

UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

METODI DEL CALCOLO SCIENTIFICO

PROGETTO 1 (ALTERNATIVO)

Mini Libreria per sistemi lineari

Autori:

Mattia Ingrassia - 879204 - m.ingrassia3@campus.unimib.it

Riccardo Ghilotti - 879259 - r.ghilotti@campus.unimib.it

Appello di Giugno 2025



Indice

1	Introduzione	1
1.1	Richieste sulla libreria	1
2	Struttura del progetto	2
2.1	Architettura	2
2.1.1	Jacobi	3
2.1.2	Gauss-Seidel	4
2.1.3	Gradiente	4
2.1.4	Gradiente Coniugato	4
2.2	Interfaccia	5
3	Analisi dei risultati	6
3.1	Spa1.mtx	6
3.1.1	Distribuzione	6
3.1.2	Risultati	7
3.1.3	Analisi dei risultati	9
3.2	Spa2.mtx	10
3.2.1	Distribuzione	10
3.2.2	Risultati	10
3.2.3	Analisi dei risultati	12
3.3	Vem1.mtx	12
3.3.1	Distribuzione	12
3.3.2	Risultati	13
3.3.3	Analisi dei risultati	15
3.4	Vem2.mtx	15
3.4.1	Distribuzione	15
3.4.2	Risultati	16
3.4.3	Analisi dei risultati	18
4	Conclusioni	18
	Bibliografia	19

Elenco delle figure

1	Esempio di esecuzione su vem2.mtx con tolleranza 1e-06	5
2	Distribuzione degli elementi di Spa1	7
3	Variazione dell'errore relativo dei vari metodi su Spa1 al variare della tolleranza.	8
4	Variazione del numero di iterazioni dei vari metodi su Spa1 al variare della tolleranza.	8
5	Variazione del tempo di esecuzione dei vari metodi su Spa1 al variare della tolleranza.	9
6	Distribuzione degli elementi di Spa2	10
7	Variazione dell'errore relativo dei vari metodi su Spa2 al variare della tolleranza.	11
8	Variazione del numero di iterazioni dei vari metodi su Spa2 al variare della tolleranza.	11

9	Variazione del tempo di esecuzione dei vari metodi su Spa2 al variare della tolleranza.	12
10	Distribuzione degli elementi di Vem1	13
11	Variazione dell'errore relativo dei vari metodi su Vem1 al variare della tolleranza.	14
12	Variazione del numero di iterazioni dei vari metodi su Vem1 al variare della tolleranza.	14
13	Variazione del tempo di esecuzione dei vari metodi su Vem1 al variare della tolleranza.	15
14	Distribuzione degli elementi di Vem2	16
15	Variazione dell'errore relativo dei vari metodi su Vem2 al variare della tolleranza.	17
16	Variazione del numero di iterazioni dei vari metodi su Vem2 al variare della tolleranza.	17
17	Variazione del tempo di esecuzione dei vari metodi su Vem2 al variare della tolleranza.	18

Elenco delle tabelle

1	Risultati per Spa1 con tolleranza 10^{-4}	7
2	Risultati per Spa1 con tolleranza 10^{-6}	7
3	Risultati per Spa1 con tolleranza 10^{-8}	7
4	Risultati per Spa1 con tolleranza 10^{-10}	7
5	Risultati per Spa2 con tolleranza 10^{-4}	10
6	Risultati per Spa2 con tolleranza 10^{-6}	10
7	Risultati per Spa2 con tolleranza 10^{-8}	11
8	Risultati per Spa2 con tolleranza 10^{-10}	11
9	Risultati per Vem1 con tolleranza 10^{-4}	13
10	Risultati per Vem1 con tolleranza 10^{-6}	13
11	Risultati per Vem1 con tolleranza 10^{-8}	13
12	Risultati per Vem1 con tolleranza 10^{-10}	13
13	Risultati per Vem2 con tolleranza 10^{-4}	16
14	Risultati per Vem2 con tolleranza 10^{-6}	16
15	Risultati per Vem2 con tolleranza 10^{-8}	16
16	Risultati per Vem2 con tolleranza 10^{-10}	16

1 Introduzione

Il progetto prevede lo sviluppo di una mini libreria che implementi dei metodi iterativi per la risoluzione di sistemi lineari, in particolare per casi di matrici simmetriche e definite positive. Tra le richieste, è sottolineata l'importanza dello sviluppo della libreria in un ambiente open source, utilizzando un'architettura ben strutturata.

Nello specifico, i metodi iterativi da implementare sono i seguenti:

- metodo di Jacobi
- metodo di Gauss-Seidel
- metodo del Gradiente
- metodo del Gradiente coniugato

Le implementazioni di tali metodi sono illustrate meglio nella sezione a loro dedicata.

1.1 Richieste sulla libreria

La libreria sviluppata deve:

- Operare con ognuno dei metodi iterativi sopra menzionati, senza utilizzare implementazioni già esistenti.
- I metodi iterativi devono partire dal vettore iniziale nullo, ovvero da un vettore che contenga solo zeri, ed arrestarsi qualora la k-esima iterata $x^{(k)}$ soddisfi la seguente equazione

$$\frac{\|Ax^{(k)} - b\|}{\|b\|} < tol \quad (1)$$

dove tol è la tolleranza assegnata dall'utente. Inoltre, deve anche arrestarsi qualora venga raggiunto un numero massimo di iterazioni prestabilito, che deve essere un numero molto elevato (almeno 20000). Nel caso in cui si raggiunga questo numero di iterazioni, bisogna specificare di non essere giunti a convergenza nel numero di iterazioni disponibili e terminare l'esecuzione di tale metodo.

- Il codice deve avere il formato di un eseguibile che, presi come input:
 - una matrice A simmetrica e definita positiva,
 - un membro destro b,
 - un vettore soluzione esatta x,
 - una tolleranza tol,

applichi i 4 metodi implementati e riporti su schermo le informazioni sui risultati ottenuti: errore relativo tra la x esatta e la soluzione computata dal metodo, numero di iterazioni richieste e tempo di calcolo.


2 Struttura del progetto

Per lo sviluppo della libreria è stato utilizzato il linguaggio Python (🐍), utilizzando delle librerie open source ottimizzate per la gestione delle varie task:

- NumPy [1] per la gestione di vettori, matrici e operazioni tra di essi,
- SciPy [2] per leggere le matrici in formato .mtx e trasformarle in formati adatti per l'esecuzione delle operazioni,
- Click [3] per realizzare una interfaccia a linea di comando che fosse intuitiva ed efficace
- Matplotlib [4] per realizzare i grafici utilizzati in questa relazione.

La libreria implementata presenta la seguente struttura:

```
linear_system_solver/  
├── solvers/  
│   ├── base_solver.py  
│   ├── conjugate_gradient.py  
│   ├── gauss_seidel.py  
│   ├── gradient.py  
│   └── jacobi.py  
├── cli_interface.py  
└── plot_generator.py
```

La repository è visualizzabile in maniera completa su GitHub al seguente link  - Linear System Solver.

2.1 Architettura

Per garantire una struttura ottimale della libreria, rendendola facilmente espandibile in caso di aggiunte e facilitando inoltre la modifica di eventuali funzionalità, abbiamo utilizzato una struttura modulare, separando le varie responsabilità delle classi.

Più specificamente, abbiamo definito una classe astratta contenente la struttura tipica di tutti i risolutori iterativi, contenuta nel file `base_solver.py`.

Questa classe contiene un metodo `solve()`, che prende in ingresso:

- **A**: La matrice da risolvere;
- **b**: Il vettore contenente i termini noti (RHS);
- **tol**: La tolleranza;
- **correct_x** (opzionale): La soluzione esatta del sistema, necessaria per calcolare l'errore;
- **max_iterations**(opzionale): numero massimo di iterazioni consentite, di default viene impostato a 20000;
- **verbose**: flag per decidere se stampare o meno degli aggiornamenti durante l'esecuzione del programma, default = true.

