Università degli Studi di Milano-Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo)

Corso di Laurea Magistrale

Progetto 1 alternativo

Luca Colombo (885900) | Matteo Comi (886035)

Anno Accademico 2024/2025

Indice

Introduzione **1**

1 Struttura Della Libreria **3**

1.1 Utils 3

1.2 IterativeMethods 4

1.2.1 Metodo di Jacobi 4

1.2.2 Metodo di Gauss-Seidel 5

1.2.3 Metodo del Gradiente 5

1.2.4 Metodo del Gradiente Coniugato 6

2 Risultati Sperimentali **7**

Introduzione

In questo progetto ho sviluppato una libreria in **Python** per risolvere sistemi lineari $Ax = b$ con matrici *simmetriche e definite positive* (SPD) mediante quattro metodi iterativi: Jacobi, Gauss–Seidel, Gradiente e Gradiente Coniugato. La libreria utilizza esclusivamente le strutture dati vettoriali/matriciali fornite da `NumPy`/`SciPy` senza ricorrere ai solver già implementati, come richiesto. Gli esperimenti sono stati condotti sulle matrici sparse `spa1`, `spa2`, `vem1`, `vem2` in formato `.mtx`, con vettore soluzione esatta $x = \mathbf{1}$, membro destro $b = Ax$ e tolleranze $10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}$.

Scelte Implementative

In questo progetto ho utilizzato le librerie **NumPy** e **SciPy** esclusivamente come *infrastruttura di dati* e di *I/O*, senza ricorrere ad alcun solver già implementato per la risoluzione di sistemi lineari. L'implementazione dei quattro metodi iterativi (Jacobi, Gauss–Seidel, Gradiente, Gradiente Coniugato) è interamente sviluppata a mano.

Di seguito riportiamo un'indagine preliminare svolta sulle matrici di cui ci andremo ad occupare.

	Dimensione	**# Elementi nulli**	**Condizionamento**	**Simmetrica**	**Definita positiva**	**Dominanza diagonale**
spa 1	1000 x 1000	182434	2048.15	Si	Si	No
spa 2	3000 x 3000	1633298	1411.97	Si	Si	No
vem 1	1681 x 1681	13385	324.64	Si	Si	No
vem 2	2601 x 2601	21225	507.02	Si	Si	No

Tabella 1: Caratteristiche delle matrici utilizzate. Con dominanza diagonale si intende quella stretta e per righe.

Al fine di garantire la possibilità di riprodurre gli esperimenti svolti, riportiamo di seguito le caratteristiche del sistema utilizzate per la realizzazione della libreria e l'esecuzione dei vari esperimenti ed indagini condotte:

- CPU: Intel Core i7-1255U
- RAM: 16 GB
- Sistema Operativo: Windows 11

- Python: versione 3.12.3

1. Struttura Della Libreria

Il repository è organizzato in diversi moduli:

La libreria è organizzata nel package `iterative_solver` con due sottomoduli principali:

- **iterative_methods**: implementa i metodi iterativi per la risoluzione di sistemi lineari.
- **utils**: raccoglie le funzioni di supporto per caricamento dati, preparazione variabili e gestione dei risultati.

Come già spiegato brevemente all'interno dell'introduzione, abbiamo utilizzato le librerie `NumPy`, `SciPy (sparse, io.mmread)`, `Matplotlib`, oltre a moduli standard `os`, `csv`, `time`.

La cartella `matrices/` contiene i file `.mtx` usati nei test (`spa1`, `spa2`, `vem1`, `vem2`); la cartella `results/` viene generata automaticamente per salvare output e grafici. Lo script `iterative_solver/test_matrices_folder.py` esegue l'intero flusso sui dataset.

1.1 Utils

Il modulo **utils** include le seguenti utilità:

- `matrix_loader.load_matrix(filepath)`: carica una matrice `.mtx` e la restituisce in formato `csr_matrix`.
- `setup_variable.setup_variable(A)`: costruisce il vettore $x_{\text{esatto}} = \mathbf{1}$ e $b = A\,x_{\text{esatto}}$.
- `print_results.print_results(...)`: stampa formattata di iterazioni, errore relativo, tempo e stato di convergenza.
- `results_saver.results_saver(risultati, matrix_name)`: salva i risultati in `results/{matrix_name}_results/output.csv`.
- `plot_results.plot_results(risultati, matrix_name)`: genera grafici *iterazioni vs tolleranza*, *errore vs tolleranza*, *tempo vs tolleranza* (PNG) nella stessa cartella dei risultati.

