Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

PROGETTO 1 ALTERNATIVO

METODI DEL CALCOLO SCIENTIFICO

Clemente Diego
MAT. 933223

Sommario

1. Introduzione

1.1 Obiettivo del progetto

L'obiettivo della relazione è quello di implementare una mini-libreria contenente dei risolutori iterativi per sistemi lineari, limitandoci al caso di matrici simmetriche e definite positive; i metodi iterativi implementati sono i seguenti:

- metodo di Jacobi
- metodo di Gauß-Seidel
- metodo del gradiente
- metodo del gradiente coniugato

Ogni metodo parte da un vettore iniziale nullo e si arresta al soddisfacimento di un criterio di convergenza basato sull' residuo relativo, oppure al raggiungimento di un numero massimo di iterazioni.

I metodi sono testati su matrici sparse fornite in formato '.mtx', su differenti valori di tolleranza ed i risultati ottenuti verranno analizzati in termini di accuratezza (valutando l'errore relativo tra la soluzione esatta x e la soluzione approssimata ottenuta), numero di iterazioni e tempo di esecuzione.

1.2 Metodi iterativi

I metodi iterativi servono alla risoluzione di sistemi lineari del tipo:

$$A\vec{x} = \vec{b}$$

Il processo di risoluzione dei metodi iterativi è il seguente: partendo da un vettore iniziale nullo $\overrightarrow{x_0}$ creiamo una sequenza di vettori $\overrightarrow{x_k}$ che si avvicinano sempre di più alla soluzione esatta $\vec{x}$ ad ogni iterazione; arresteremo le iterazioni quando la k-esima iterata soddisfa:

$$\frac{\left\|A\overrightarrow{x_k} - \vec{b}\right\|}{\left\|\vec{b}\right\|} < tol$$

I metodi iterativi possiamo classificarli in: metodi iterativi stazionari (tramite la procedura di splitting) e metodi iterativi non stazionari (scalare α che dipende dall'iterazione).

2. Struttura della libreria

In questo capitolo è descritta l'architettura utilizzata per l'implementazione del Progetto1-alternativo secondo le richieste descritte.

2.1 Tecnologie utilizzate

Tra le tecnologie utilizzate, troviamo *PyCharm* come ambiente di sviluppo e, *Python* come linguaggio di programmazione.

Per lo svolgimento delle operazioni elementari e la gestione delle strutture dati, quali matrici e vettori, sono state utilizzate le librerie *NumPy* e *SciPy*.

2.2 Descrizione della struttura

Nella libreria abbiamo una cartella **dati** contenente tutte le matrici fornite per calcolo delle performance: *spa1.mtx, spa2.mtx, vem1.mtx e vem2.mtx*; e una cartella **risultati** contenete i risultati ottenuti dall'esecuzione della libreria.

Nella cartella lib sono presenti i moduli Python relativi alla implementazione della libreria:

- ***controlli.py***: contiene una serie di funzioni di verifica sulla validità dei dati di input. Nello specifico:
  - *controlliDim(A, x):* verifica che la matrice A sia quadrata, che le sue dimensioni siano compatibili con il vettore x e che la diagonale non contenga zeri, al fine di garantire la fattibilità dei metodi iterativi.
  - *is_simmetrica(A):* controlla che la matrice sia simmetrica
  - *is_positive_definite(A):* valuta se A è definita positiva, sfruttando il fatto che tutti gli autovalori devono essere strettamente positivi.
  - *controlloGradientePossibile(A):* funzione composta che richiama i due controlli sopra per assicurarsi che le condizioni necessarie per applicare il metodo del gradiente (simmetria e definita positività) siano soddisfatte.
- ***esegui.py***: contiene le funzioni per eseguire e confrontare cinque metodi iterativi per risolvere sistemi lineari $A\vec{x} = \vec{b}$ mediante l'utilizzo dei metodi: *runJacobi(…), runGaussSeidel(…), runGradiente(…)* e *runGradienteConiugato(…).*
  Ogni metodo:
  - Richiama il rispettivo metodo iterative implementato,
  - Stampa l'errore relativo, il numero di iterazioni e il tempo di esecuzione,
  - Restituisce questi tre valori in una lista.

  La funzione *routine (…)* esegue tutti i metodi iterativi in sequenza per un confronto diretto.

- ***metodIterativi.py***: al suo interno sono implementati i quattro metodi iterativi previsti dal progetto, ciascuno dei quali restituisce l'errore relativo finale, il numero di iterazioni effettuate e il tempo di esecuzione. Le funzioni presenti sono:
    - *metodo_jacobi(...):* implementa il metodo di Jacobi. Per ogni iterazione viene calcolata $\overrightarrow{x_k}$ nel seguente modo:

      $$\overrightarrow{x_{k+1}} = \overrightarrow{x_k} + D^{-1}(\overrightarrow{r_k})$$

      dove $\overrightarrow{r_k}$ è il residuo al passo k e D è la matrice diagonale di A.
    - *metodo_gaus_seidelMyLU(...):* versione del metodo di Gauss-Seidel che utilizza una funzione personalizzata (SolvTriangularLower) per risolvere sistemi triangolari inferiori.
    - metodo_gaus_seidel(...): implementazione alternativa del metodo di Gauss-Seidel che utilizza *scipy.sparse.linalg.spsolve_triangular*, specifica per matrici sparse per risolvere il sistema triangolare inferiore.