Questo metodo inizializza le variabili necessarie per la risoluzione dei vari sistemi, ed esegue in maniera ciclica le iterazioni necessarie finché non viene raggiunta almeno una delle condizioni di arresto.

Questa classe contiene inoltre diversi metodi che tornano utili ai vari risolutori, tra cui:

- `get_residue(A, b, x)`: restituisce il residuo $b - np.dot(A, x)$;
- `get_relative_error(x, correct_x)`: restituisce l'errore relativo $\frac{\|x - correct_x\|}{\|correct_x\|}$
- `check_stopping_criteria(A, b, x)`: Controlla se viene rispettato almeno uno dei criteri di arresto:
 1. `n_iterations > max_iterations`: termina l'esecuzione (convergenza non raggiunta),
 2. $\frac{\|get_residue(A, b, x)\|}{\|b\|} < tolerance$: termina l'esecuzione (convergenza raggiunta).
- `is_diagonally_dominant(A)`: Controlla che la matrice A sia a dominanza diagonale, utile per i metodi di Jacobi e Gauss-Seidel.
- `is_symmetric_positive_definite(A)`: Controlla che la matrice A sia Simmetrica e Definita Positiva (SPD).

Algorithm 1 Struttura del metodo `solve`

```

1: setup()                                ▷ Operazioni specifiche che cambiano a seconda del tipo di solver
2: initial_x = 0
3: validate_inputs(A, b, x)                ▷ Controlli base sui dati in input
4: n_iterations = 0
5: while check_stopping_criteria() do
6:   x = iterate(A, b, x)
7:   n_iterations = n_iterations + 1
8: end while

```

Il metodo `iterate` è un metodo astratto che viene sovrascritto in ogni istanza di risolutore che estende la classe base.

Per ogni metodo, abbiamo cercato di implementare una versione ottimale seguendo le nozioni apprese durante il corso.

2.1.1 Jacobi

Nella funzione di `setup` viene calcolata `D_inv = 1 / np.diag(A)`, ovvero la diagonale inversa della matrice A, che verrà utilizzata per il calcolo della soluzione.

Algorithm 2 Struttura del metodo `iterate - Jacobi`

```

1: x_new = x + D_inv * get_residue(A, b, x) ▷ Moltiplicazione elemento per elemento
2: return x_new

```

2.1.2 Gauss-Seidel

Nella funzione di `setup` viene calcolata `L = np.tril(A)`, ovvero la matrice triangolare inferiore, che verrà utilizzata per il calcolo della soluzione.

Algorithm 3 Struttura del metodo *iterate* - Gauss-Seidel

```
1: x_new = x + tril_matrix_solver(get_residue(A, b, x))
2: return x_new
```

Dove `tril_matrix_solver()` è un risolutore diretto di matrici triangolari inferiori, definito come segue:

Algorithm 4 `tril_matrix_solver(rhs)`

Sia N la dimensione del vettore delle soluzioni, sia $N \times N$ la dimensione della matrice

```
1: sol =  $\vec{0}$ 
2: sol[0] = rhs[0] / L[0][0]
3: for i = 1 to N-1 do
4:   b = rhs[i]
5:   u = L[i][i]
6:   sol[i] = (1/u) * (b - np.dot(L[i, :i], sol[:i]))
7: end for
8: return sol
```

2.1.3 Gradiente

Per questo metodo la sezione di `setup` è vuota, in quanto non è necessario calcolare nessun nuovo elemento.

Algorithm 5 Struttura del metodo *iterate* - Gradiente

```
1: residue = get_residue(A, b, x)
2:  $\alpha$  = np.dot(residue.T, residue) / np.dot(residue.T, np.dot(A, residue))
3: x_new = x + np.dot( $\alpha$ , residue)
4: return x_new
```

2.1.4 Gradiente Coniugato

Nella funzione di `setup` viene calcolata la direzione iniziale `direction = get_residue(A, b, initial_x)`, che corrisponde al residuo della x di partenza.

Algorithm 6 Struttura del metodo *iterate* - **Gradiente Coniguato**

N.B.: `direction` ha uno scope globale rispetto alle altre variabili.

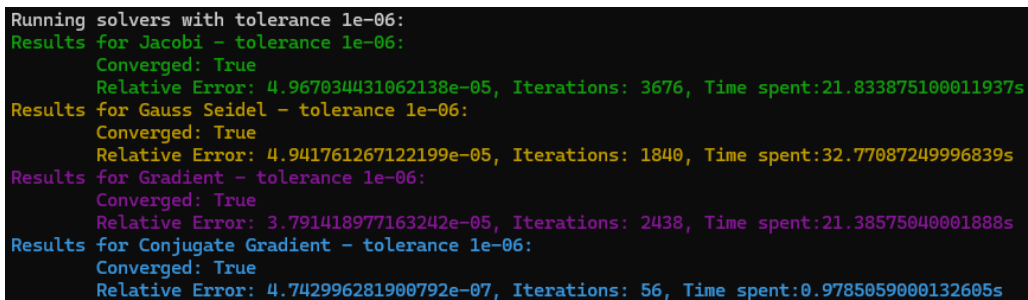
```
1: residue = get_residue(A, b, x)
2: alpha_beta_denominator = np.dot(direction.T, np.dot(A, direction))
3:  $\alpha$  = np.dot(direction.T, residue) / alpha_beta_denominator
4: x_new = x + np.dot(alpha, direction)
5: residue_new = get_residue(A, b, x_new)
6:  $\beta$  = np.dot(direction.T, np.dot(A, residue_new)) / alpha_beta_denominator
7: direction_new = residue_new - np.dot( $\beta$ , direction)
8: direction = direction_new
9: return x_new
```

2.2 Interfaccia

Per utilizzare la libreria è stata implementata un'interfaccia a linea di comando che prende come parametri:

- `--a_path`: il percorso del file contenente la matrice in formato `.mtx`
- `--b_path`[opzionale]: il percorso del file contenente il vettore `b` in formato `.csv` o `.txt`. Se non viene specificato, viene calcolato dal programma (`b = np.dot(A,x)`)
- `--x_path`[opzionale]: il percorso del file contenente il vettore `x` con la soluzione esatta in formato `.csv` o `.txt`. Se non viene specificato, viene utilizzato un vettore contenente tutti uno.
- `--tol`[opzionale]: uno o più numeri in virgola mobile che rappresentano le tolleranze utilizzate dal programma. Se non specificato, vengono utilizzate come tolleranze i seguenti valori: [`1e-4`, `1e-6`, `1e-8`, `1e-10`]
- `--verbose`[opzionale]: una variabile di flag, se `True` vengono stampati dei messaggi riguardanti l'esecuzione. `default=True`.

Il codice fa partire l'esecuzione dei vari risolutori stampando sul terminale i dettagli di esse.



```
Running solvers with tolerance 1e-06:
Results for Jacobi - tolerance 1e-06:
  Converged: True
  Relative Error: 4.967034431062138e-05, Iterations: 3676, Time spent:21.833875100011937s
Results for Gauss Seidel - tolerance 1e-06:
  Converged: True
  Relative Error: 4.941761267122199e-05, Iterations: 1840, Time spent:32.77087249996839s
Results for Gradient - tolerance 1e-06:
  Converged: True
  Relative Error: 3.791418977163242e-05, Iterations: 2438, Time spent:21.38575040001888s
Results for Conjugate Gradient - tolerance 1e-06:
  Converged: True
  Relative Error: 4.742996281900792e-07, Iterations: 56, Time spent:0.9785059000132605s
```

Figura 1: Esempio di esecuzione su `vem2.mtx` con tolleranza `1e-06`

3 Analisi dei risultati

Per effettuare test e analisi sono state utilizzate quattro matrici:

- Spa1: una matrice sparsa 1000×1000 .
- Spa2: una matrice sparsa 3000×3000 .
- Vem1: una matrice sparsa con elementi sulla diagonale ed altre due rette parallele alla diagonale 1681×1681 .
- Vem2: una matrice costruita come Vem1 di dimensione 2601×2601 .