1.2 IterativeMethods

Nel modulo **iterative_methods** sono stati implementati i metodi iterativi per la risoluzione di sistemi lineari. In particolare, sono presenti le seguenti funzioni:

- `jacobi(A, b, x_true, tol, max_iter)`
- `gauss_seidel(A, b, x_true, tol, max_iter)`
- `gradient(A, b, x_true, tol, max_iter)`
- `conjugate_gradient(A, b, x_true, tol, max_iter)`

Tutti i metodi implementati utilizzano come criterio di arresto il confronto tra l'errore relativo e la tolleranza imposta dall'utente:

$$\frac{\|x^{(k)} - x_{\text{vero}}\|}{\|x_{\text{vero}}\|} \leq \text{tol}$$

con $x^{(k)}$ soluzione al passo k e x_{vero} soluzione esatta fissata pari al vettore unitario. È inoltre previsto un numero massimo di iterazioni (`max_iter`, default = 20000) per prevenire cicli infiniti. Tutti i metodi partono da una soluzione iniziale nulla:

$$x^{(0)} = (0, 0, \ldots, 0)^T$$

1.2.1 Metodo di Jacobi

Il metodo di Jacobi aggiorna la soluzione utilizzando soltanto i valori della diagonale della matrice A. La regola di aggiornamento è:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) \qquad i = 1, \ldots, n$$

dove a_{ii} è l'elemento diagonale della matrice A.

Controlli implementati:

- Diagonale non nulla: $a_{ii} \neq 0 \ \forall i$.
- Dominanza diagonale: $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$ per ogni riga i.
- Matrice sparsa: A è rappresentata come matrice sparsa (es. formato CSR/CSC).
- Matrice quadrata: $A \in \mathbb{R}^{n \times n}$.

- Dimensioni incoerenti: verifica di coerenza $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ (e variabili di dimensioni compatibili).

1.2.2 Metodo di Gauss-Seidel

Il metodo di Gauss-Seidel sfrutta i valori già aggiornati nella stessa iterazione. La regola di aggiornamento è:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) \qquad i = 1, \ldots, n$$

Questo lo rende, in generale, più veloce del metodo di Jacobi, sebbene la condizione di convergenza resti legata alla dominanza diagonale o a proprietà di simmetria e definitezza positiva della matrice. Per quanto riguarda i **Controlli implementati** sono gli stessi di Jacobi.

1.2.3 Metodo del Gradiente

Il metodo del gradiente (o discesa ripida) si basa sull'aggiornamento lungo la direzione del residuo $r^{(k)}$:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} r^{(k)}$$

dove il passo $\alpha^{(k)}$ viene calcolato come:

$$\alpha^{(k)} = \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle r^{(k)}, A r^{(k)} \rangle}$$

con $r^{(k)} = b - Ax^{(k)}$.

Controlli implementati:

- A sia una matrice simmetrica;
- A sia definita positiva;
- Matrice sparsa: A è rappresentata come matrice sparsa (es. formato CSR/CSC).
- Matrice quadrata: $A \in \mathbb{R}^{n\times n}$.
- Dimensioni incoerenti: verifica di coerenza $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ (e variabili di dimensioni compatibili).

1.2.4 Metodo del Gradiente Coniugato

Il metodo del gradiente coniugato migliora il gradiente semplice evitando il fenomeno di zig-zag. La regola di aggiornamento è:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$$

dove la direzione $d^{(k)}$ è aggiornata come:

$$d^{(k)} = r^{(k)} + \beta^{(k-1)} d^{(k-1)}$$

e i coefficienti sono calcolati come:

$$\alpha^{(k)} = \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle d^{(k)}, A d^{(k)} \rangle}, \qquad \beta^{(k)} = \frac{\langle r^{(k+1)}, r^{(k+1)} \rangle}{\langle r^{(k)}, r^{(k)} \rangle}$$

con $r^{(k)} = b - Ax^{(k)}$.

Controlli implementati:

- Matrice sparsa: A è rappresentata come matrice sparsa (es. formato CSR/CSC).
- Matrice quadrata: $A \in \mathbb{R}^{n \times n}$.
- Dimensioni incoerenti: verifica di coerenza $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ (e variabili di dimensioni compatibili).

Le quattro implementazioni forniscono un quadro completo dei principali metodi iterativi utilizzati in ambito scientifico per la risoluzione di sistemi lineari di grandi dimensioni, specialmente nel caso di matrici sparse.

2. Risultati Sperimentali

Dopo aver implementato i quattro metodi iterativi descritti in precedenza, si è proceduto a valutarne le prestazioni su un insieme di matrici sparse fornite in formato `.mtx`. In particolare, le matrici utilizzate nei test sono `spa1`, `spa2`, `vem1` e `vem2`, le cui caratteristiche strutturali sono state riportate nella sezione di introduzione.