      Per ogni iterazione viene calcolata $\overrightarrow{x_k}$ nel seguente modo:

      $$\overrightarrow{x_{k+1}} = \overrightarrow{x_k} + \text{solve\_triangular}(L, \overrightarrow{r_k})$$

      dove $\overrightarrow{r_k}$ è il residuo al passo k e L è la matrice triangolare inferiore estratta da A.
    - *metodo_gradiente(...):* implementa il metodo del gradiente, sfruttando la direzione del residuo come direzione di discesa. Adatto a matrici simmetriche e definite positive.
    - *metodo_gradiente_coniugato(...)*: metodo del gradiente coniugato, aggiorna la direzione di discesa in modo ortogonale rispetto alle iterazioni precedenti, garantendo la convergenza in al più n passi. Adatto a matrici simmetriche e definite positive.

  Tutte le funzioni prevedono un controllo di validità sulla matrice A e sulla compatibilità dimensionale, e si arrestano al raggiungimento della tolleranza imposta o di un numero massimo di iterazioni (MAXITE = 200000).

- ***risolvi.py***: contiene funzioni utili che supportano l'esecuzione dei metodi iterativi. Le principali operazioni offerte sono:

    - *SolvTriangularLower(A, b)*: esegue la sostituzione in avanti per la risoluzione di un sistema triangolare inferiore.

- *InverseMatrixDiagonal(A):* calcola l'inversa della matrice diagonale estratta da A. Viene utilizzata per esplicitare formule nei metodi iterativi, in particolare in Jacobi.
- *errorRelativo(x, x_k):* restituisce l'errore relativo tra la soluzione esatta x e la soluzione approssimata $\overrightarrow{x_k}$.
- *errorRelativoResiduo(A, b, x):* calcola il residuo $\vec{r} = \vec{b} - A\vec{x}$ e il suo errore relativo rispetto al termine noto $\vec{b}$.
- *inizializza(A, b):* prepara l'ambiente per i metodi iterativi, generando una soluzione iniziale nulla $\overrightarrow{x_0}$,il residuo iniziale, l'errore relativo del residuo e azzera il contatore delle iterazioni.

Infine, il modulo ***main.py*** che rappresenta il punto iniziale del progetto. Questo script esegue in modo automatizzato l'intero ciclo di test sui metodi iterativi, applicandoli alle matrici fornite e salvando i risultati in file xml. Le sue principali funzionalità sono:

- **Lettura dei file ".mtx":** le matrici vengono caricate da file utilizzando mmread e convertite in formato sparso CSR.
- **Costruzione del sistema di test**: per ogni matrice, viene creato un sistema lineare con soluzione esatta formata da tutti 1 e il corrispondente termine noto $\vec{b} = A\vec{x}$
- **Esecuzione dei metodi**: per ciascuna delle tolleranze specificate (1e-4, 1e-6, 1e-8, 1e-10), vengono eseguiti i cinque metodi (Jacobi, Gauss-Seidel classico e custom, Gradiente, Gradiente Coniugato) tramite la funzione routine() del modulo esegui.py.
- **Raccolta dei risultati**: gli errori relativi, il numero di iterazioni e i tempi di calcolo vengono salvati in strutture dati specifiche.
- **Esportazione in XML**: i risultati vengono salvati in file .xml, uno per ciascuna matrice, all'interno della cartella *risultati*, tramite la funzione *salvaInXML(…).*

Nella cartella lib è presente il modulo metodIterativiMulti.py che implementa tutti e quattro i metodi in un'unica funzione, cambiando il metodo di aggiornamento della $\overrightarrow{x_k}$ a seconda del metodo scelto. Ogni metodo viene eseguito fino al raggiungimento della tolleranza specificata o al superamento del numero massimo di iterazioni consentite (MAXITE), fissato a 200000. Al termine, restituire i valori richiesti: errore relativo, numero di iterazioni, tempo.

```
#inizio a calcolare il tempo
start = time.time()
#inizio iterazioni
while (errR > tol):

    if (nIte < MAXITE):
        #controllo come aggiornare x_k
        if type == 1:
            x_k = updateJacobi(D_inv, x_k, r)
            r, errR = ri.errorRelativoResiduo(A, b, x_k)
        elif type == 2:
            x_k = updateGausSeidel(A, x_k, r)
            r, errR = ri.errorRelativoResiduo(A, b, x_k)
        elif type == 3:
            x_k = updateGradiente(A, x_k, r)
            r, errR = ri.errorRelativoResiduo(A, b, x_k)
        else:
            x_k, d, r, errR= updateGradienteConiugato(A, b, x_k, r, d)
        nIte += 1
    else:
        raise ValueError("Arrivato al massimo di iterazioni")
stop = time.time()
# calcolo il tempo di esecuzione
timeIte = stop - start
# calcolo errore relativo
print(errR)
errRel = ri.errorRelativo(x, x_k)
return errRel, nIte, timeIte
```

Figura 1 modulo metodIterativiMulti.py

Il modulo metodIterativiMulti.py non è stato utilizzato nei test in quanto sceglie il metodo da usare a ogni iterazione tramite una struttura *if-elif*. Questo la rende più flessibile, ma rallenta leggermente l'esecuzione, soprattutto se le iterazioni sono tante, rendendo i tempi meno precisi nei confronti.