Un' ulteriore informazione da aggiungere è che tutte le matrici sono simmetriche e definite positive (SPD), condizione necessaria per la convergenza del metodo del Gradiente e del Gradiente Coniugato.

I quattro metodi sono stati testati con:

1. Le matrici sopra indicate come matrice associata al sistema lineare $A \in \mathbb{R}^{N \times N}$.
2. La soluzione corretta $correct_x \in \mathbb{R}^N$, uguale ad un' array di 1.
3. il vettore dei termini noti b è calcolato nel seguente modo: $b = \text{np.dot}(A, x)$;
4. Quattro tolleranze diverse ($1e-4, 1e-6, 1e-8, 1e-10$).

3.1 Spa1.mtx

3.1.1 Distribuzione

La matrice è piuttosto sparsa, con elementi distribuiti irregolarmente su tutta la matrice ma con una concentrazione maggiore lungo la diagonale principale.

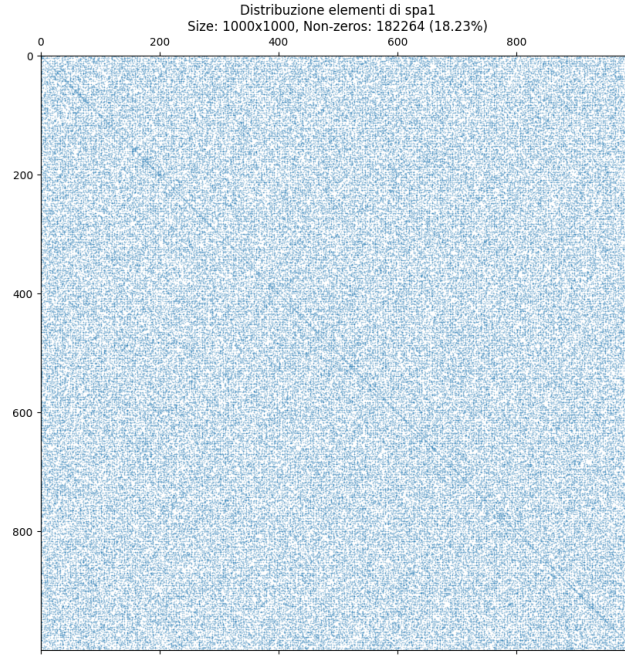


Figura 2: Distribuzione degli elementi di Spa1.
In azzurro sono evidenziate le entrate non nulle.

3.1.2 Risultati

In seguito sono riportati i risultati in forma tabellare e dei grafici che mostrano l'andamento delle metriche prese in considerazione.

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	1.771e-03	115	0.03406	Jacobi	1.798e-05	181	0.05287
Gauss Seidel	1.821e-02	9	0.01650	Gauss Seidel	1.300e-04	17	0.03036
Gradient	3.457e-02	143	0.06414	Gradient	9.680e-04	3577	1.49095
Conjugate Gradient	2.079e-02	49	0.03725	Conjugate Gradient	2.553e-05	134	0.09532

Tabella 1: Risultati per Spa1 con tolleranza 10^{-4}

Tabella 2: Risultati per Spa1 con tolleranza 10^{-6}

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	1.825e-07	247	0.07222	Jacobi	1.852e-09	313	0.09059
Gauss Seidel	1.710e-06	24	0.04140	Gauss Seidel	2.248e-08	31	0.05355
Gradient	9.816e-06	8233	3.33086	Gradient	9.820e-08	12919	5.26037
Conjugate Gradient	1.320e-07	177	0.11251	Conjugate Gradient	1.217e-09	200	0.14648

Tabella 3: Risultati per Spa1 con tolleranza 10^{-8}

Tabella 4: Risultati per Spa1 con tolleranza 10^{-10}

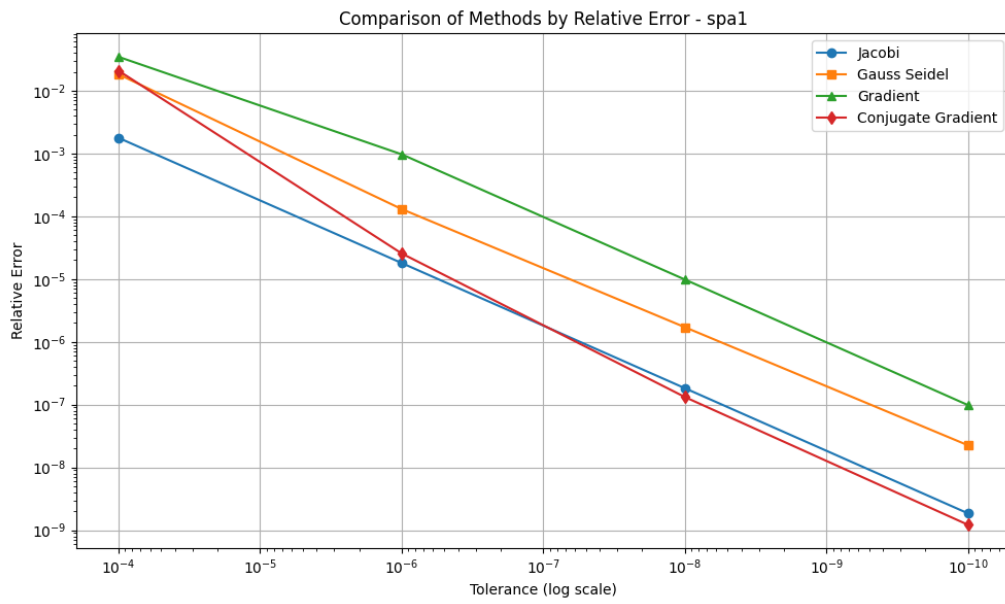


Figura 3: Variazione dell'errore relativo dei vari metodi su Spa1 al variare della tolleranza.

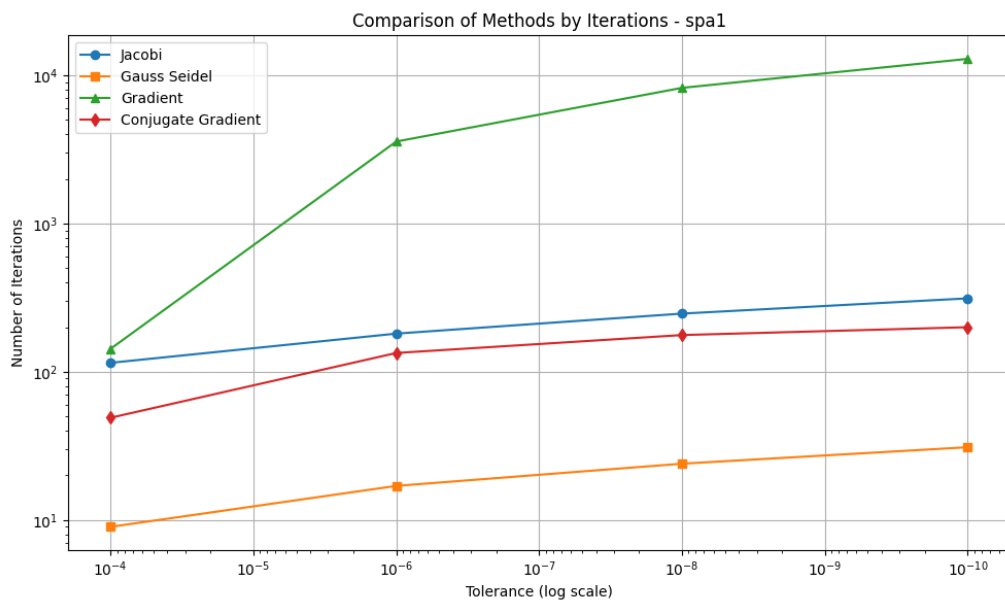


Figura 4: Variazione del numero di iterazioni dei vari metodi su Spa1 al variare della tolleranza.