Per ciascuna matrice è stato costruito il termine noto b come

$$b = A\,x_{\text{vero}},$$

dove x_{vero} è un vettore costituito esclusivamente da 1. In questo modo, la soluzione esatta è nota a priori e può essere utilizzata per il calcolo dell'errore relativo.

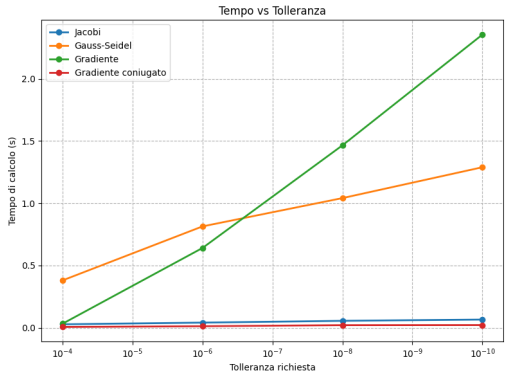
Gli esperimenti sono stati condotti variando la tolleranza di arresto tra diversi valori (10^{-4}, 10^{-6}, 10^{-8} e 10^{-10}). Per ogni configurazione, sono stati registrati:

- il numero di iterazioni necessarie a raggiungere la convergenza;
- l'errore relativo finale rispetto alla soluzione esatta;
- il tempo di esecuzione.

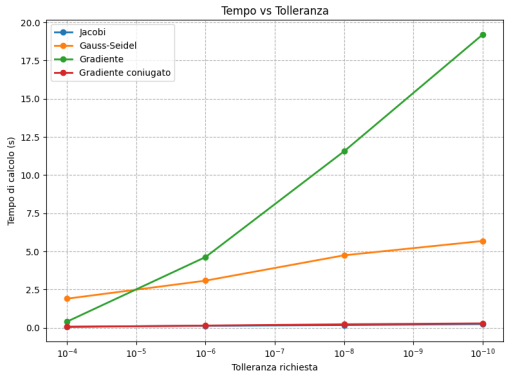
I risultati vengono salvati automaticamente in file `.csv` all'interno della cartella `results`, insieme ai grafici che mostrano l'andamento di tempo, errore e numero di iterazioni in funzione della tolleranza. Tali grafici permettono un confronto diretto tra i diversi metodi implementati.

L'obiettivo dell'analisi è osservare le differenze di comportamento tra Jacobi, Gauss-Seidel, Gradiente e Gradiente Coniugato al variare della tolleranza e della tipologia di matrice, evidenziando i metodi più efficienti in termini di velocità e accuratezza.

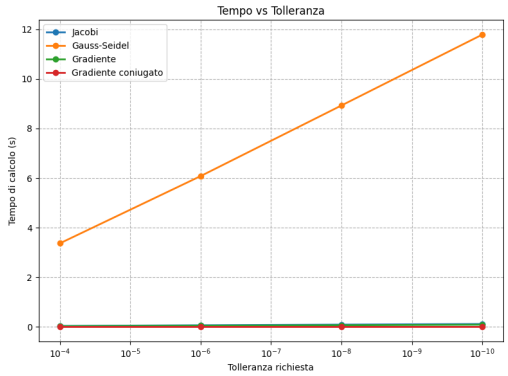
Ci siamo occupati di eseguire i metodi per ciascuna matrice e livello di tolleranza, per poi valutare i risultati ottenuti. Innanzitutto, dopo aver ottenuto i dati, ci siamo occupati di generare una visualizzazione mediante dei grafici, in particolare ci siamo occupati di realizzare ed analizzare grafici per quanto concerne i tempi di esecuzione, il numero di iterazioni, l'errore in funzione della tolleranza per ciascuna matrice, così facendo siamo in grado di confrontare i diversi metodi in maniera efficace.



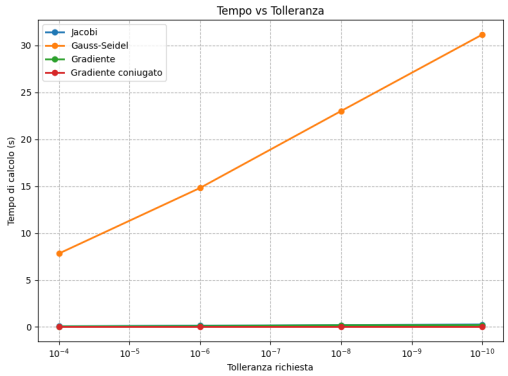
(a) Matrice spa1



(b) Matrice spa2



(c) Matrice vem1



(d) Matrice vem2

Figura 2.1: Confronto dei tempi di esecuzione (in secondi) dei metodi iterativi Jacobi, Gauss-Seidel, Gradiente e Gradiente Coniugato al variare della tolleranza richiesta, per le quattro matrici di test: (a) `spa1`, (b) `spa2`, (c) `vem1`, (d) `vem2`.

