenzano). A questo punto ho implementato un programma per generare una matrice (casuale) simmetrica che rispettasse la proprietà del punto precedente.
> Per ottenere *ex* invece (non avendo evidenze sperimentali su cui basarmi) si è deciso di attribuire ai primi $\frac{n}{100}$ geni il valore $\frac{1}{100}$ e di azzerare gli altri.
> Il motivo è che l'array di geni è scelto arbitrariamente dallo sperimentatore, dunque l'ordine, lavorando con matrici casuali, non è rilevante.

Problema 2:
Nel generare le matrici casuali con il comando $rand$ o $rand\_int$ di Fortran la successione generata risultava essere sempre la stessa (non andando a modificare il seme iniziale).

> **Risoluzione:**
> Si è implementata una subroutine che genera il seme casuale della sequenza a partire dall'orario del calcolatore.

Problema 3:
Come memorizzare l'informazione della matrice delle giacenze così da poter utilizzare gli stessi dati per le varie sperimentazioni numeriche.

> **Risoluzione:**
> Ogni singolo programma salva l'informazione sulla matrice (elaborata) in un file di testo.
> Nonostante il processo di lettura vada a rallentare l'algoritmo poiché siamo interessati esclusivamente ad un confronto fra i vari metodi, possiamo ritenerlo accettabile.

Programma I: crea_matricePR

File: /home/giulio/Desktop/Tesina/G...una matrice/crea_matricePR.f90 Page 1 of 1

```
! Programma: crea_matricePR
! Autore: Giulio Del Corso
! Input: Richiede l'inserimento da tastiera della dimensione n della matrice di giacenza.
! Output: Genera un file pgm che contiene la dimensione n della matrice di giacenza e i valori casuali
(0,1) generati
! per i collegamenti del micrarray di geni. Il formato pgm è dato per poter avere un'immagine della
matrice di giacenza.


PROGRAM crea_matricePR
INTEGER :: i , j , n
INTEGER , DIMENSION(:,:) , ALLOCATABLE :: A
REAL :: r , r2


OPEN(UNIT=1,FILE="Matrice.pgm",STATUS="UNKNOWN",ACTION="READWRITE")


WRITE(*,*) "Inserire la dimensione della matrice da creare:"
        READ(*,*) n


! Creazione della matrice casuale
ALLOCATE(A(n,n))
CALL init_random_seed
A=0

DO i=1,n
        DO j=1,n
        CALL random_number(r)
        CALL random_number(r2)
        IF (r2>0.99) then
                A(i,j)=NINT(r)
        END IF
        END DO
END DO


! Trascrizione dei valori della matrice nel file Matrice.pgm
write(1,300)"P2"
write (1,100) n , n , 1

DO i=1,n
        DO j=1,n
        WRITE (1,200) A(i,j)
        END DO
END DO


! Formati usati per la trascrizione.
100 FORMAT(3I7) ! Formato per righe et colonne.
200 FORMAT(I5)  ! Formato per i numeri.
300 FORMAT(A2)  ! Formato per il P2.

END PROGRAM crea_matricePR


SUBROUTINE init_random_seed()
INTEGER :: i, n, clock
INTEGER, DIMENSION(:), ALLOCATABLE :: seed

CALL RANDOM_SEED(size = n)
ALLOCATE(seed(n))

CALL SYSTEM_CLOCK(COUNT=clock)

seed = clock + 37 * (/ (i - 1, i = 1, n) /)

CALL RANDOM_SEED(PUT=seed)

DEALLOCATE(seed)
END SUBROUTINE
```

Programma I bis: crea_matriceGR

File: /home/giulio/Desktop/Tesina/G...una matrice/crea_matriceGR.f90 Page 1 of 1

```
! Programma: crea_matriceGR
! Autore: Giulio Del Corso
! Input: Richiede l'inserimento da tastiera della dimensione n della matrice di giacenza.
! Output: Genera un file pgm che contiene la dimensione n della matrice di giacenza e i valori casuali
(0,1) generati
! per i collegamenti del micrarray di geni. Il formato pgm è dato per poter avere un'immagine della
matrice di giacenza.
! ATTENZIONE: A differenza di prima la matrice generata casualmente è simmetrica.


PROGRAM crea_matriceGR
INTEGER :: i , j , n
INTEGER , DIMENSION(:,:) , ALLOCATABLE :: A
REAL :: r , r2


OPEN(UNIT=1,FILE="Matrice.pgm",STATUS="UNKNOWN",ACTION="READWRITE")


WRITE(*,*) "Inserire la dimensione della matrice da creare:"
        READ(*,*) n


! Creazione della matrice casuale
ALLOCATE(A(n,n))
A=0
CALL init_random_seed

DO i=1,n
        DO j=i,n
        CALL random_number(r)
        CALL random_number(r2)
        IF (r2>0.99) then
                A(i,j)=NINT(r)
        END IF
        END DO
END DO

DO i=1,n
        DO j=1,i
        A(i,j)=A(j,i)
        END DO
END DO


! Trascrizione dei valori della matrice nel file Matrice.pgm
write(1,300)"P2"
write (1,100) n , n , 1

DO i=1,n
        DO j=1,n
        WRITE (1,200) A(i,j)
        END DO
END DO


! Formati usati per la trascrizione.
100 FORMAT(3I7) ! Formato per righe et colonne.
200 FORMAT(I5)  ! Formato per i numeri.
300 FORMAT(A2)  ! Formato per il P2.

END PROGRAM crea_matriceGR


SUBROUTINE init_random_seed()
...
END SUBROUTINE
```

Programma II: elabora

File: /home/giulio/Desktop/Tesina/E...orare la matrice A/elabora.f90 Page 1 of 2

```
! Programma: elabora
! Autore: Giulio Del Corso
! Input: Richiede la presenza di un file pgm nella cartella di esecuzione.
! Studiando il GeneRank la matrice deve essere simmetrica.
! Output: Genera un file pgm che contiene la matrice stocastica associata al problema.
! ATTENZIONE: Il programma utilizza un damping factor fissato e genera in maniera propria il vettore
di inizio
! (Nel vero problema del GeneRank il vettore contiene alcune evidenze sperimentali).
! ATTENZIONE: Il problema calcola A trasposta (Quella presente nel PR)
! La matrice conclusiva è STOCASTICA PER RIGA.


PROGRAM elabora
INTEGER , PARAMETER :: dp=KIND(0.d0)
INTEGER :: i , j , n
REAL(dp) , DIMENSION(:,:) , ALLOCATABLE :: A , B
REAL(dp) :: d   ! Damping factor
REAL(dp) , DIMENSION(:) , ALLOCATABLE :: Diag , ex



OPEN(UNIT=1,FILE="Matrice.pgm",STATUS="UNKNOWN",ACTION="READWRITE")
OPEN(UNIT=2,FILE="A.txt",STATUS="UNKNOWN",ACTION="READWRITE")



! Lettura della matrice da file
READ(1,*)
READ(1,*) n
ALLOCATE(A(n,n), ex(n))

DO i=1,n
        DO j=1,n
        read(1,*) A(i,j)
        END DO
END DO


! Scegliamo il damping factor:
WRITE(*,*) "Inserire il damping factor scelto:"
        READ(*,*) d


! Sostituiamo le righe di tutti 0 e di tutti 1. ATTENZIONE, l'ho resa non simmetrica.
ALLOCATE(B(n,n))
B=0

DO i=1,n
        IF (sum(INT(A(i,:)))==0) THEN
        B(i,:)=1
        END IF
END DO

A=A+B


! Nella seconda fase costruiamo la matrice diagonale delle somme delle righe.
ALLOCATE(Diag(n))
Diag=0

DO i=1,n
        Diag(i)=sum(A(i,:))
END DO


! Invertiamo la diagonale
Diag=1/Diag



! Generiamo un vettore di norma unitaria ex (Osservazione: Non varia seed)
CALL random_number(ex)
```

```
ex=ex/sum(ex)


! Calcoliamo la matrice dD^-1W+(1-d)e*ex^T
DO i=1,n
        DO j=1,n
        A(i,j)=A(i,j)*Diag(i)*d+(1-d)*ex(j)
        END DO
END DO


! Trascrizione dei valori della matrice nel file A.txt

write (2,100) n , n
DO i=1,n
        DO j=1,n
        WRITE (2,200) A(i,j)
        END DO
END DO


! Formati usati per la trascrizione.
100 FORMAT(3I7)         ! Formato per righe et colonne.
200 FORMAT(F28.26)      ! Formato per i numeri.

END PROGRAM elabora
```