3.Analisi dei Risultati

3.1 Composizione delle matrici

Le strutture delle matrici utilizzate si presentano nel seguente modo:

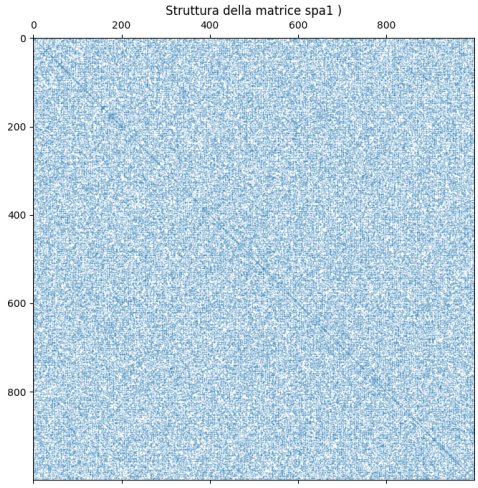


Figura 2 spa1.mtx

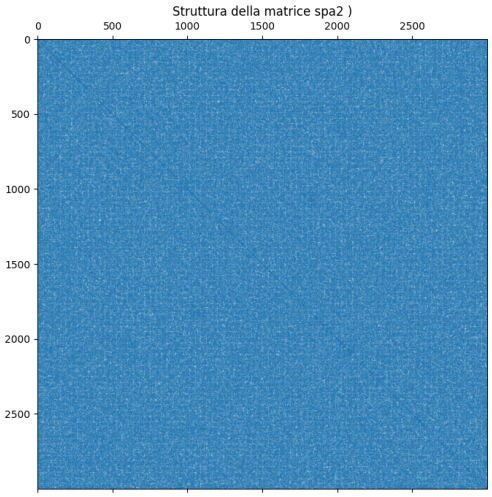


Figura 3 spa2.mtx

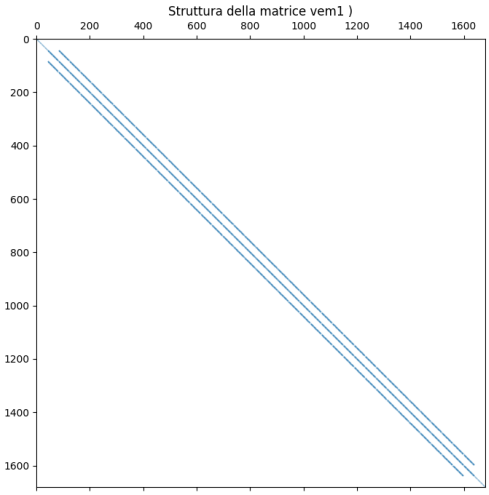


Figura 5 vem1.mtx

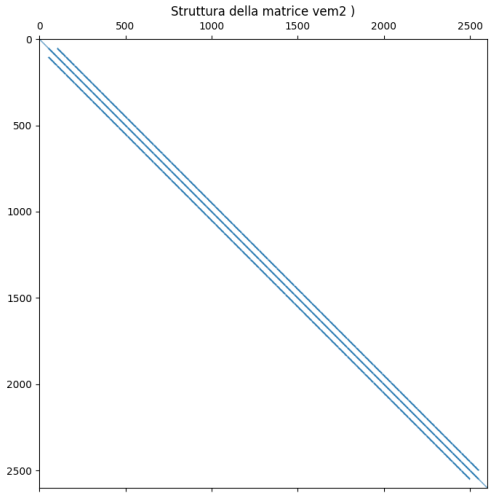


Figura 4 vem2.mtx

Le 4 matrici fornite sono in forma sparsa e presentano le seguenti caratteristiche raccolte nella seguente Tabella 1:

MATRICE	DIMENSIONI	DATI NON ZERO	DENSITÀ	NUMERO DI CONDIZIONAMENTO
SPA1.MTX	1000x1000	18.2434	18,24%	2048
SPA2.MTX	3000x3000	1.633.298	18,13%	1412
VEM1.MTX	1681x1681	13.385	0,47%	325
VEM2.MTX	2601x2601	21.225	0,31%	507

Tabella 1 Analisi Matrici

Dall'analisi della struttura e del numero di condizionamento delle matrici utilizzate, si possono trarre alcune considerazioni preliminari sull'efficacia dei metodi iterativi considerati. Le matrici *vem1.mtx* e *vem2.mtx* presentano un numero di condizionamento contenuto: ciò le rende particolarmente adatte ai metodi del Gradiente e del Gradiente Coniugato, garantendo una buona velocità di convergenza.

Al contrario, le matrici *spa1.mtx* e *spa2.mtx* mostrano una struttura sparsa più disordinata e un condizionamento più elevato, quindi l'utilizzo del metodo del Gradiente potrebbe risultare meno efficiente.