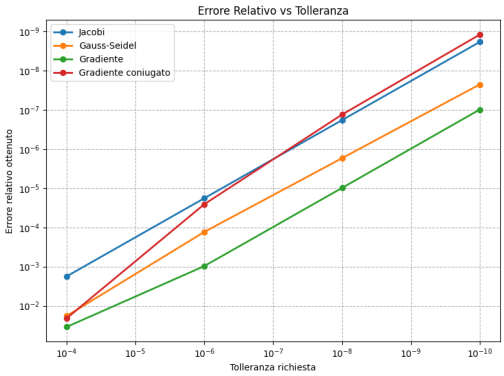
Riportiamo di seguito i risultati ottenuti dei metodi sulle varie matrici a disposizione.

Nei grafici presenti in figura 2.1 possiamo osservare come per tutte le diverse implementazioni il tempo di esecuzione aumenti man mano che la tolleranza si avvicina allo zero. Questo è quello che ci saremmo potuti aspettare, infatti, si tratta di un comportamento piuttosto prevedibile, in quanto il numero di iterazioni necessarie per raggiungere la tolleranza aumenta, come possiamo osservare all'interno delle tabelle 2.1, 2.2, 2.3 e 2.4.

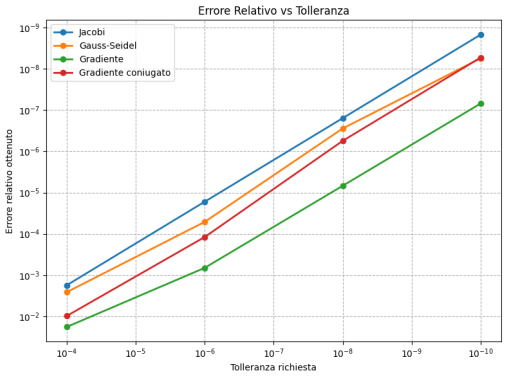
Confrontando i vari metodi iterativi in base al tempo, quello di Gauß-Seidel risulta essere il più lento per quanto riguarda le matrici vem1 e vem2, questo anche se il numero di iterazioni è inferiore se confrontato con quello di altri metodi, è possibile osservare questo fenomeno all'interno delle tabella appena riportate.

Questo è tendenzialmente dovuto dal fatto che stiamo sostanzialmente richiedendo di risolvere un sistema lineare al solo scopo di aggiornare la soluzione attuale. Per quanto riguarda invece, le matrici spa1 e spa 2, il metodo che richiede più tempo è quello del gradiente. La spiegazione di questo fenomeno potrebbe essere dovuta all'andamento a zig zag della convergenza del metodo preso in esame. Questo ci viene anche confermato dai numeri di condizionamento, infatti quello di spa1 è pari a 2048 mentre quello di spa2 a 1411, quindi potrebbe essere che combinando un numero di condizionamento elevato e il punto di partenza scelto per il metodo, ovvero il vettore nullo, abbiamo ottenuto un andamento a zig-zag nella convergenza. Per quanto concerne invece il metodo di Jacobi, esso risulta il più veloce per le matrici spa1 e spa2, mentre per le matrici vem1 e vem2 il più rapido è il gradiente coniugato. Ciò accade principalmente a causa del numero di condizionamento che essendo alto per le matrici spa1 e spa2, rallenta l'esecuzione dei metodi che si basano sul gradiente.

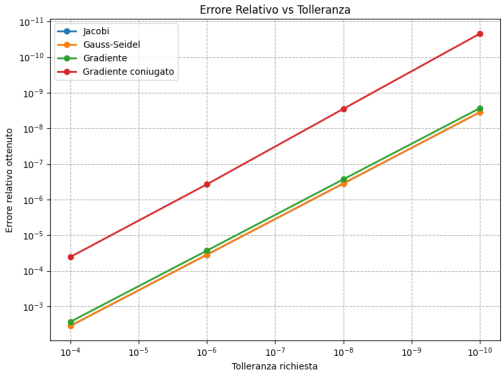
All'interno dei grafici in figura 2.2 è possibile osservare un andamento che decresce man mano che la tolleranza si avvicina allo zero, ciò è legato al fatto che i metodi implementati convergono prima di raggiungere il numero massimo di iterazioni (pari a 20000); inoltre, non notiamo una differenza significativa tra i metodi implementati in termini di errore ottenuto.