2.1 Il metodo delle potenze:

Teorema (Metodo delle potenze):
Per la ricerca della soluzione del problema del GeneRank possiamo applicare il metodo delle potenze.

Dimostrazione:
Per il teorema di Perron – Frobenius poiché A ha solo elementi positivi allora 1 è l'unico autovalore di modulo massimo ed è semplice. La forma di Jordan è dunque:

$$S^{-1}AS = J = \begin{pmatrix} 1 & 0 \\ 0 & \hat{J} \end{pmatrix} \text{ con } \rho(\hat{J}) < 1$$

Osserviamo che vale $Je_1 = e_1 \rightarrow e_1^T J = e_1^T \rightarrow ASe_1 = Se_1 \rightarrow e_1^T S^{-1} A = e_1^T S^{-1}$
Allora $Se_1 = e$ è un autovettore destro mentre $e_1^T S^{-1}$ è un autovettore sinistro.
Vale $J^K = \begin{pmatrix} 1 & 0 \\ 0 & \hat{J}^K \end{pmatrix} \rightarrow \lim \rho(\hat{J}^K) = 0 \rightarrow \lim \hat{J}^K = e_1 e_1^T$ da cui:

$$\lim A^K = Se_1 e_1^T S^{-1} = ew^T$$

Con convergenza tanto più veloce quanto è minore $\rho(\hat{J})$.

Teorema:
Se il vettore iniziale è scelto con norma 1 pari ad 1 allora i vettori successivi non devono essere normalizzati.

Dimostrazione:
Se $x \mid \|x\|_1 = 1 \rightarrow$ per $y = Ax$ vale $\|y\|_1 = 1$.
Infatti $\|y\|_1 = y^T e = x^T Ae = x^T e = 1$.

Programma III: PotenzeGR


```
! Programma: PotenzeGR
! Autore: Giulio Del Corso
! Input: Richiede la presenza del file .txt contenente la matrice già elaborata.
! Output: Genera un file txt contenente il vettore di GeneRank. Pubblica il  valore in tempo richiesto
per l'iterazione.
! ATTENZIONE: Sto sfruttando la notazione PR.
! ATTENZIONE: Non sfrutta la sparsità della matrice.


PROGRAM PotenzeGR
INTEGER , PARAMETER :: dp=KIND(0.d0)
INTEGER :: i , j , n , tmp
INTEGER , PARAMETER :: maxit=300
REAL(dp) , DIMENSION(:,:) , ALLOCATABLE :: A
REAL(dp) , DIMENSION(:) , ALLOCATABLE :: GR , GR2
REAL(dp) :: start, finish, p=1


OPEN(UNIT=1,FILE="A.txt",STATUS="UNKNOWN",ACTION="READWRITE")
OPEN(UNIT=2,FILE="GR.txt",STATUS="UNKNOWN",ACTION="READWRITE")


! Lettura della matrice da file
READ(1,*) n
ALLOCATE(A(n,n),GR(n),GR2(n))

DO i=1,n
        DO j=1,n
        read(1,*) A(i,j)
        END DO
END DO


! Inizio calcolo tempo.
CALL CPU_TIME(start)


! Inizializziamo il vettore (In maniera casuale).
CALL RANDOM_NUMBER(GR)
GR=GR/SUM(GR)


! Esecuzione. Attenzione: r=rA
tmp=0 ; p=1

DO WHILE ((tmp<maxit).and.(p>1E-14))
        GR2=0
        GR2=Matmul(GR,A)
        p=sum(abs(GR2-GR))      ! Condizione sulla norma1

        write(*,*) p
        GR=GR2
END DO


! Interrompiamo il calcolo del tempo.
CALL CPU_TIME(finish)
        write(*,*) "Time:" , finish-start


! Trascrizione dei valori della matrice nel file GR.txt
write (2,100) n
DO i=1,n
        WRITE (2,200) GR(i)
END DO


! Formati usati per la trascrizione.
100 FORMAT(3I7)         ! Formato per righe et colonne.
200 FORMAT(F30.28)      ! Formato per i numeri.

END PROGRAM PotenzeGR
```

3.1 Il metodo di Arnoldi (Krylov):

Idea:
Stiamo cercando di risolvere un sistema lineare della forma:

$Ax = b$; $A \in M_n(\mathbb{R})$

Definizione (Risoluzione per proiezione):
Fissati due spazi K_m ed L_m di dimensione m e una stima iniziale della soluzione x_0 allora si prende come stima della soluzione lineare del sistema il valore x_m tale che:

$x_m \in x_0 + K_m$
$b - Ax_m \perp L_m$ *(Condizione di Petrov – Galerkin)*

Costruzione mediante basi:
Siccome la soluzione che stiamo cercando è della forma:

$x_m = x_0 + y_m$ con $y_m \in K_m$

Presa $V_m = (v_1, \dots, v_m)$ una base di K_m e $W_m = (w_1, \dots, w_m)$ una base di L_m allora è possibile esprimere:

$y_m = \sum_{i=1}^{m} z_i v_i = V_m z$

Il residuo m-esimo verrà definito dunque come:

$r_m = b - Ax_m = r_0 - Ay_m$ (Per la condizione di Petrov – Galerkin sarà ortogonale ad L_m)

Ricavare la soluzione approssimata:
Scrivendo in maniera esplicita l'ortogonalità:

$\langle w_i, r_0 - Ay_m \rangle = 0$

Sostituendo y_m otteniamo:

$W_m^T(r_0 - AV_m z) = 0$

Supponendo $W_m^T AV_m$ invertibile allora il vettore può essere ricavato come:

$z = (W_m^T AV_m)^{-1} W_m^T r_0$

La soluzione approssimata è dunque:

$$x_m = x_0 + V_m (W_m^T AV_m)^{-1} W_m^T r_0$$

Definizione (Metodo di Krylov):
Il metodo di Krylov è una particolare risoluzione per proiezione per la quale, una volta fissata una stima iniziale x_0 si definisce il residuo come $r_0 \coloneqq b - Ax_0$ e si sceglie K_m come il sottospazio di Krylov $K_m(A, r_0)$.

Definizione (Sottospazio di Krylov):
$K_m(A, r_0) = \mathrm{Span}(r_0, Ar_0, A^2 r_0, \dots, A^{m-1} r_0)$

Osservazione:
A seconda della scelta di L_m varia il metodo, due possibilità sono:
$L_m = K_m$ Metodo di Arnoldi
$L_m = AK_m$ Metodo GMRES

Osservazione approssimazione ottenuta:
$A^{-1}b \approx x_m = x_0 + q_{m-1}(A)r_0$ con q_{m-1} un polinomio di grado al più $m-1$

Convergenza per metodi di proiezione:
Ogni metodo di proiezione è convergente in quanto minimizza il residuo su uno spazio di dimensione crescente.