3.2 Risultati ottenuti spa1.mtx

ERRORE RELATIVO

TOL.	Jacobi	Gauss-Seidel	Gauss-Seidel MySolve	Gradiente	Gradiente Coniugato
1E-04	1,77E-03	1,82E-02	1,82E-02	3,46E-02	2,08E-02
1E-06	1,80E-05	1,30E-04	1,30E-04	9,68E-04	2,55E-05
1E-08	1,82E-07	1,71E-06	1,71E-06	9,82E-06	1,32E-07
1E-10	1,85E-09	2,25E-08	2,25E-08	9,82E-08	1,21E-09

Tabella 2 Errore Relativo spa1.mtx

NUMERO ITERAZIONI

TOL.	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	115	9	9	143	49
1E-06	181	17	17	3577	134
1E-08	247	24	24	8233	177
1E-10	313	31	31	12919	200

Tabella 3 Numero di iterazioni spa1.mtx

TEMPO

TOL.	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	0,0209	0,0238	0,1712	0,0609	0,0286
1E-06	0,0426	0,0606	0,3815	2,2930	0,1658
1E-08	0,0878	0,0796	0,5207	2,4010	0,1334
1E-10	0,2337	0,1308	0,5630	3,7832	0,1584

Tabella 4 Tempo di esecuzione spa1.mtx

Per quanto riguarda l'errore relativo, il metodo di Jacobi ha mostrato buoni risultati per tolleranze moderate (1×10^{-4} e 1×10^{-6}). Tuttavia, al crescere della precisione richiesta (1×10^{-8} e 1×10^{-10}), il Gradiente Coniugato si distingue come il metodo più accurato, raggiungendo l'errore più basso in assoluto ($1{,}21\times10^{-9}$). Al contrario, il metodo del Gradiente si è dimostrato il meno preciso in tutte le tolleranze testate.

Analizzando il numero di iterazioni necessarie alla convergenza, il metodo di Gauss-Seidel si è confermato particolarmente efficiente, convergendo in un numero significativamente ridotto di passi rispetto agli altri metodi. Il metodo del Gradiente, invece, richiede il maggior numero di iterazioni.

Dal punto di vista dei tempi di esecuzione, i metodi di Jacobi e Gauss-Seidel risultano i più performanti, con tempi comparabili. Tuttavia, che la versione Gauss-SeidelMySolve, che utilizza una mia implementata per la soluzione del sistema triangolare inferiore, risulta meno efficiente rispetto all'implementazione basata sulla libreria SciPy. Il metodo del Gradiente, anche sotto questo aspetto, si conferma il meno efficiente in termini di tempo computazionale.

In conclusione, per la matrice spa1.mtx, i metodi di Jacobi e Gauss-Seidel rappresentano le soluzioni più efficienti in termini di tempo e iterazioni, mentre il Gradiente Coniugato garantisce la massima accuratezza, ma un costo computazionale leggermente maggiore. Il metodo del Gradiente risulta inadatto per la matrice spa1.mtx, a causa dell'elevato condizionamento della matrice, che ne influenza negativamente le prestazioni.

3.3 Risultati ottenuti spa2.mtx

ERRORE RELATIVO

TOL.	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	1,77E-03	2,60E-03	2,60E-03	1,81E-02	9,82E-03
1E-06	1,67E-05	5,14E-05	5,14E-05	6,69E-04	1,20E-04
1E-08	1,57E-07	2,79E-07	2,79E-07	6,87E-06	5,59E-07
1E-10	1,48E-09	5,57E-09	5,57E-09	6,94E-08	5,32E-09

Tabella 5 Errore Relativo spa2.mtx

NUMERO ITERAZIONI

TOL	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	36	5	5	161	42
1E-06	57	8	8	1949	122
1E-08	78	12	12	5087	196
1E-10	99	15	15	8285	240

Tabella 6 Numero di iterazioni spa2.mtx

TEMPO

TOL.	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	0,0634	0,0505	0,8366	0,4847	0,2424
1E-06	0,1046	0,0779	1,2638	5,0876	0,6170
1E-08	0,1369	0,1218	1,8924	13,3021	0,9126
1E-10	0,1725	0,1566	2,3605	21,4634	1,0905

Tabella 7 Tempo di esecuzione spa2.mtx

Per quanto riguarda l'errore relativo, il metodo di Jacob risulta il più accurati per tutte le tolleranze considerate, ottenendo valori leggermente inferiori rispetto a Gauss-Seidel, Gauss-SeidelMySolve. Il metodo del Gradiente, invece, evidenzia prestazioni peggiori, risultando il meno accurato per ogni tolleranza.

In termini di numero di iterazioni, il metodo di Gauss-Seidel si dimostra estremamente efficiente, convergendo in un numero molto ridotto di passi, mentre Jacobi richiede un numero maggiore di iterazioni, ma comunque contenuto. Il metodo del Gradiente, invece, richiede il maggior numero di iterazioni.

Considerando i tempi di esecuzione, Gauss-Seidel risulta il metodo più veloce tra quelli testati, seguito da Jacobi. Anche in questo caso, il metodo del Gradiente si conferma il meno efficiente, con tempi che aumentano drasticamente con la tolleranza richiesta.

In conclusione, per la matrice spa2.mtx, i metodi di Jacobi e Gauss-Seidel si confermano i più efficienti in termini di tempo e iterazioni mentre il metodo del Gradiente, ancora una volta, risulta inadeguato, probabilmente a causa delle caratteristiche della matrice e del suo condizionamento.