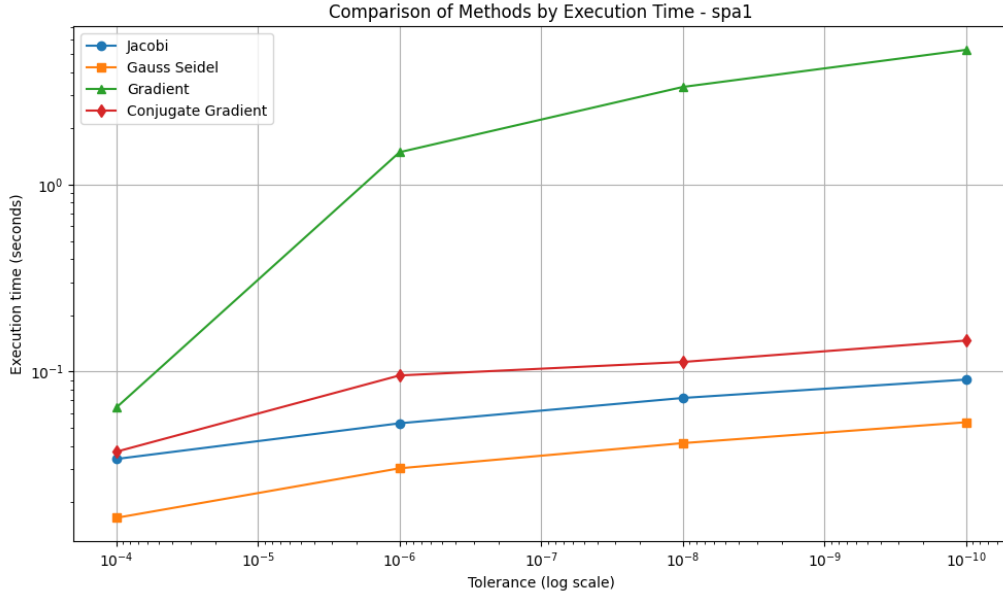


Figura 5: Variazione del tempo di esecuzione dei vari metodi su Spa1 al variare della tolleranza.

3.1.3 Analisi dei risultati

Dai grafici 3, 4 e 5 e dalle tabelle 1, 2, 3 e 4, si osserva che tutti i metodi convergono per la matrice Spa1.

Inoltre, come si può notare dai grafici 4 e 5, il metodo del gradiente effettua molte più iterazioni rispetto agli altri metodi e impiega una quantità di tempo molto più elevata per raggiungere la convergenza (con tolleranza $1e-10$ rispetto a Jacobi è 58 volte più lento, 98 volte più lento rispetto a Gauss-Seidel e 36 volte più lento rispetto al Gradiente Coniugato). Questo potrebbe dipendere dal fatto che il condizionamento della matrice influenza di molto la convergenza del metodo del gradiente. Infatti, dato che $\lambda_{max} \gg \lambda_{min}$ (per spa1 abbiamo $\lambda_{max} = 999.4558$ e $\lambda_{min} = 0.4880$), il numero di iterazioni è molto alto se il punto iniziale $x^{(0)}$ non si trova nella posizione adatta. La posizione "scomoda" di $x^{(0)}$ potrebbe far scaturire il fenomeno presentato a lezione come "convergenza a zig-zag", ovvero la situazione in cui al posto di andare direttamente verso il minimo della funzione ϕ , il metodo del Gradiente si sposta verso direzioni non ottimali, aumentando così il numero di iterazioni necessarie per raggiungere il punto di minimo di ϕ .

Il metodo di Jacobi ed il metodo del Gradiente Coniugato invece sono piuttosto competitivi l'uno con l'altro, con Jacobi che presenta una crescita lineare nel tempo assieme a una diminuzione altrettanto lineare dell'errore relativo.

Invece Gauss-Seidel ottiene performance ottime, facendo molte meno iterazioni ed impiegando la metà del tempo rispetto a Jacobi.

Le tabelle rinforzano quanto visto dai grafici, quindi si possono osservare i risultati scarsi del metodo del Gradiente e invece quelli ottimi di Gauss-Seidel per quanto riguarda le performance temporali. Per quanto riguarda l'errore, invece, Jacobi riesce ad ottenere l'errore minimo con 2 tolleranze su 4, per poi essere superato, di poco, dal Gradiente Coniugato.

Dunque, con matrici simili a Spa1, il metodo di Gauss-Seidel è sicuramente il più rapido, ma anche il metodo di Jacobi ed il metodo del Gradiente Coniugato ottengono una buona performance in termini di errore, a discapito di un tempo di esecuzione leggermente maggiore.

3.2 Spa2.mtx

3.2.1 Distribuzione

Anche questa matrice è piuttosto sparsa, con elementi distribuiti irregolarmente su tutta la matrice e con una concentrazione maggiore lungo la diagonale principale. Ha una densità simile a Spa1, ma presenta molte più entrate.

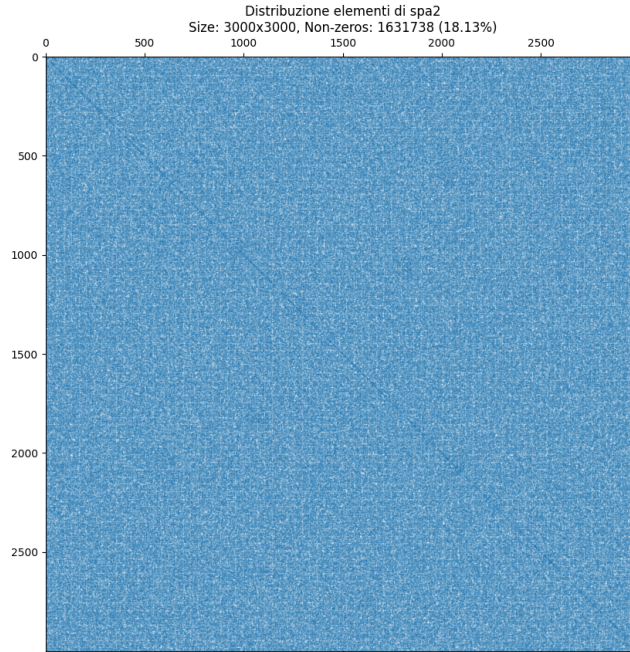


Figura 6: Distribuzione degli elementi di Spa2.
In azzurro sono evidenziate le entrate non nulle.

3.2.2 Risultati

In seguito sono riportati i risultati in forma tabellare e dei grafici che mostrano l'andamento delle metriche prese in considerazione.