$\|r_k\|_2$ è una successione decrescente
$\|r_n\|_2 = 0$

Caso specifico metodi di Krylov:
Per osservazioni sugli spazi di Krylov l'approssimazione sarà della forma:

$p_k(A)x_0$ con $\mathrm{grad}(p_k) \leq k$

Osservazione:
Supponiamo A diagonalizzabile (per rendere più facile la trattazione).

Sia $A = VDV^{-1}$ allora:

$\|r_k\|_2 = \min_{p_k} \|Vp_k(D)V^{-1}\|_2 \|r_0\|_2 \leq K(V) \min_{p_k} \|p_k(D)\|_2 \|r_0\|_2$

Quindi:

$\frac{\|r_k\|_2}{\|r_0\|_2} \leq K(V) \min_{p_k} \max_{1 \leq i \leq m} |p_k(\lambda_i)|$

Osservazione:
Per il teorema di Hamilton – Cayley il residuo è nullo per $k = m$.

Osservazione problemi computazionali:
La convergenza dipende dal condizionamento di V dunque anche se in teoria abbiamo sempre garantito la convergenza del metodo può accadere che m sia eccessivamente grande per V mal condizionato.
Si può precondizionare il problema ma questo esula dallo scopo di questo approfondimento.

3.2 Metodo di Arnoldi per il problema del GeneRank:

Dato un vettore iniziale v_1 di norma unitaria il metodo di Arnoldi in m passi genera in successione (Se stiamo lavorando in aritmetica esatta) una base ortonormale $V_{m+1} = (v_1, \dots., v_{m+1})$ del sottospazio di Krylov $K_{m+1}(A, v_1) = \text{Span}\{v_1, Av_1, \dots, A^m v_1\}$.

Osservazione:
Nel sottospazio $K_m(A, v)$ la restrizione di A è rappresentabile come una matrice di Hessenberg $H_m = (h_{i,j})$ quadrata $m \times m$.
Vale inoltre la seguente relazione (Bai et al., 2000; Saad, 2003):

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T = V_{m+1} \widetilde{H}_m \quad \textbf{(13)}$$

Con $e_m = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1_m \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

$\widetilde{H}_m \in M_{(m+1)\times m}$ matrice di Hessenberg superiore.

$\widetilde{H}_m = \begin{pmatrix} & & H_m & \\ 0 & \cdots & 0 & h_{m+1,m} \end{pmatrix}$

Osservazione:
L'algoritmo proposto è semplicemente quello di ortonormalizzazione di Gram – Schmidt.
Può essere infatti scritto in una forma più intuitiva come:

Algoritmo:
Dato un vettore iniziale v_1 di norma unitaria.
***For** j=1,…,m*
Calcoliamo $h_{i,j} = \langle Av_j, v_i \rangle = v_j^T A^t v_i$
$w_{i,j} \coloneqq Av_j - \sum_{i=1}^{j} h_{i,j} v_i$ *(Eliminiamo le componenti lungo i vettori precedenti).*
$h_{j+1,j} = \|w_{i,j}\|_2$
Viene 0 se il vettore appartiene allo spazio invariante descritto fino a questo momento).

If** $h_{j+1,j} = 0$ **then
Interrompere l'algoritmo
Else
$v_{j+1} = \frac{w_j}{h_{j+1,j}}$ *(Ortonormaliziamo)*
End if
End for

Programma IV: Arnoldi


```
! SUBROUTINE: Arnoldi
! Autore: Giulio Del Corso
! Input: Vm contiene i vettori calcolati iterativamente, il numero di passi che il metodo deve fare è
indicato nella subroutine.
! Secondo le specifiche dell'articolo imponiamo m=3.
! H è la matrice di Hessenberg che rappresenta la scrittura in base Vm della matrice A.
! p è il residuo.
! m è il numero di iterazioni.
! n è la dimensione della matrice A.


SUBROUTINE Arnoldi(A,Vm,H,m,n)
IMPLICIT NONE
INTEGER , PARAMETER :: dp=KIND(0.d0)
REAL(dp) :: norm
INTEGER :: m , n , i , j , k
REAL(dp) , DIMENSION(n,m+1) :: Vm
REAL(dp) , DIMENSION(m+1,m) :: H
REAL(dp) , DIMENSION(n,n) :: A
REAL(dp) , DIMENSION(n) :: q


DO j=1,m
        ! q=Av_j
        q=Matmul(A,Vm(:,j))

        ! Ortogonalizzazione rispetto ai vettori precedenti.
        DO i=1,j
                ! h_ij=v_i^Tq
                H(i,j)=0
                DO k=1,n
                        H(i,j)=H(i,j)+Vm(k,i)*q(k)
                END DO

                ! q=q-h_ijv_i
                q=q-H(i,j)*Vm(:,i)
        END DO

        ! h_i+1,j=norma2(q) ; Calcolo della norma 2.
        norm=0
        DO k=1,n
                norm=norm+q(k)**2
        END DO
        norm=sqrt(norm)

        H(j+1,j)=norm


        ! Condizione di interruzione.
        IF (abs(H(j+1,j))<1E-15) THEN
                Vm(:,m+1)=Vm(:,j)       ! Con questa condizione garantisco di avere nell'ultimo
vettore quello che reiteriamo.
                return
        ELSE
                ! v_j+1=q/h_j+1
                Vm(:,j+1)=q/H(j+1,j)
        END IF


END DO

END SUBROUTINE Arnoldi
```

3.3 Minimizzare il risultato:

Sfruttare la struttura del problema di GeneRank:
La matrice del GeneRank data la sua struttura ha massimo autovalore 1.

Sfruttando questo fatto possiamo trasformarlo in un problema di minimizzazione, cerchiamo dunque un $x^G \in K_m(A, v)$ tale che:

$$\|(A - \mathrm{Id})x^G\|_2 = \min_{\substack{u \in K_m(A,v) \\ \|u\|_2 = 1}} \|(A - \mathrm{Id})u\|_2 \quad \textbf{(14)}$$

Come risolvere l'equazione efficacemente:
Il problema di minimizzazione può essere risolto grazie ad una singola decomposizione a valori singolari al costo di $O(m^3)$

Denotiamo $[\mathrm{Id}_m, 0] \in M_{m+1,m} = \begin{pmatrix} \mathrm{Id}_m \\ 0 \quad \cdots \quad 0 \end{pmatrix}$

Dalla ricorsione (13):
$\|(A - \mathrm{Id})x^G\|_2 = \min\limits_{\substack{u \in K_m(A,v) \\ \|u\|_2 = 1}} \|(A - \mathrm{Id})u\|_2 = \min\limits_{\substack{z \in \mathbb{C}^m \\ \|z\|_2 = 1}} \|(A - \mathrm{Id})V_m z\|_2 = \min\limits_{\substack{z \in \mathbb{C}^m \\ \|z\|_2 = 1}} \left\|V_{m+1}\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right)z\right\|_2 =$
$= \min\limits_{\substack{z \in \mathbb{C}^m \\ \|z\|_2 = 1}} \left\|\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right)\mathrm{z}\right\|_2$ In quanto V_{m+1} è unitaria. **(15)**

Applichiamo la SVD:
$\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right) = UTS^T$

Allora dalla (15):

$$x^G = V_m z^G$$

Dove $z^G = S(:, m)$ è il vettore corrispondente al valore minimo $\sigma_{min}\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right)$

$r^G = (A - \mathrm{Id})x^G = \sigma_{min}\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right) \cdot V_{m+1}U(:, m)$ il residuo.