3.4 Risultati ottenuti vem1.mtx

ERRORE RELATIVO

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	3,54E-03	3,51E-03	3,51E-03	2,70E-03	4,08E-05
1E-06	3,54E-05	3,53E-05	3,53E-05	2,71E-05	3,73E-07
1E-08	3,54E-07	3,52E-07	3,52E-07	2,70E-07	2,83E-09
1E-10	3,54E-09	3,51E-09	3,51E-09	2,71E-09	2,19E-11

Tabella 8 Errore Relativo vem1.mtx

NUMERO ITERAZIONI

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	1314	659	659	890	38
1E-06	2433	1218	1218	1612	45
1E-08	3552	1778	1778	2336	53
1E-10	4671	2338	2338	3058	59

Tabella 9 Numero di iterazioni vem1.mtx

TEMPO

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	0,0309	2,1056	1,7875	0,0674	0,0029
1E-06	0,0900	3,8899	3,2072	0,1239	0,0034
1E-08	0,1217	5,6152	4,7144	0,1929	0,0040
1E-10	0,1988	7,4470	6,3489	0,2146	0,0165

Tabella 10 Tempo di esecuzione vem1.mtx

Il metodo del Gradiente Coniugato si conferma nettamente il metodo più accurato per tutte le tolleranze considerate, con errori relativi che decrescono in modo significativo fino a 2,19E-11 per la tolleranza 1E-10. Gli altri tre metodi mostrano invece errori simili tra loro.

Inoltre, il metodo del Gradiente Coniugato è anche il metodo che converge nel minor numero di iterazioni, variando tra 38 e 59 al variare della tolleranza. Al contrario, il metodo Jacobi richiede un numero di iterazioni molto elevato, confermando la sua lentezza nel convergere. Il metodo del Gradiente presenta anch'esso un numero elevato di iterazioni, che cresce in modo pressoché lineare al diminuire della tolleranza.

Nonostante il numero ridotto di iterazioni, il metodo di Gauss-Seidel risulta tra i più lenti in termini di tempo computazionale, probabilmente a causa della maggiore complessità per iterazione. Il metodo Jacobi, pur richiedendo molte iterazioni, mantiene tempi di esecuzione contenuti. Il metodo del Gradiente mostra prestazioni temporali buone, paragonabili a

quelle di Jacobi. Infine, il metodo del Gradiente Coniugato risulta essere il metodo più veloce in assoluto.

In sintesi, per la matrice vem1.mtx, il metodo del Gradiente Coniugato si conferma il metodo più efficiente e preciso, dominando su tutte le metriche considerate.

3.5 Risultati ottenuti vem2.mtx

ERRORE RELATIVO

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	4,97E-03	4,95E-03	4,95E-03	3,81E-03	5,73E-05
1E-06	4,97E-05	4,94E-05	4,94E-05	3,79E-05	4,74E-07
1E-08	4,97E-07	4,96E-07	4,96E-07	3,81E-07	4,30E-09
1E-10	4,96E-09	4,95E-09	4,95E-09	3,80E-09	2,25E-11

Tabella 11 Errore relativo vem2.mtx

NUMERO ITERAZIONI

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	1927	965	965	1308	47
1E-06	3676	1840	1840	2438	56
1E-08	5425	2714	2714	3566	66
1E-10	7174	3589	3589	4696	74

Tabella 12 Numero di iterazioni vem2.mtx

TEMPO

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1E-04	0,0568	4,7475	4,2522	0,1263	0,0047
1E-06	0,1732	8,9442	7,6133	0,2493	0,0055
1E-08	0,2500	13,0798	11,1997	0,3080	0,0064
1E-10	0,2934	17,2050	15,0589	0,3828	0,0073

Tabella 13 Tempo di esecuzione vem2.mtx

Anche nel caso della matrice vem2.mtx, il metodo del Gradiente Coniugato si conferma il più accurato per tutte le tolleranze considerate, raggiungendo errori relativi inferiori rispetto agli altri metodi. Gli altri tre metodi (Jacobi, Gauss-Seidel e Gradiente) mostrano invece un comportamento simile tra loro in termini di accuratezza, con errori relativi comparabili.

Per quanto riguarda il numero di iterazioni, il Gradiente Coniugato si distingue ancora una volta per l'efficienza: il metodo converge sempre in un numero ridotto di passi, compreso tra 47 e 74 iterazioni. Al contrario, Jacobi richiede fino a 7174 iterazioni per le tolleranze più stringenti, mentre Gauss-Seidel si attesta su valori dimezzati, ma comunque elevati.

Dal punto di vista del tempo di esecuzione, il Gradiente Coniugato è di gran lunga il più rapido, con tempi sempre inferiori a 0,01 secondi, indipendentemente dalla tolleranza. Seguono i metodi di Jacobi e Gradiente, i cui tempi sono simili e generalmente contenuti; ed

infine il metodo di Gauss-Seidel che mostra tempi computazionali molto alti, nonostante il numero di iterazioni sia inferiore rispetto a Jacobi. Questo comportamento è attribuibile alla maggiore complessità computazionale per iterazione, che influisce significativamente sul tempo totale.

In sintesi, come visto per la matrice vem1.mtx anche per la matrice vem2.mtx, il metodo del Gradiente Coniugato si conferma il metodo più efficiente e preciso, dominando su tutte le metriche considerate.