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	1.766e-03	36	0.09461	Jacobi	1.667e-05	57	0.14196
Gauss Seidel	2.599e-03	5	0.04335	Gauss Seidel	5.142e-05	8	0.07078
Gradient	1.813e-02	161	0.59750	Gradient	6.694e-04	1949	7.08698
Conjugate Gradient	9.821e-03	42	0.26503	Conjugate Gradient	1.198e-04	122	0.73057

Tabella 5: Risultati per Spa2 con tolleranza 10^{-4}

Tabella 6: Risultati per Spa2 con tolleranza 10^{-6}

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	1.573e-07	78	0.19065	Jacobi	1.484e-09	99	0.24340
Gauss Seidel	2.794e-07	12	0.09771	Gauss Seidel	5.571e-09	15	0.12465
Gradient	6.865e-06	5087	18.28152	Gradient	6.938e-08	8285	29.63438
Conjugate Gradient	5.587e-07	196	1.20186	Conjugate Gradient	5.324e-09	240	1.43802

Tabella 7: Risultati per Spa2 con tolleranza 10^{-8}

Tabella 8: Risultati per Spa2 con tolleranza 10^{-10}

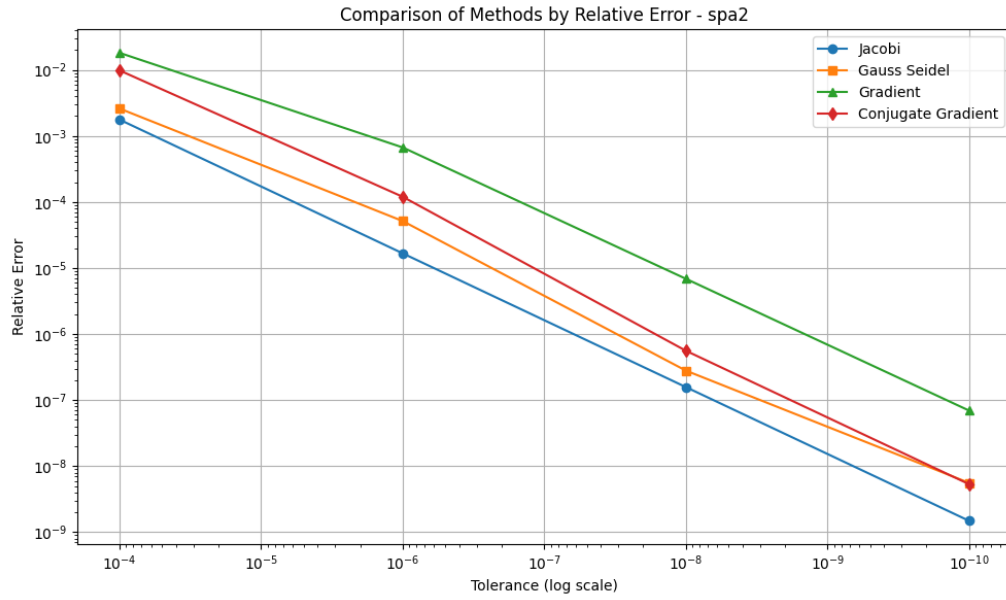


Figura 7: Variazione dell'errore relativo dei vari metodi su Spa2 al variare della tolleranza.

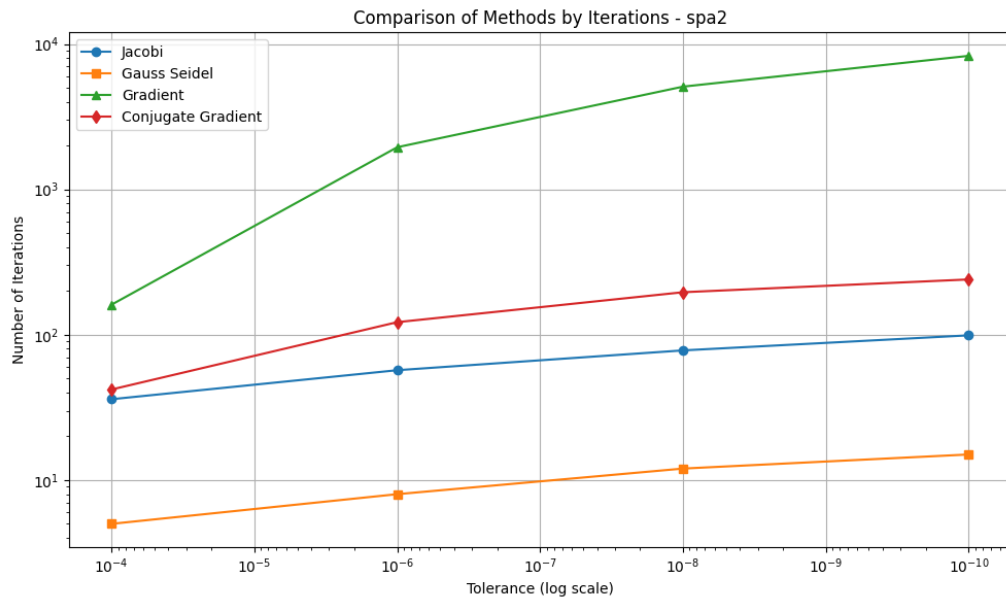


Figura 8: Variazione del numero di iterazioni dei vari metodi su Spa2 al variare della tolleranza.

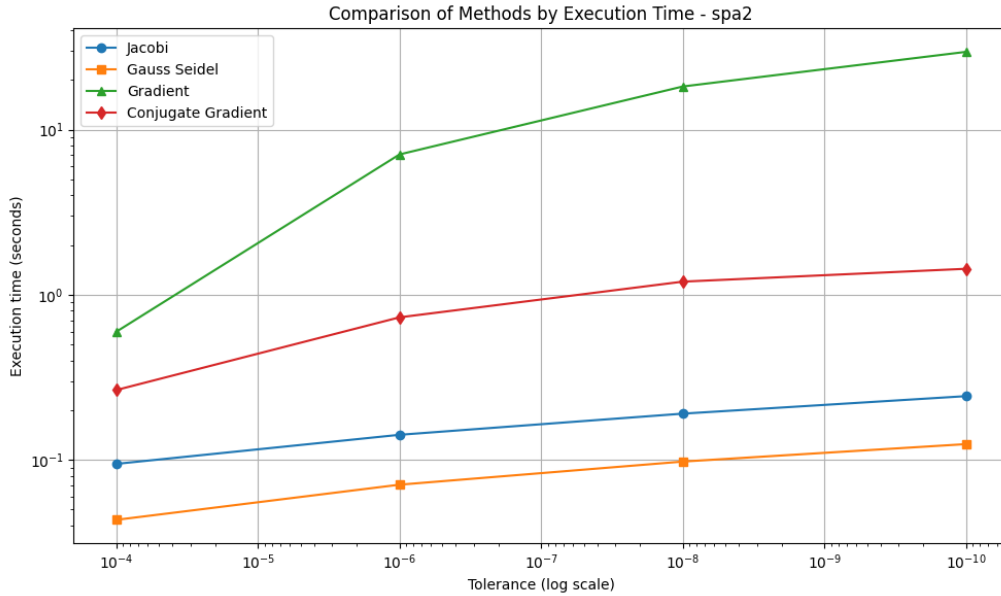


Figura 9: Variazione del tempo di esecuzione dei vari metodi su Spa2 al variare della tolleranza.

3.2.3 Analisi dei risultati

Dai grafici 7, 8 e 9 e dalle tabelle 5, 6, 7 e 8 si nota che le performance ottenute su Spa2 sono molto simili a quelle su Spa1, con qualche differenza.

Per prima cosa, il metodo del Gradiente non riesce ancora ad essere competitivo per via del tempo impiegato; dal numero di iterazioni si intuisce che il problema potrebbe essere simile a quello di Spa1, in cui $\lambda_{max} \gg \lambda_{min}$ (infatti $\lambda_{max} = 2998.3585$ e $\lambda_{min} = 2.1235$).

Jacobi, invece, riesce ad ottenere performance ottime per quanto riguarda l'errore relativo, raggiungendo il minimo con tutte le tolleranze, ma impiegandoci il doppio del tempo rispetto a Gauss-Seidel.

Quest'ultimo, infatti, ottiene ancora performance ottimali per quanto riguarda il numero di iterazioni e il tempo di esecuzione per raggiungere la convergenza.