Vale:

$$\|r^G\|_2 = \sigma_{min}\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right)$$

$U(:, m)$ è la m-esima colonna di U (Questo è il singolo vettore sinistro associato a $\sigma_{min}\left(\widetilde{H}_m - [\mathrm{Id}_m, 0]\right)$)

Osservazione su problemi computazionali:
Calcolare il metodo di Arnoldi per m elevato presenta delle difficoltà di implementazione a causa della grande quantità di memoria da dedicare al processo (oltre che dal costo computazionale spesso eccessivo). Un sistema per risolvere il problema può essere applicare il restart ossia, dopo aver individuato una buona approssimazione del vettore x^G, applichiamo nuovamente il metodo di Arnoldi partendo questa volta dal vettore prima approssimato.

Valutare il residuo:
Per ridurre il costo potremmo usare la norma 2 del residuo: $\|r^G\|_2$
Conviene però valutare la norma 1 del residuo direttamente sfruttando la relazione:

$$\rho = \frac{\|Ax^G - x^G\|_1}{\|x^G\|_1}$$

Così formulata dovremmo considerare un prodotto matrice vettore, questo però può essere realizzato implicitamente sfruttando la (13) da cui segue la relazione conclusiva:

$$\rho = \frac{\|Ax^G - x^G\|_1}{\|x^G\|_1} = \frac{\|V_{m+1}[\tilde{H}_m S(:,m)] - V_m S(:,m)\|_1}{\|V_m S(:,m)\|_1}$$

Con costo computazionale $O(N)$

Programma V: ArnoldiGR


```
! Programma: Arnoldi
! Autore: Giulio Del Corso
! Input: Richiede la presenza di un file txt nella cartella di esecuzione.
! Output: Genera un file pgm che contiene la matrice stocastica associata al problema.
! Il programma nel mentre genere una base ortonormale V_(m+1)

! Il vettore iniziale viene scelto in maniera arbitraria.
! p è il residuo e tol è l'indicatore di tolleranza.


PROGRAM ArnoldiGR
IMPLICIT NONE
INTEGER , PARAMETER :: dp=KIND(0.d0)
INTEGER :: i , j , n , k
INTEGER , PARAMETER :: m=3 , maxit=500 ! Numero di iterazioni del metodo di Arnoldi.
REAL(dp) :: p=1 , start, finish
REAL(dp) , DIMENSION(:,:) , ALLOCATABLE :: A , Vm
REAL(dp) , DIMENSION(:) , ALLOCATABLE :: v
REAL(dp) :: d(m) , H(m+1,m) , S(m,m) , U(m+1,m+1)       ! H è direttamente H segnato, ossia ha
l'ultima colonna di 0.


! Ricavare i dati dalla matrice:
OPEN(UNIT=1,FILE="A.txt",STATUS="UNKNOWN",ACTION="READWRITE")
OPEN(UNIT=2,FILE="GRArnoldi.txt",STATUS="UNKNOWN",ACTION="READWRITE")

READ(1,*) n
ALLOCATE(A(n,n),v(n),Vm(n,m+1))


! Trasponiamo la matrice per ricondurci al caso noto:
DO i=1,n
        DO j=1,n
        read(1,*) A(i,j)
        END DO
END DO

A=transpose(A)


! Inizio calcolo tempo.
CALL CPU_TIME(start)


! Inizializziamo un vettore di norma unitaria:
CALL random_number(v)
v=v/sum(v)
Vm(:,1)=v


H=0 ; k=0


! Arnoldi m-step
DO WHILE ((p>1E-14).and.(k<maxit))
        ! Calcoliamo una base ortonormale Vm con il metodo di Arnoldi e una matrice in forma di
Hessenberg H.
        CALL Arnoldi(A,Vm,H,m,n)
                ! OSSERVAZIONE: A prescindere dal fatto che si interrompa prima il vettore conclusivo
è in Vm(:,m+1)

        ! Calcoliamo la SVD di H segnato.
        DO i=1,m
                H(i,i)=H(i,i)-1
        END DO

        CALL calcola_svd(m+1,m,H,d,U,S) ! Attenzione, è la trasposta.

        ! Trasponiamo:
        S=transpose(S)

        ! Scegliamo il nuovo vettore iniziale. Calcoliamolo come Vm*S(:,m)
```

```
        v=0
        DO i=1,n
        DO j=1,m
        v(i)=v(i)+Vm(i,j)*S(j,m)
        END DO
        END DO

        Vm(:,1)=v

        ! Calcoliamo la norma residua (Non ottimizzato)
        p=sum(abs(Matmul(A,v)-v))/sum(abs(v))

        write(*,*) p
        ! Aumentiamo il contatore di controllo.
        k=k+1
END DO


CALL CPU_TIME(finish)

write(*,*) "Time:" , finish-start


! Trascrizione dei valori della matrice nel file GRArnoldi.txt
v=v/sum(v)
write (2,100) n
DO i=1,n
        WRITE (2,200) v(i)
END DO


! Formati usati per la trascrizione.
100 FORMAT(3I7)         ! Formato per righe et colonne.
200 FORMAT(F30.28)      ! Formato per i numeri.

END PROGRAM ArnoldiGR
```

Programma VI: Calcola_svd


```
! Subroutine Lapack per il calcolo della SVD


SUBROUTINE calcola_svd(M,N,A,D,U,VT)
  IMPLICIT NONE
  INTEGER, PARAMETER :: dp=KIND(0.d0)
  INTEGER :: m, n, ldu, ldvt, info, lwork, lda
  REAL(dp)                       :: a(m,n), u(m,m), vt(n,n)
  REAL(dp), DIMENSION(MIN(m,n)) :: d
  REAL(dp), DIMENSION(3*(m+n))  :: work
  CHARACTER :: jobu, jobvt
  jobu='S'
  jobvt='s'
  ldu=m
  ldvt=n
  lda=m
  lwork=3*(m+n)
  CALL dgesvd( jobu, &  ! U contiene i vettori singolari sinistri
         jobvt,     &  ! V contiene i vettori singolari destri
         m, n,      &  ! Dimensioni della matrice di lavoro
         a,         &  ! Matrice di lavoro
         lda,       &
         d,         &  ! Il vettore dei valori singolari
         u,         &  !la matrice dei v.s. sinistri
         ldu,       &
         vt,        &! la matrice dei v.s. destri
         ldvt, work, lwork, info )
END SUBROUTINE
```

4.1 Algoritmo modificato per il GeneRank:

Detta x^G l'approssimazione ottenuta iterando il metodo di Arnoldi, ricordando che questa appartiene al sottospazio di Krylov $K_m(A, v_1) = \mathrm{Span}(v_1, v_2, \dots, v_m)$ ottenuto come base ortonormale dall'iterazione ad m passi del metodo di Arnoldi.