3.6 Considerazioni generali

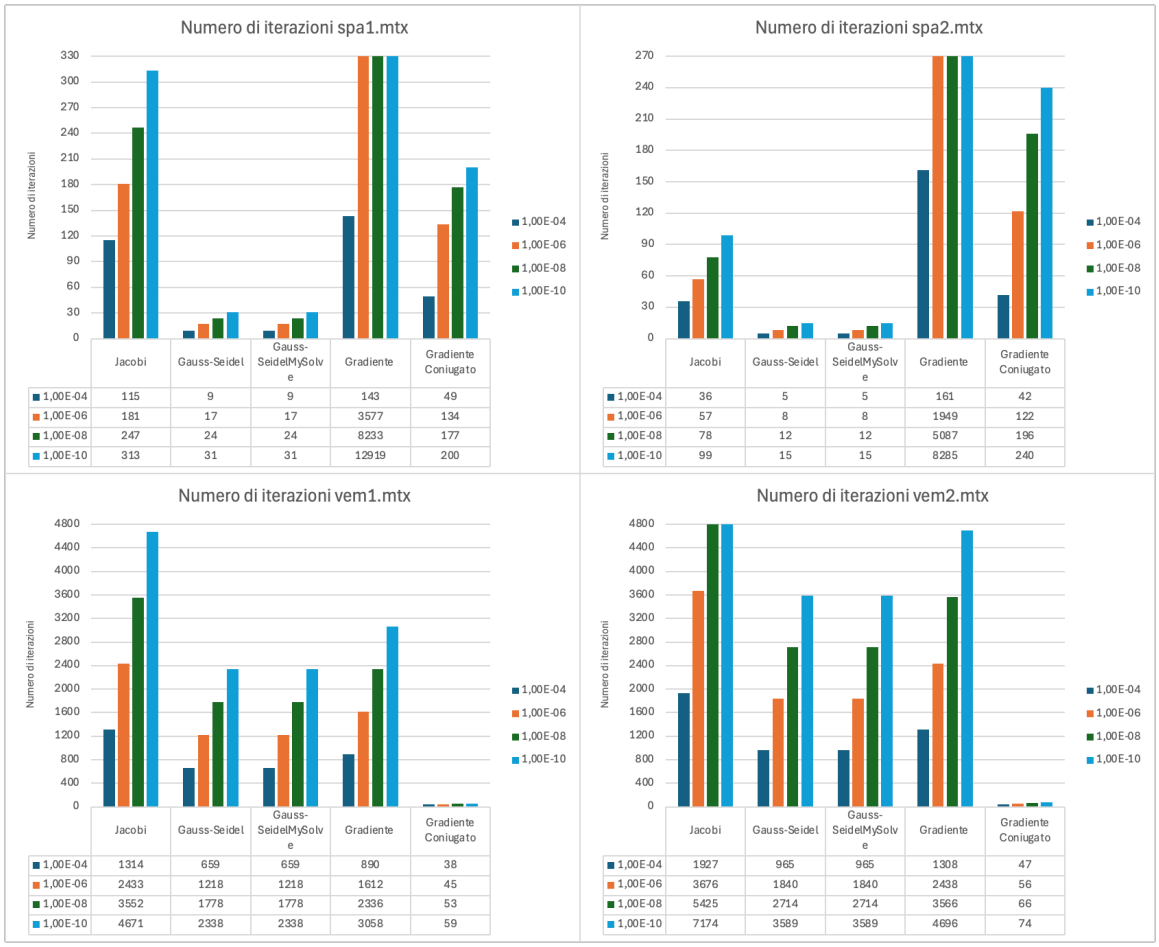


Figura 6 Iterazioni

Si può notare che, per le matrici spa1 e spa2, il metodo del Gradiente converge a soluzione con un numero di iterazioni molto alto rispetto agli altri metodi. Per quanto riguarda, invece, le matrici vem1 e vem2, il metodo di Gauss-Seidel e del Gradiente convergono a soluzione con numeri, in termini di iterazioni, abbastanza similari, mentre il metodo di Jacobi circa il doppio di Gauss-Seidel. Possiamo notare che per ogni tolleranza, il metodo del Gradiente Coniugato ha una media di iterazioni migliore indipendentemente dalla matrice utilizzata e la dimensione della matrice.

Dopo aver effettuato uno studio delle iterazioni, si è passati ad analizzare i tempi delle singole esecuzioni:

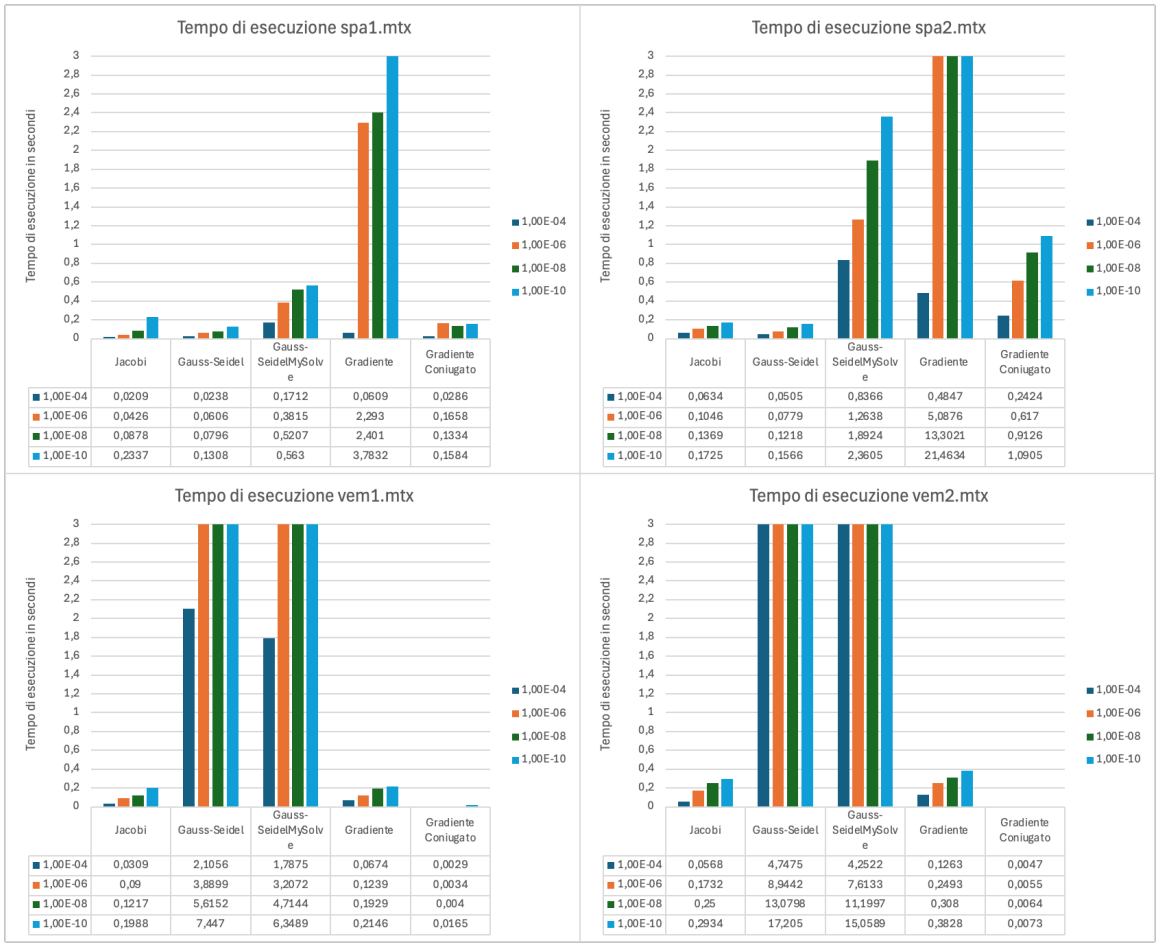
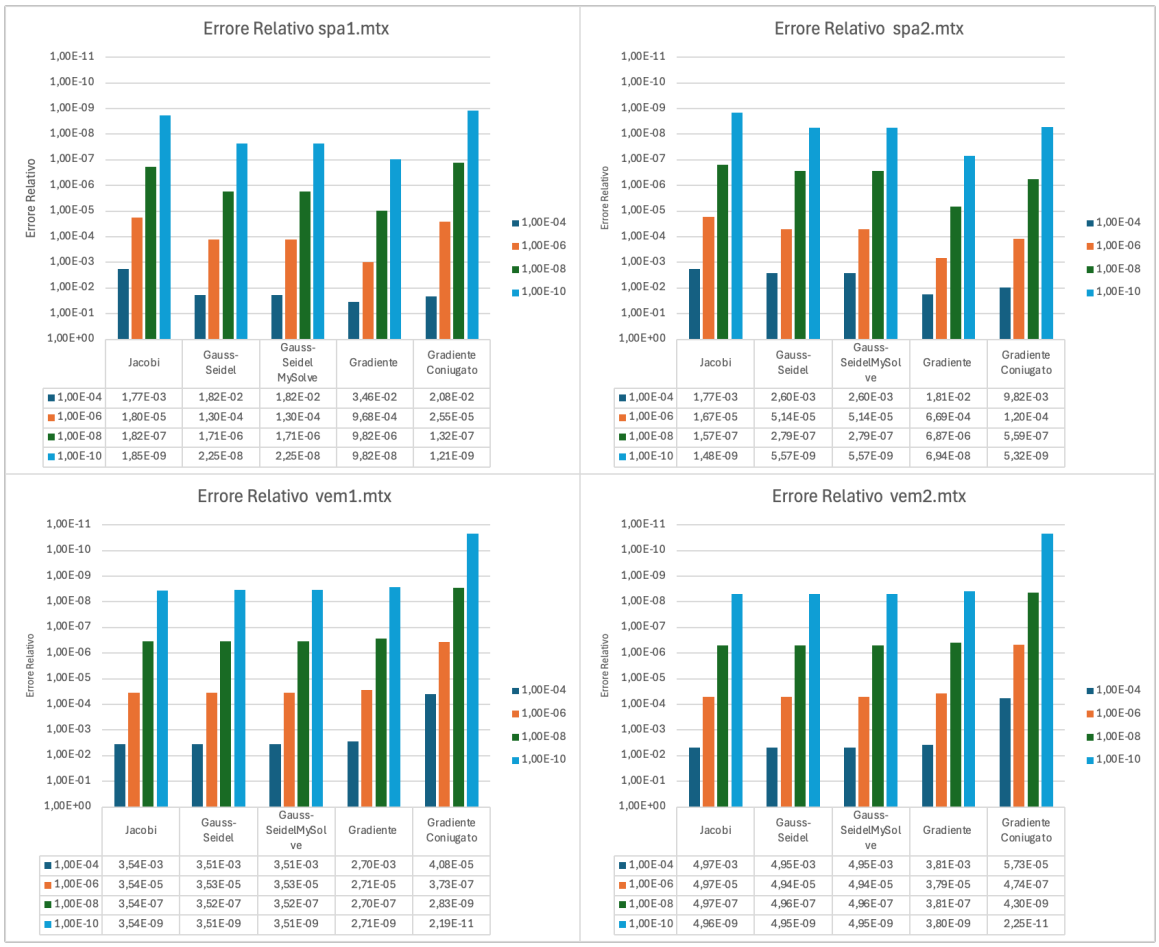


Figura 7 Tempo di esecuzione

Vediamo che il tempo d'esecuzione di una matrice cresce alla riduzione della tolleranza.