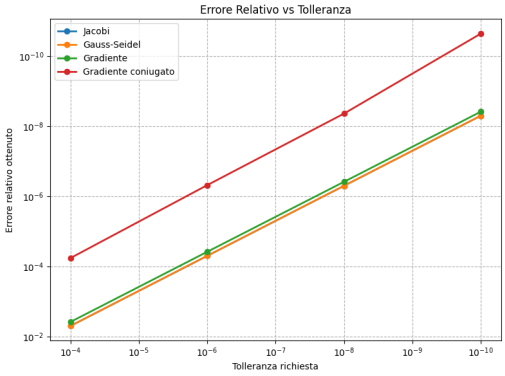
(a) Matrice spa1



(b) Matrice spa2



(c) Matrice vem1



(d) Matrice vem2

Figura 2.2: Confronto dell'errore relativo al variare della tolleranza richiesta per i metodi iterativi Jacobi, Gauss-Seidel, Gradiente e Gradiente Coniugato, applicati alle quattro matrici di test: (a) `spa1`, (b) `spa2`, (c) `vem1`, (d) `vem2`.

Tolleranza	Jacobi			Gauss-Seidel			Gradiente			Gradiente Coniugato		
	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.
1e-4	0.00177	0.0377	115	0.01821	0.4353	9	0.03457	0.0290	143	0.02079	0.0054	49
1e-6	1.80e-05	0.0512	181	1.30e-04	0.8171	17	9.68e-04	0.6686	3577	2.55e-05	0.0169	134
1e-8	1.82e-07	0.0744	247	1.71e-06	1.0748	24	9.82e-06	1.5265	8233	1.32e-07	0.0198	177
1e-10	1.85e-09	0.0865	313	2.25e-08	1.4281	31	9.82e-08	2.4340	12919	1.20e-09	0.0218	200

Tabella 2.1: Confronto dei metodi iterativi per diverse tolleranze per la matrice spa1.mtx

Tolleranza	Jacobi			Gauss-Seidel			Gradiente			Gradiente Coniugato		
	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.
1e-4	0.00177	0.1061	36	2.6e-03	2.0391	5	1.81e-02	4.7772	161	9.82e-03	0.1381	42
1e-6	1.67e-05	0.1826	57	5.14e-05	3.2024	8	6.69e-04	4.7772	1949	1.20e-04	0.3530	122
1e-8	1.57e-07	0.2055	78	2.79e-07	5.6155	12	6.87e-06	12.4772	5087	5.59e-07	0.6756	196
1e-10	1.48e-09	0.3092	99	5.57e-09	6.3975	15	6.94e-08	22.0696	8285	5.32e-09	0.7916	240

Tabella 2.2: Confronto dei metodi iterativi per diverse tolleranze per la matrice spa2.mtx

Tolleranza	Jacobi			Gauss-Seidel			Gradiente			Gradiente Coniugato		
	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.
1e-4	3.54e-3	0.1607	1314	3.51e-3	8.1703	659	2.70e-3	0.0348	890	4.08e-5	0.0011	38
1e-6	3.54e-5	0.2299	2433	3.53e-5	23.1884	1218	2.71e-5	0.0553	1612	3.73e-7	0.0009	45
1e-8	3.54e-7	0.3236	3552	3.52e-7	19.1330	1778	2.70e-7	0.0790	2336	2.83e-9	0.0009	53
1e-10	3.54e-9	0.2873	4671	3.51e-9	36.4854	2338	2.71e-9	0.0955	3058	2.19e-11	0.0009	59

Tabella 2.3: Confronto dei metodi iterativi per diverse tolleranze per la matrice vem1.mtx

Tolleranza	Jacobi			Gauss-Seidel			Gradiente			Gradiente Coniugato		
	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.	Rel.Err.	T (s)	Iter.
1e-4	4.97e-3	0.0718	1927	4.95e-3	8.5992	965	3.81e-3	0.0583	1308	5.73e-5	0.0023	47
1e-6	4.97e-5	0.1353	3676	4.94e-5	35.9781	1840	3.79e-5	0.1068	2438	4.74e-7	0.0020	56
1e-8	4.97e-7	0.1944	5425	4.96e-7	43.0832	2714	3.81e-7	0.1470	3566	4.30e-9	0.0020	66
1e-10	4.96e-9	0.2728	7174	4.95e-9	43.9140	3589	3.80e-09	0.1910	4696	2.25e-11	0.0030	74

Tabella 2.4: Confronto dei metodi iterativi per diverse tolleranze per la matrice vem2.mtx