La strategia proposta negli articoli di Ja e Elsner (2000) e Wu (2007) è di cercare un vettore della forma:

$x^G + \hat{\beta} v_{m+1} \in K_{m+1}(A, v_1)$ che soddisfi la seguente proprietà:

$$\left\|(A - \mathrm{Id})\left(x^G + \hat{\beta} v_{m+1}\right)\right\|_2 = \min_{\beta \in \mathbb{C}} \|(A - \mathrm{Id})(x^G + \beta v_{m+1})\|_2 = \min_{\beta \in \mathbb{C}} \|r^G + \beta(A v_{m+1} - v_{m+1})\|_2 \quad \textbf{(19)}$$

Detto $\hat{\beta}$ il valore che minimizza (19) allora l'algoritmo modificato di Arnoldi utilizza come approssimazione del vettore di GeneRank r^* il seguente:

$$x^M = \frac{x^G + \hat{\beta} v_{m+1}}{\left\|x^G + \hat{\beta} v_{m+1}\right\|_2}$$

Osservazione:
Il Teorema 3 serve a fornire un metodo pratico per il calcolo di $\hat{\beta}$

Teorema 3:

Sia $w_{m+1} = (A - \mathrm{Id}) v_{m+1} = A v_{m+1} - v_{m+1}$, allora:

$$\hat{\beta} = -\frac{w_{m+1}^T r^G}{\|w_{m+1}\|_2^2} \quad \textbf{(20)}$$

E la nuova approssimazione x^M soddisfa:

$$\begin{cases} x^M \in \mathrm{Span}(x^G, v_{m+1}) \\ (A - \mathrm{Id}) x^M \perp \mathrm{Span}(w_{m+1}) \end{cases} \quad \textbf{(21)}$$

Dimostrazione:
È un risultato noto (Golub and Van Loan, 1996) che $\hat{\beta}$ soddisfa:
$(w_{m+1}^T w_{m+1})\hat{\beta} = -w_{m+1}^T r^G$
Da cui segue: $\hat{\beta} = -\frac{w_{m+1}^T r^G}{\|w_{m+1}\|_2^2}$
Ricordando che $(A - \mathrm{Id}) x^M = r^G + \hat{\beta} w_{m+1}$ da $\hat{\beta} = -\frac{w_{m+1}^T r^G}{\|w_{m+1}\|_2^2}$ otteniamo il risultato.

Teorema 4:

Denotando $r^M = (A - \mathrm{Id})x^M$ il residuo rispetto ad x^M allora vale:

$$\|r^M\|_2 = \frac{\|r^G\|_2 \sin \angle(r^G, w_{m+1})}{\sqrt{1 + \frac{\|r^G\|_2}{\|w_{m+1}\|_2^2} \cos^2 \angle(r^G, w_{m+1})}} \qquad \textbf{(22)}$$

E:

$$\frac{\|r^M - r^G\|_2}{\|r^G\|_2} = \cos \angle(r^G, w_{m+1}) \quad \textbf{(23)}$$

Dimostrazione:

Abbiamo osservato che:

$$\|r^M\|_2 = \frac{\|r^G + \hat{\beta} w_{m+1}\|_2}{\|x^G + \hat{\beta} w_{m+1}\|_2}$$

Per il numeratore al quadrato vale:

$$\left\|r^G + \hat{\beta} w_{m+1}\right\|_2^2 = \|r^G\|_2^2 + 2\hat{\beta}(r^G, w_{m+1}) + \hat{\beta}^2 \|w_{m+1}\|_2^2 =$$
$$= \|r^G\|_2^2 - \hat{\beta}^2 \|w_{m+1}\|_2^2 = \sigma_{\min}^2 \left(\widehat{H}_m - [\mathrm{Id}, \mathbf{0}]\right) - \hat{\beta}^2 \|w_{m+1}\|_2^2$$

Abbiamo usato la (20).

Per il denominatore al quadrato vale:

$$\left\|x^G + \hat{\beta} w_{m+1}\right\|_2^2 = \|x^G\|_2^2 + 2\hat{\beta}(x^G, v_{m+1}) + \hat{\beta}^2 \|v_{m+1}\|_2^2 = 1 + \hat{\beta}^2$$

Abbiamo usato:

$\|x^G\|_2 = 1$

$\|v_{m+1}\|_2 = 1$

$v_{m+1}^T x^G = 0$

Per la (22) otteniamo:

$$\|r^M\|_2^2 = \frac{\|r^G\|_2^2 - \frac{\left|w_{m+1}^T r^G\right|^2}{\|w_{m+1}\|_2^2}}{1 + \frac{\left|w_{m+1}^T r^G\right|^2}{\|w_{m+1}\|_2^4}} = \frac{\|r^G\|_2^2 - \frac{\left|w_{m+1}^T r^G\right|^2}{\|w_{m+1}\|_2^2 \cdot \|r^G\|_2^2} \cdot \|r^G\|_2^2}{1 + \frac{\left|w_{m+1}^T r^G\right|^2}{\|w_{m+1}\|_2^2 \cdot \|r^G\|_2^2} \cdot \frac{\|r^G\|_2^2}{\|w_{m+1}\|_2^2}} = \frac{\|r^G\|_2 \sin \angle(r^G, w_{m+1})}{\sqrt{1 + \frac{\|r^G\|_2}{\|w_{m+1}\|_2^2} \cos^2 \angle(r^G, w_{m+1})}}$$

Per la (23) otteniamo:

$$\frac{\|r^M - r^G\|_2}{\|r^G\|_2} = \frac{|\hat{\beta}| \cdot \|w_{m+1}\|_2}{\|r^G\|_2} = \frac{|w_{m+1}^T \cdot r^G|}{\|w_{m+1}\|_2 \cdot \|r^G\|_2} = \cos\left(\angle(r^G, w_{m+1})\right)$$

Osservazione:

Il teorema 4 afferma che fino a che $\sin \angle(r^G, w_{m+1}) < 1 \rightarrow \|r^M\|_2 < \|r^G\|_2$

Inoltre minore è la distanza fra r^G e w_{m+1} e più grande sarà la differenza fra r^M ed r^G, ossia migliore sarà l'approssimazione x^M.

Residuo:

Per valutare il residuo in norma 1 nell'algoritmo di Arnoldi modificato usiamo:

$$\frac{\|Ax^M - x^M\|}{\|x^M\|_1} = \frac{\|A[V_m S(:,m) + \widehat{\beta} v_{m+1}] - [V_m S(:,m) + \widehat{\beta} v_{m+1}]\|_1}{\|V_m S(:,m) + \widehat{\beta} v_{m+1}\|_1} = \frac{\|[V_{m+1} \tilde{H}_m S(:,m) + \widehat{\beta} A v_{m+1}] - [V_m S(:,m) + \widehat{\beta} v_{m+1}]\|_1}{\|V_m S(:,m) + \widehat{\beta} v_{m+1}\|_1}$$

Costo computazionale dei due algoritmi

Algoritmo:	Arnoldi $(m+1)$	Arnoldi – Modificato (m)
Prodotti matrice vettore	$m+1$	$m+1$
Spazio occupato in memoria	$m+2$	$m+2$
Prodotti interni	$\frac{m^2+3m+2}{2}$	$\frac{m^2+m+2}{2}$