Il metodo che ottiene performance molto inferiori su Spa2 rispetto a Spa1 è quello del Gradiente Coniugato, ciò lo si può notare dal fatto che ci mette molto più tempo per raggiungere la convergenza rispetto a Spa1, risultando in un metodo che è 5 volte più lento di Jacobi e 10 di Gauss-Seidel. Concludiamo che anche in questo caso il metodo di Gauss-Seidel è preferibile se si cerca un metodo che converga in fretta, mentre se si cerca di ottenere un errore minore potrebbe essere preferibile il metodo di Jacobi.

3.3 Vem1.mtx

3.3.1 Distribuzione

Questa matrice è molto sparsa, contiene quasi solo 0 (presenta una densità dello 0.47%) e gli elementi non nulli sono distribuiti principalmente sulla diagonale della matrice e su due "linee rette" parallele ad essa.

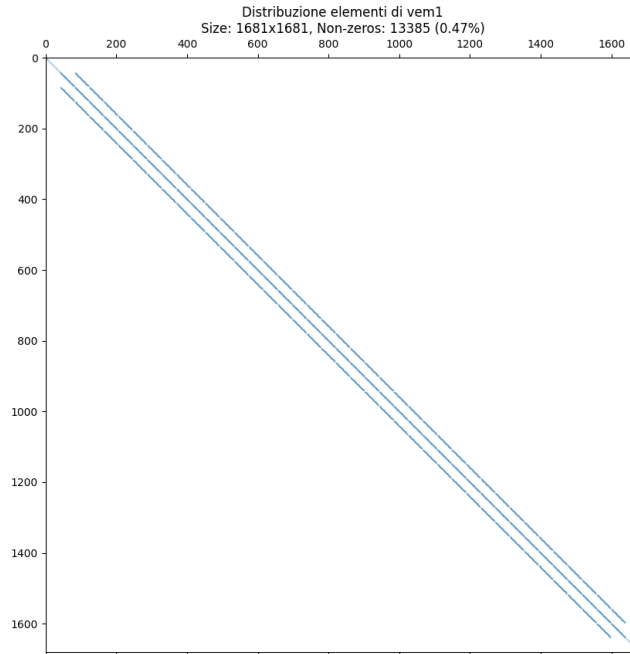


Figura 10: Distribuzione degli elementi di Vem1.
In azzurro sono evidenziate le entrate non nulle.

3.3.2 Risultati

In seguito sono riportati i risultati in forma tabellare e dei grafici che mostrano l'andamento delle metriche prese in considerazione.

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	3.540e-03	1314	0.51778	Jacobi	3.540e-05	2433	0.87876
Gauss Seidel	3.507e-03	659	2.05701	Gauss Seidel	3.527e-05	1218	3.75660
Gradient	2.705e-03	890	0.47772	Gradient	2.713e-05	1612	0.85105
Conjugate Gradient	4.083e-05	38	0.05373	Conjugate Gradient	3.732e-07	45	0.04256

Tabella 9: Risultati per Vem1 con tolleranza 10^{-4}

Tabella 10: Risultati per Vem1 con tolleranza 10^{-6}

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	3.540e-07	3552	1.25828	Jacobi	3.539e-09	4671	1.71313
Gauss Seidel	3.517e-07	1778	5.44983	Gauss Seidel	3.508e-09	2338	7.17882
Gradient	2.695e-07	2336	1.25575	Gradient	2.713e-09	3058	1.61016
Conjugate Gradient	2.832e-09	53	0.04954	Conjugate Gradient	2.192e-11	59	0.05616

Tabella 11: Risultati per Vem1 con tolleranza 10^{-8}

Tabella 12: Risultati per Vem1 con tolleranza 10^{-10}

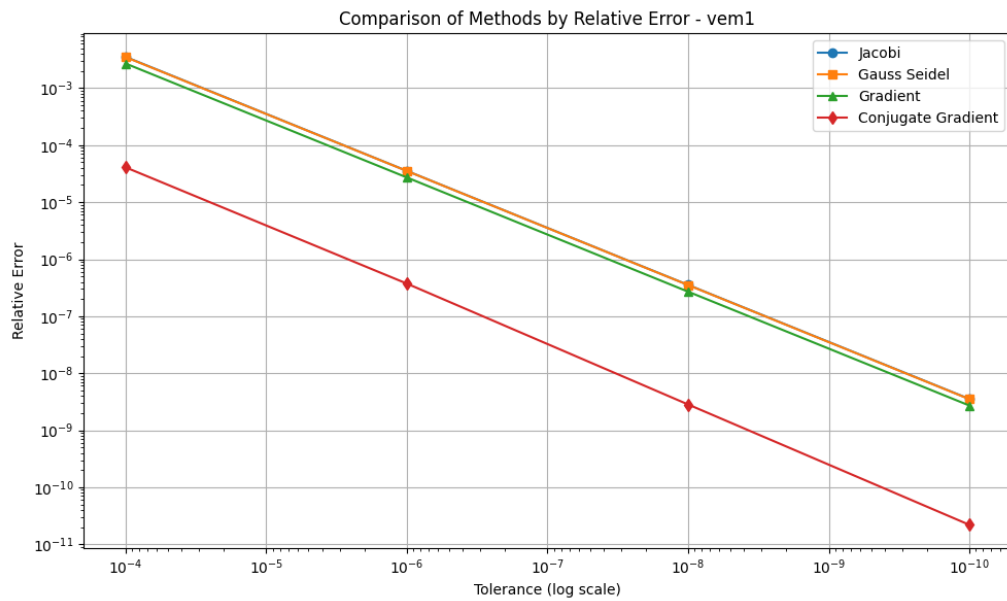


Figura 11: Variazione dell'errore relativo dei vari metodi su Vem1 al variare della tolleranza.

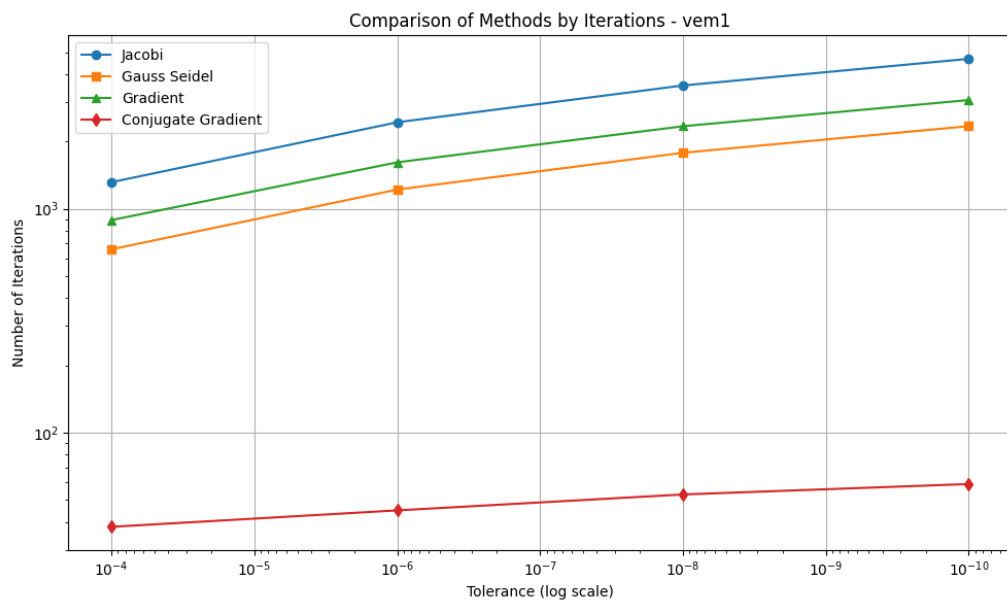


Figura 12: Variazione del numero di iterazioni dei vari metodi su Vem1 al variare della tolleranza.