I metodi di Jacobi e del Gradiente Coniugato risultano i metodi più stabili in termini di tempo di esecuzione. Mentre il metodo del Gradiente per le matrici spa1.mtx e spa2.mtx ha tempi maggiori dovuti al mal condizionamento delle due matrici in più per la matrice spa2.mtx per la dimensione maggiore della matrice. In fine confrontando il metodo di Gauss-Seidel e di Jacobi si può notare che hanno prestazioni simili per le matrici spa1.mtx e spa2.mtx ma questo non si verifica per le matrici vem1.mtx e vem2.mtx in quanto per convergere alla soluzione il metodo di Gauss-Seidel ci mette un numero elevato di iterazioni (rispetto a spa1.mtx e spa2.mtx) che hanno un costo computazionale maggiore in quanto deve risolvere un sistema triangolare inferiore.

In ultima istanza riportiamo i dati raccolti riguardanti il calcolo dell'errore relativo percentuale:



	Jacobi	Gauss-Seidel	Gauss-Seidel MySolve	Gradiente	Gradiente Coniugato
1,00E-04	1,77E-03	1,82E-02	1,82E-02	3,46E-02	2,08E-02
1,00E-06	1,80E-05	1,30E-04	1,30E-04	9,68E-04	2,55E-05
1,00E-08	1,82E-07	1,71E-06	1,71E-06	9,82E-06	1,32E-07
1,00E-10	1,85E-09	2,25E-08	2,25E-08	9,82E-08	1,21E-09

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1,00E-04	1,77E-03	2,60E-03	2,60E-03	1,81E-02	9,82E-03
1,00E-06	1,67E-05	5,14E-05	5,14E-05	6,69E-04	1,20E-04
1,00E-08	1,57E-07	2,79E-07	2,79E-07	6,87E-06	5,59E-07
1,00E-10	1,48E-09	5,57E-09	5,57E-09	6,94E-08	5,32E-09

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1,00E-04	3,54E-03	3,51E-03	3,51E-03	2,70E-03	4,08E-05
1,00E-06	3,54E-05	3,53E-05	3,53E-05	2,71E-05	3,73E-07
1,00E-08	3,54E-07	3,52E-07	3,52E-07	2,70E-07	2,83E-09
1,00E-10	3,54E-09	3,51E-09	3,51E-09	2,71E-09	2,19E-11

	Jacobi	Gauss-Seidel	Gauss-SeidelMySolve	Gradiente	Gradiente Coniugato
1,00E-04	4,97E-03	4,95E-03	4,95E-03	3,81E-03	5,73E-05
1,00E-06	4,97E-05	4,94E-05	4,94E-05	3,79E-05	4,74E-07
1,00E-08	4,97E-07	4,96E-07	4,96E-07	3,81E-07	4,30E-09
1,00E-10	4,96E-09	4,95E-09	4,95E-09	3,80E-09	2,25E-11

Figura 8 Errore Relativo

L'analisi dell'errore relativo sulle quattro matrici considerate (spa1.mtx, spa2.mtx, vem1.mtx, vem2.mtx) evidenzia la superiorità del metodo del Gradiente Coniugato.

In particolare, per le matrici spa1.mtx e spa2.mtx, l'errore relativo risulta simile per tutti i metodi. Al contrario, per le matrici vem1.mtx e vem2.mtx, il Gradiente Coniugato mostra un netto vantaggio in termini di precisione, raggiungendo errori inferiori rispetto agli altri metodi, che invece tendono a produrre risultati simili tra loro ma con un errore più elevato.

4. Conclusioni

Il presente lavoro ha affrontato l'implementazione e l'analisi di vari metodi iterativi per la risoluzione di sistemi lineari con matrici simmetriche e definite positive. Gli algoritmi esaminati includono il metodo di Jacobi, il metodo di Gauß-Seidel, il metodo del gradiente e il metodo del gradiente coniugato.

L'obiettivo principale di questa analisi era confrontare l'efficienza e l'efficacia dei metodi iterativi considerati, valutandone le prestazioni sia in termini di tempo di esecuzione, sia in relazione al numero di iterazioni necessarie per raggiungere la convergenza. Dai risultati ottenuti emerge chiaramente che il metodo del Gradiente Coniugato si distingue come il più efficiente, combinando un numero ridotto di iterazioni con tempi di esecuzione complessivamente inferiori rispetto agli altri metodi. Al contrario, il metodo del Gradiente e il metodo di Gauss-Seidel risultano meno performanti sotto il profilo del tempo computazionale, seppur in contesti differenti: il Gradiente richiede un numero elevato di iterazioni per spa1 e spa2, mentre Gauss-Seidel, mostra tempi più lunghi probabilmente a causa della maggiore complessità computazionale per iterazione per vem1 e vem2.