Programma VII: MArnoldi


```
! Programma: MArnoldi
! Autore: Giulio Del Corso
! Input: Richiede la presenza di un file txt nella cartella di esecuzione.
! Output: Genera un file pgm che contiene la matrice stocastica associata al problema.
! Il programma nel mentre genere una base ortonormale V_(m+1)

! Il vettore iniziale viene scelto in maniera arbitraria.
! p è il residuo e tol è l'indicatore di tolleranza.


PROGRAM MArnoldiGR
IMPLICIT NONE
INTEGER , PARAMETER :: dp=KIND(0.d0)
INTEGER :: i , j , n , k
INTEGER , PARAMETER :: m=3 , maxit=500 ! Numero di iterazioni del metodo di Arnoldi.
REAL(dp) :: p=1 , start, finish , norm , beta
REAL(dp) , DIMENSION(:,:) , ALLOCATABLE :: A , B , Vm
REAL(dp) , DIMENSION(:) , ALLOCATABLE :: v , w , r , xg
REAL(dp) :: d(m) , H(m+1,m) , S(m,m) , ST(m,m) , U(m+1,m+1), temp(m+1,m+1)
        ! H è direttamente H segnato, ossia ha l'ultima colonna di 0.


! Ricavare i dati dalla matrice:
OPEN(UNIT=1,FILE="A.txt",STATUS="UNKNOWN",ACTION="READWRITE")
OPEN(UNIT=2,FILE="MGRArnoldi.txt",STATUS="UNKNOWN",ACTION="READWRITE")

READ(1,*) n
ALLOCATE(A(n,n),v(n),xg(n),r(n),w(n),Vm(n,m+1),B(n,n))


! Trasponiamo la matrice per ricondurci al caso noto:
DO i=1,n
        DO j=1,n
        read(1,*) A(i,j)
        END DO
END DO


DO i=1,n
        DO j=1,n
        B(j,i)=A(i,j)
        END DO
END DO

! Trascriviamo.
A=B


! Inizio calcolo tempo.
CALL CPU_TIME(start)


! Inizializziamo un vettore di norma unitaria:
CALL random_number(v)
v=v/sum(v)
Vm(:,1)=v


H=0
k=0

! Arnoldi m-step
DO WHILE ((p>1E-14).and.(k<100))
        ! Calcoliamo una base ortonormale Vm con il metodo di Arnoldi e una matrice in forma di
Hessenberg H.
        CALL Arnoldi(A,Vm,H,m,n)
                ! OSSERVAZIONE: A prescindere dal fatto che si interrompa prima il vettore conclusivo
è in Vm(:,m+1)

        ! Calcoliamo la SVD di H segnato.
        DO i=1,m
                H(i,i)=H(i,i)-1
```

```
        END DO

        CALL calcola_svd(m+1,m,H,d,U,ST) ! Attenzione, è la trasposta.

        ! Trasponiamo:
        DO i=1,m
        DO j=1,m
        S(i,j)=ST(j,i)
        END DO
        END DO

        ! Scegliamo il nuovo vettore iniziale. DIFFERENZA rispetto all'algoritmo precedente.
        ! Calcolod di xg=Vm*S(:,m):
        xg=0
        DO i=1,n
        DO j=1,m
                xg(i)=xg(i)+Vm(i,j)*S(j,m)
        END DO
        END DO

        ! Calcolo di R^G=(A-Id)X^G il residuo precedente:
        r=Matmul(A,xg)-xg

        ! Calcolo di w:
        w=Matmul(A,Vm(:,m+1))-Vm(:,m+1)

        ! Norma di w:
        norm=0
        DO i=1,n
        norm=norm+w(i)**2
        END DO

        ! Calcolo di beta:
        beta=-Dot_product(w,r)/norm

        ! Calcolo della nuova approsimazione:
        w=xg+beta*Vm(:,m+1)

        ! Ricalcoliamo la norma:
        norm=0
        DO i=1,n
        norm=norm+w(i)**2
        END DO
        norm=sqrt(norm)

        ! Normalizziamo rispetto alla norma 2:
        w=w/norm
        w=abs(w)/sum(abs(w))

        ! Calcoliamo il residuo:
        p=sum(abs(Matmul(A,w)-w))/sum(abs(Matmul(A,w)))
        write(*,*) p
        ! Iteriamo sfruttando il vettore prima calcolato:
        Vm(:,1)=w

        ! Aumentiamo il contatore di controllo.
        k=k+1
END DO


CALL CPU_TIME(finish)
write(*,*) "Time:" , finish-start


! Trascrizione dei valori della matrice nel file GRArnoldi.txt
w=w/sum(v)
write (2,100) n
DO i=1,n
        WRITE (2,200) w(i)
END DO
```

```
! Formati usati per la trascrizione.
100 FORMAT(3I7)          ! Formato per righe et colonne.
200 FORMAT(F30.28)       ! Formato per i numeri.

END PROGRAM MArnoldiGR
```

Programma VIII: MArnoldi2


```
! Programma: MArnoldi2
! Autore: Giulio Del Corso
! Input: Richiede la presenza di un file txt nella cartella di esecuzione.
! Output: Genera un file pgm che contiene la matrice stocastica associata al problema.
! Il programma nel mentre genere una base ortonormale V_(m+1)

! Il vettore iniziale viene scelto in maniera arbitraria.
! p è il residuo e tol è l'indicatore di tolleranza.


PROGRAM MArnoldi2
IMPLICIT NONE
INTEGER , PARAMETER :: dp=KIND(0.d0)
INTEGER :: i , j , n , k=0
INTEGER , PARAMETER :: m=3 , maxit=100
        ! Numero di iterazioni del metodo di Arnoldi, numero massimo di passaggi.
REAL(dp) :: p=1 , start, finish , norm , beta
REAL(dp) , DIMENSION(:,:) , ALLOCATABLE :: A , Vm
REAL(dp) , DIMENSION(:) , ALLOCATABLE :: v , w , xg
REAL(dp) :: d(m) , H(m+1,m)=0 , S(m,m) , U(m+1,m+1)
        ! H è direttamente H segnato, ossia ha l'ultima colonna di 0.


! Ricavare i dati dalla matrice:
OPEN(UNIT=1,FILE="A.txt",STATUS="UNKNOWN",ACTION="READWRITE")
OPEN(UNIT=2,FILE="MGRArnoldi2.txt",STATUS="UNKNOWN",ACTION="READWRITE")

READ(1,*) n
ALLOCATE(A(n,n),v(n),xg(n),w(n),Vm(n,m+1))

DO i=1,n
        DO j=1,n
        read(1,*) A(i,j)
        END DO
END DO


! Trasponiamo la matrice per ricondurci al caso noto:
A=transpose(A)


! Inizio calcolo tempo.
CALL CPU_TIME(start)


! Inizializziamo un vettore di norma unitaria:
CALL random_number(v) ; v=v/sum(v) ; Vm(:,1)=v


! Arnoldi m-step
DO WHILE ((p>1E-14).and.(k<maxit))
        ! Calcoliamo una base ortonormale Vm con il metodo di Arnoldi e una matrice in forma di
Hessenberg H.
        CALL Arnoldi(A,Vm,H,m,n)
                ! OSSERVAZIONE: A prescindere dal fatto che si interrompa prima il vettore conclusivo
è in Vm(:,m+1)

        ! Calcoliamo la SVD di H segnato.
        DO i=1,m
                H(i,i)=H(i,i)-1
        END DO

        CALL calcola_svd(m+1,m,H,d,U,S) ! Attenzione, è la trasposta.

        ! Trasponiamo:
        S=transpose(S)


        ! 1: Calcoliamo un primo p:
        xg=Matmul(Vm(:,1:m),S(:,m))
        p=abs(sum(Matmul(A,xg)-xg)/sum(xg))
```

```
        ! 2: Ricaviamo w e la norma 2 di w:
        w=Matmul(A,Vm(:,m+1))-Vm(:,m+1)

        norm=0
        DO i=1,n
                norm=norm+w(i)**2
        END DO


        ! 3: Ricaviamo beta segnato:
        beta=-p*Dot_product(w,Matmul(Vm,U(:,m)))/norm


        ! 4: Calcolo della nuova approssimazione:
        v=0
        DO i=1,n
                DO j=1,m
                v(i)=v(i)+Vm(i,j)*S(j,m)
                END DO
                v(i)=v(i)+Vm(i,m+1)*beta
        END DO


        ! 5: Calcolo di w e del residuo in norma 1:
        w=Matmul(A,v)
        p=sum(abs(w-v))/sum(abs(w))


        ! 6: Calcolo del nuovo vettore di iterazione:
        norm=0
        DO i=1,n
                norm=norm+w(i)**2
        END DO
        norm=sqrt(norm)

        v=w/norm
        Vm(:,1)=v


        ! 7: Iterazione del contatore:
        k=k+1

END DO

write(*,*) k

CALL CPU_TIME(finish)
write(*,*) "Time:" , finish-start


! Trascrizione dei valori della matrice nel file GRArnoldi.txt
w=w/sum(v)
write (2,100) n
DO i=1,n
        WRITE (2,200) w(i)
END DO


! Formati usati per la trascrizione.
100 FORMAT(3I7)         ! Formato per righe et colonne.
200 FORMAT(F30.28)      ! Formato per i numeri.

END PROGRAM MArnoldi2
```