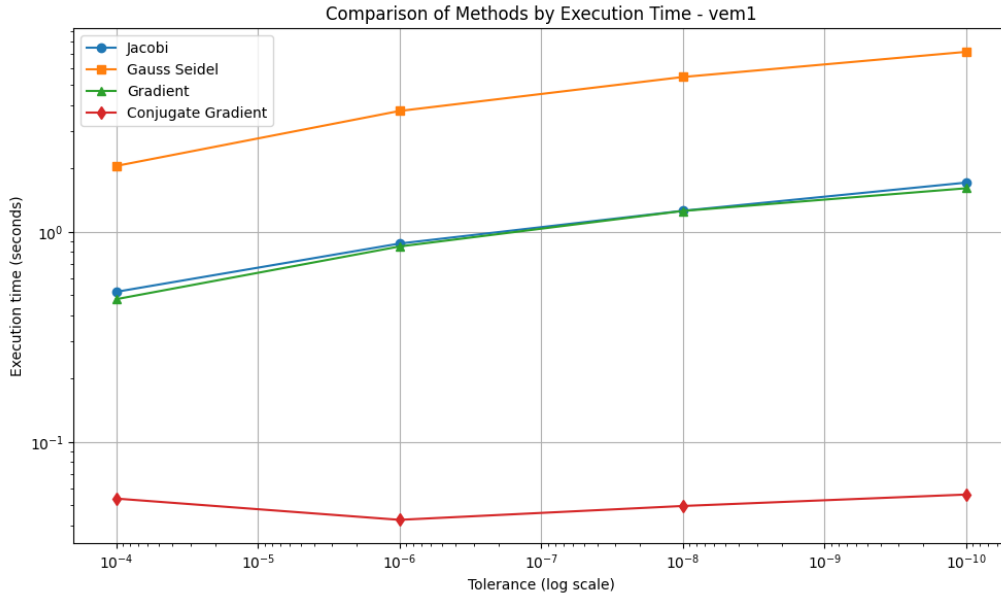


Figura 13: Variazione del tempo di esecuzione dei vari metodi su Vem1 al variare della tolleranza.

3.3.3 Analisi dei risultati

Dai grafici 11, 12 e 13 e dalle tabelle 9, 10, 11 e 12, si osserva che i metodi convergono tutti anche sulla matrice Vem1.

La prima cosa che si nota dai grafici 11, 12 e 13 è che il metodo del Gradiente Coniugato ottiene risultati nettamente migliori rispetto agli altri tre. Questo perché riesce più velocemente a trovare la direzione ottimale di discesa rispetto al metodo del gradiente permettendogli di arrivare immediatamente al punto di minimo di ϕ . Nonostante ciò, il metodo del gradiente non performa male rispetto a Gauss Seidel e Jacobi, anche se, pure in questo caso, $\lambda_{max} \gg \lambda_{min}$ (infatti $\lambda_{max} = 4.0000$ e $\lambda_{min} = 0.0123$). È inoltre molto singolare il fatto che il metodo del Gradiente e Jacobi ottengano performance molto simili in termini di tempo.

È interessante notare che l'andamento dell'errore relativo di Gauss-Seidel e Jacobi è pressoché identico, ciò accade perché quando si ha a che fare con matrici diagonali, i due metodi effettuano la medesima operazione durante le iterazioni.

Invece si nota che c'è un gran peggioramento nelle performance ottenute da Gauss-Seidel rispetto alle matrici Spa1 e Spa2, in quanto queste sono peggiori sia per quanto riguarda il tempo di convergenza che per l'errore ottenuto.

Date queste osservazioni, il metodo del Gradiente Coniugato è sicuramente il migliore.

3.4 Vem2.mtx

3.4.1 Distribuzione

Questa matrice è molto sparsa, contiene quasi solo 0 (presenta una densità dello 0.31%) e gli elementi non nulli sono distribuiti principalmente sulla diagonale della matrice e su due "linee rette" parallele ad essa. È simile a Vem1, ma presenta molte più entrate.

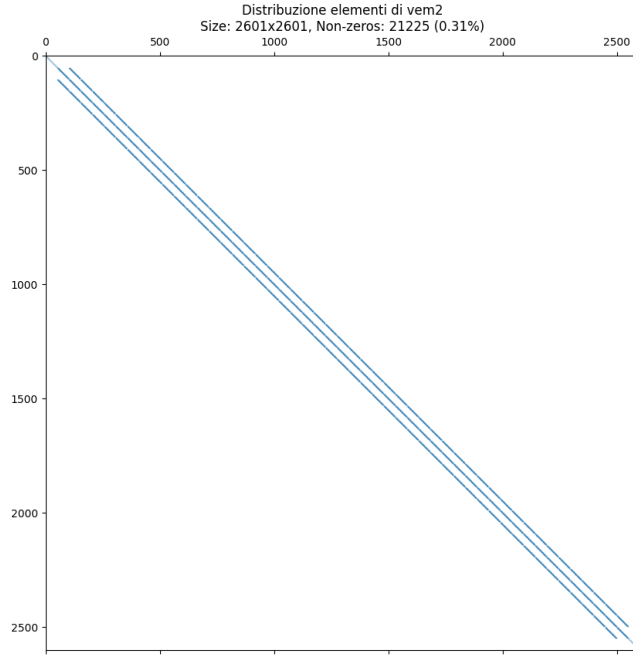


Figura 14: Distribuzione degli elementi di Vem2.
In azzurro sono evidenziate le entrate non nulle.

3.4.2 Risultati

In seguito sono riportati i risultati in forma tabellare e dei grafici che mostrano l'andamento delle metriche prese in considerazione.

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	4.968e-03	1927	4.34849	Jacobi	4.967e-05	3676	21.83388
Gauss Seidel	4.951e-03	965	17.86111	Gauss Seidel	4.942e-05	1840	32.77087
Gradient	3.812e-03	1308	11.29242	Gradient	3.791e-05	2438	21.38575
Conjugate Gradient	5.729e-05	47	0.71442	Conjugate Gradient	4.743e-07	56	0.97851

Tabella 13: Risultati per Vem2 con tolleranza 10^{-4}

Tabella 14: Risultati per Vem2 con tolleranza 10^{-6}

Metodo	Errore Relativo	Iterazioni	Tempo (s)	Metodo	Errore Relativo	Iterazioni	Tempo (s)
Jacobi	4.966e-07	5425	30.47507	Jacobi	4.964e-09	7174	40.77568
Gauss Seidel	4.958e-07	2714	49.19241	Gauss Seidel	4.949e-09	3589	61.83732
Gradient	3.810e-07	3566	30.07133	Gradient	3.799e-09	4696	39.54824
Conjugate Gradient	4.300e-09	66	0.73730	Conjugate Gradient	2.248e-11	74	1.05656