5.1 Esempi numerici:

Calcolatore utilizzato:

Processore	Intel® Core™ 2 Duo CPU T7250 2.00 GHz
RAM	3 GB
Sistema operativo	UBUNTU 14.04.1 LTS
Precisione di macchina	$2.15 \cdot 10^{-16}$
Programma di scrittura	Gedit
Linguaggio	Fortran90
Compilatore	Gfortran
Librerie utilizzate	Lapack

Confronto 1:

Tempi di esecuzione dei diversi algoritmi per raggiungere un residuo in norma 1 di 10^{-15} al variare del damping factor d per matrici di dimensione 4000×4000

Damping factor	Metodo delle Potenze	Arnoldi	MArnoldi	MArnoldi2
0.5	1.449	1.150	1.535	1.248
0.6	1.649	1.211	1.665	1.266
0.7	1.864	1.448	1.692	1.445
0.8	2.083	1.772	1.731	1.284
0.85	2.243	1.962	1.961	1.457
0.95	2.451	1.986	1.842	1.668
0.99	2.606	1.979	1.862	1.864

Confronto 2:

Tempi di esecuzione dei diversi algoritmi per raggiungere un residuo in norma 1 di 10^{-15} al variare del damping factor d per matrici di dimensione $10^4 \times 10^4$

Damping factor	Metodo delle Potenze	Arnoldi	MArnoldi	Marnoldi2
0.5	7.108	5.332	4.086	3.239
0.85	9.513	7.694	6.423	5.528

Confronto 3:

Confronto fra la soluzione "esatta" calcolata con il metodo delle potenze aumentando il numero delle iterazioni e i due metodi di Arnoldi.

$d = 0.85$; matrice 4000×4000

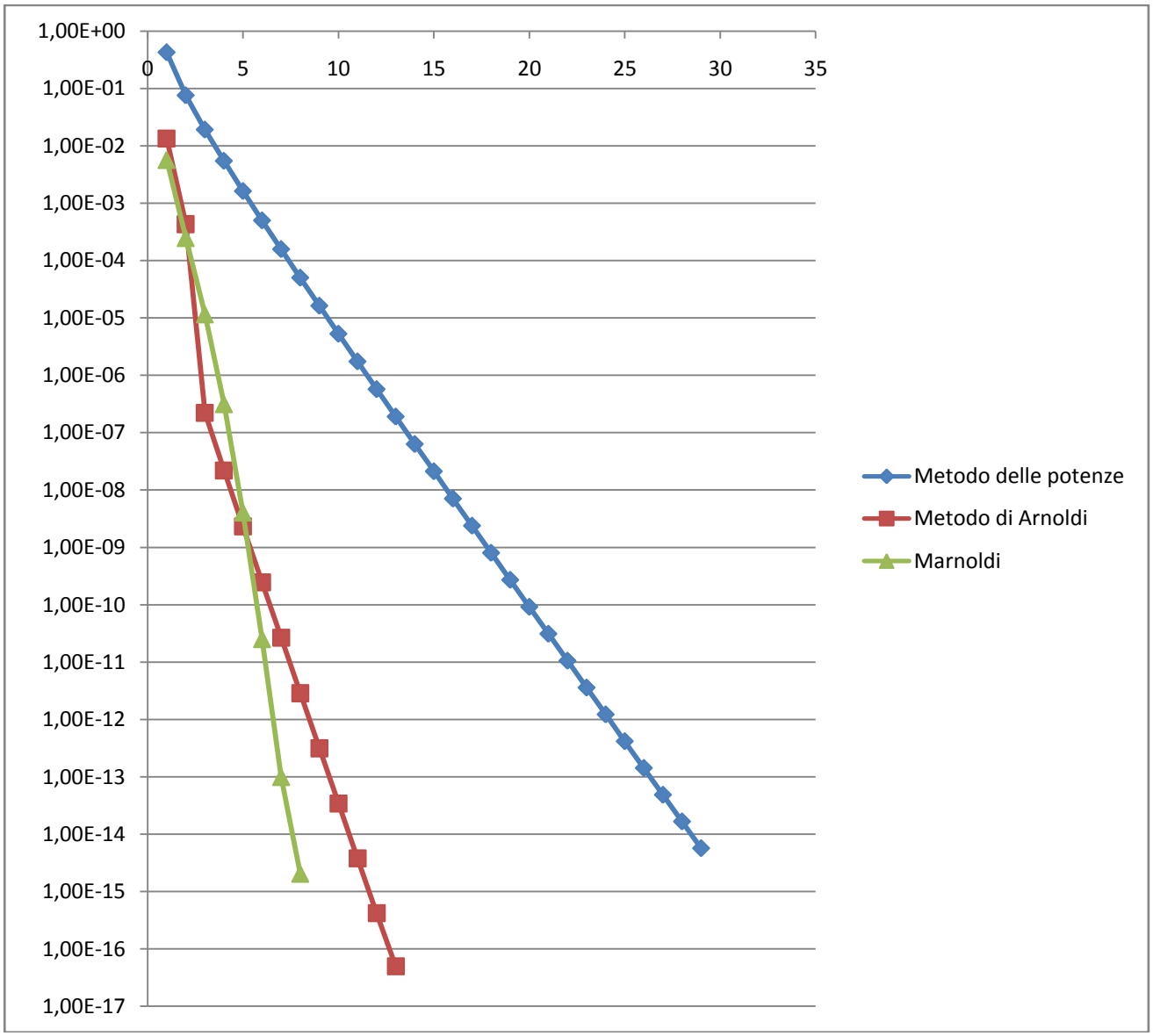
Massima differenza: $< 10^{-16}$ (Precisione di macchina
Norma della differenza: $< 10^{-15}$

Confronto 4:

Decrescita del residuo a parità di iterazioni:

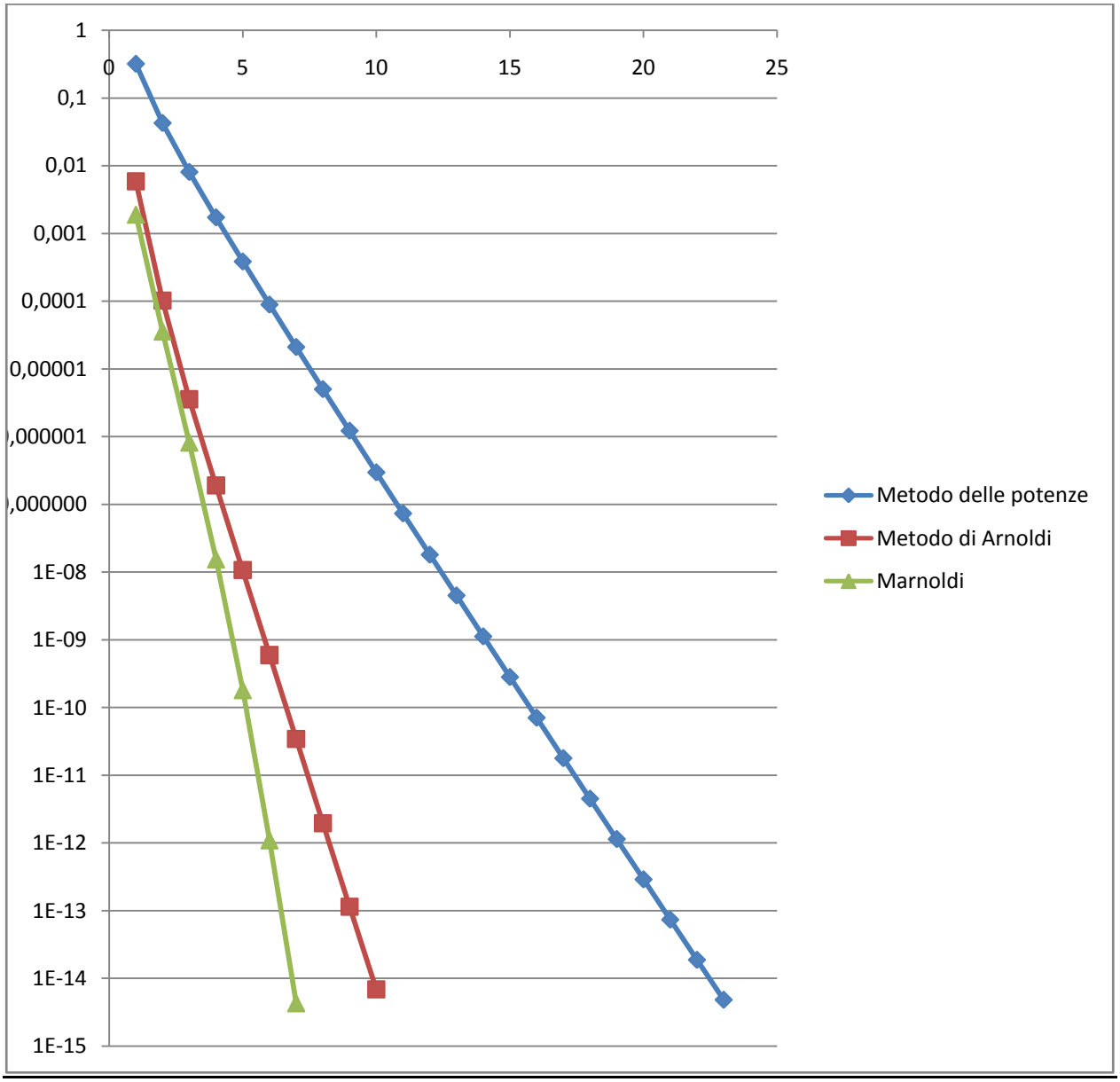
$d = 0.8$

Numero di iterazioni	Metodo delle potenze	Arnoldi	MArnoldi
1	0.42427257092821274	1.3303388312628790E-002	5.6425870579058070E-003
2	7.5526464041330829E-002	4.2891244672388205E-004	2.4623136710577110E-004
3	1.9065007200421112E-002	2.1985602703651544E-005	1.1473586758645945E-005
4	5.4324670321383647E-003	2.1788041854797908E-006	3.1256358827226247E-007
5	1.6163675097766374E-003	2.3137160598233835E-007	4.0214770627856488E-009
6	4.9783575738007063E-004	2.4494815365808863E-008	2.4990646176254821E-011
7	1.5691045494432609E-004	2.6571152450498871E-009	9.8380356053536082E-014
8	4.9969409045388363E-005	2.8628269763326271E-010	2.0275664827651426E-015
9	1.6196562996585812E-005	3.1414627865687951E-011	
10	5.2519340094516978E-006	3.4310568083507040E-012	
11	1.7319549821895466E-006	3.7965235867414624E-013	
12	5.6813845468367042E-007	4.1911696565277112E-014	
13	1.8929298259702629E-007	4.9936527156417617E-015	
14	6.2735861237653528E-008		
15	2.1035784089469219E-008		
16	7.0325463732588342E-009		
17	2.3687245212164476E-009		
18	7.9714383893400836E-010		
19	2.6945325042938319E-010		
20	9.1152267103680046E-011		
21	3.0898072295171275E-011		
22	1.0496117131908803E-011		
23	3.5669791637526690E-012		
24	1.2156577827715509E-012		
25	4.1407200205515610E-013		
26	1.4148975231460109E-013		
27	4.8245804053215213E-014		
28	1.6538907830071503E-014		
29	5.6572720683392697E-015		


1,00E+00
1,00E-01
1,00E-02
1,00E-03
1,00E-04
1,00E-05
1,00E-06
1,00E-07
1,00E-08
1,00E-09
1,00E-10
1,00E-11
1,00E-12
1,00E-13
1,00E-14
1,00E-15
1,00E-16
1,00E-17
0
5
10
15
20
25
30
35
Metodo delle potenze
Metodo di Arnoldi
Marnoldi

$d = 0.6$

Numero di iterazioni	Metodo delle potenze	Arnoldi	MArnoldi
1	0.31820442819615935	5.8344004565825183E-003	1.8940717760790866E-003
2	4.2483636023248575E-002	1.0164802949853778E-004	3.5544927418122351E-005
3	8.0430499126776210E-003	3.5447713430841568E-006	8.1492140500482201E-007
4	1.7188665218875183E-003	1.8934765263671058E-007	1.5349496022704344E-008
5	3.8357158679266798E-004	1.0639620628395570E-008	1.8034959083871434E-010
6	8.8604069123485480E-005	5.9325535972313012E-010	1.0933343667267682E-012
7	2.0945017392743858E-005	3.4185226974993966E-011	4.2995934503714449E-015
8	5.0025832022740889E-006	1.9454519091256466E-012	
9	1.2161138514075597E-006	1.1385470267019975E-013	
10	2.9575486306877743E-007	6.8849332355287070E-015	
11	7.3149354252934496E-008		
12	1.7996553670107424E-008		
13	4.4970833371608563E-009		
14	1.1178243045957418E-009		
15	2.8111085676038247E-010		
16	7.0484402927943612E-011		
17	1.7805583762508896E-011		
18	4.4940807667056756E-012		
19	1.1393060431706681E-012		
20	2.8906698811958370E-013		
21	7.3502740894661189E-014		
22	1.8759475852066221E-014		
23	4.8242117765828763E-015		


1
0
5
10
15
20
25
0,1
0,01
0,001
0,0001
0,00001
,000001
,000000
1E-08
1E-09
1E-10
1E-11
1E-12
1E-13
1E-14
1E-15
Metodo delle potenze
Metodo di Arnoldi
Marnoldi

Conclusioni:

Il metodo implementato non sfrutta la sparsità della matrice e dunque l'algoritmo modificato, che ha un prodotto matrice vettore ulteriore, ne risulta penalizzato.
Concentrandoci però esclusivamente sul numero di passi necessari a raggiungere la precisione richiesta possiamo osservare che MArnoldi risulta essere il più efficiente.

Dalla stima fatta della convergenza risulta inoltre evidente come all'aumentare del damping factor diventi sempre più preferibile MArnoldi (MArnoldi2) rispetto ad Arnoldi con restart (questo emerge anche dalla sperimentazione numerica svolta).

Infine l'implementazione successiva, MArnoldi2, risulta anche migliore dal punto di vista della velocità di convergenza.