Tabella 15: Risultati per Vem2 con tolleranza 10^{-8}

Tabella 16: Risultati per Vem2 con tolleranza 10^{-10}

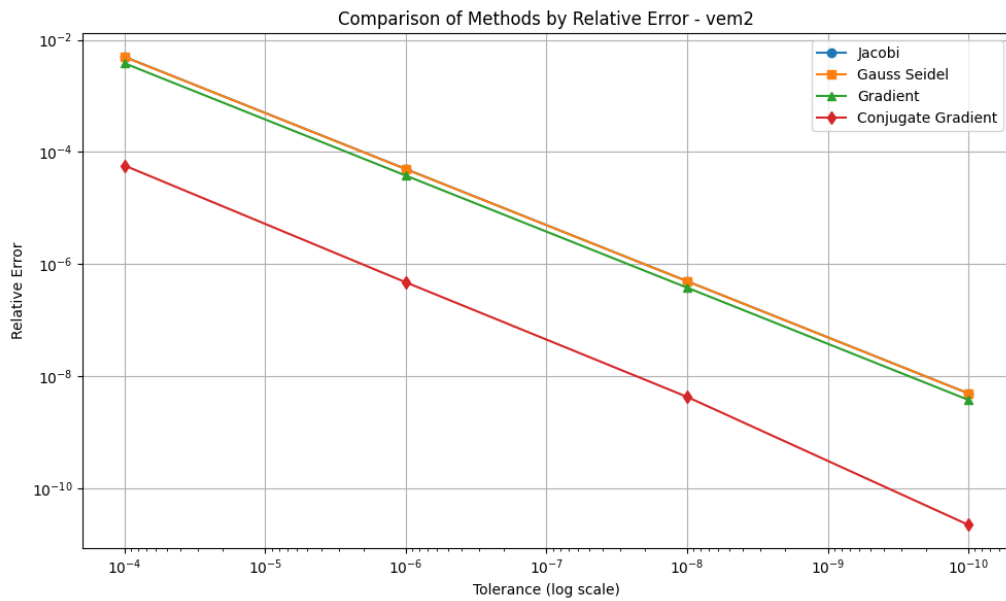


Figura 15: Variazione dell'errore relativo dei vari metodi su Vem2 al variare della tolleranza.

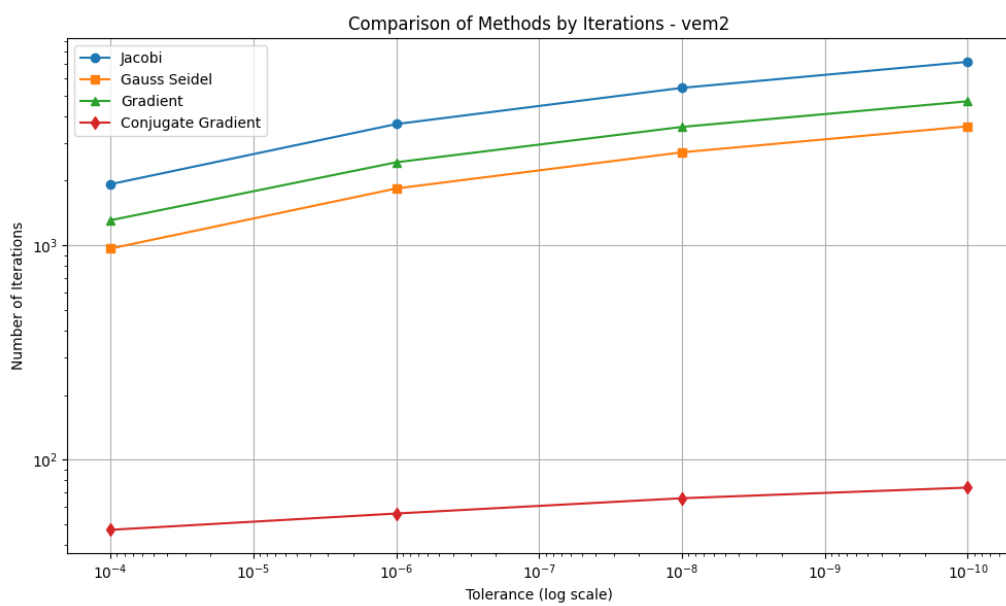


Figura 16: Variazione del numero di iterazioni dei vari metodi su Vem2 al variare della tolleranza.

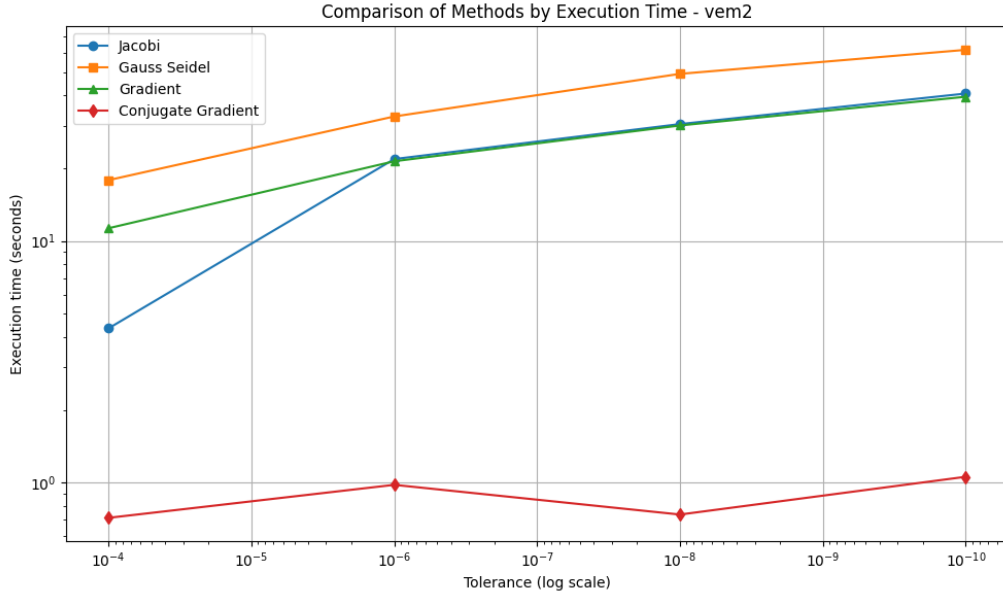


Figura 17: Variazione del tempo di esecuzione dei vari metodi su Vem2 al variare della tolleranza.

3.4.3 Analisi dei risultati

Come per Spa1 e Spa2, per Vem1 e Vem2 si possono fare più o meno le stesse osservazioni. Anche in questo caso $\lambda_{max} \gg \lambda_{min}$ (infatti $\lambda_{max} = 4.0000$ e $\lambda_{min} = 0.0079$), nonostante il metodo del Gradiente ottenga performance comparabili a Jacobi e Gauss-Seidel. Infine il metodo del Gradiente Coniugato si dimostra ancora superiore a tutti gli altri, per questo motivo non c'è metodo migliore per matrici simili a Vem1 e Vem2.

4 Conclusioni

Dalle analisi possiamo notare che per matrici sparse distribuite come Spa1 e Spa2 i metodi migliori che possiamo utilizzare sono quello di Jacobi e quello di Gauss-Seidel, a seconda delle nostre necessità. Invece, per quanto riguarda Vem1 e Vem2 il metodo che performa meglio tra tutti è quello del Gradiente Coniugato.

Si può spezzare una lancia in favore del metodo del Gradiente perchè in tutte le matrici $\lambda_{max} \gg \lambda_{min}$, rendendo probabile la possibilità che si verifichi la "convergenza a Zig-Zag", in questo caso si potrebbero provare a modificare le matrici tramite pre-condizionamento in modo tale da diminuire il rapporto tra gli autovalori, con l'obiettivo di avvicinarsi a $\frac{|\lambda_{max}|}{|\lambda_{min}|} \approx 1$ per ottenere risultati migliori.

Bibliografia

- [1] NumPy-team. «Numpy package.» (2025), indirizzo: <https://numpy.org/> (visitato il giorno 09/06/2025) (cit. a p. 2).
- [2] SciPy-team. «SciPy package.» (2025), indirizzo: <https://scipy.org/> (visitato il giorno 09/06/2025) (cit. a p. 2).
- [3] Pallets. «Click: Pallets projects.» (2025), indirizzo: <https://click.palletsprojects.com/en/stable/#> (visitato il giorno 09/06/2025) (cit. a p. 2).
- [4] The-Matplotlib-development-team. «Matplotlib: Visualization with Python.» (2025), indirizzo: <https://matplotlib.org/> (visitato il giorno 09/06/2025) (cit. a p. 2).