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

2022/2023

Progetto 1 Bis

METODI DEL CALCOLO SCIENTIFICO

MINI LIBRERIA PER SISTEMI LINEARI

Membri del team:

Matteo Rondena 847381

Luca Loddo 844529

Riccardo Moschi 856243

Indice

Introduzione

Questo progetto si concentra sull'implementazione in linguaggio Python di una libreria per la risoluzione di sistemi lineari. Lo scopo finale del progetto è l'ottenimento di una libreria che esegua solutori iterativi, limitatamente ai quattro metodi presi in esame, svolti sulla base di input costituiti da matrici simmetriche e definite positive.

I metodi principali riguardanti il core della libreria sono: Jacobi, Gauß-Seidel, Gradiente e Gradiente Coniugato.

1. Descrizione Architettura

In questo capitolo è descritta l'architettura utilizzata per la progettazione della nostra libreria Python, secondo le richieste del Progetto 1 bis.

Dato che il nostro gruppo è composto da tre membri abbiamo sviluppato funzionalità aggiuntive nel progetto, descritte nei prossimi capitoli.

La libreria, denominata pyLibraryMetodi, è costituita da diverse classi Python che interagiscono gerarchicamente tra loro.

In **pyLibraryMetodi** abbiamo tutte le classi .py necessarie alla risoluzione dei sistemi lineari.

Nella libreria abbiamo una cartella **dati** contenente tutte le matrici fornite per il testing e calcolo delle performance: spa1.mtx, spa2.mtx, vem1.mtx e vem2.mtx.

Nel package **tests** sono presenti diversi unit test per assicurarsi che l'esecuzione dei vari metodi iterativi rispetti determinate soglie in termini di tempo, numero di iterazioni ed errore relativo percentuale.

Nei seguenti paragrafi spiegheremo molto brevemente a cosa servono le classi implementate e le tecnologie utilizzate.

1.1 Tecnologie utilizzate

Tra le tecnologie utilizzate, troviamo PyCharm come ambiente di sviluppo e, ovviamente, 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**.

1.2 Descrizione della struttura

Nell'architettura, il file **main** restituisce informazioni sull'esecuzione con un metodo logger *configLogger()*, inoltre presenta un metodo denominato *buildMatrixPathsList(directory)*, al quale viene passato un path di directory, in questo caso la cartella dati di cui sopra, della quale restituisce tutti i path delle matrici presenti in essa.

Il metodo *main(),* per ognuna delle matrici restituite da *buildMatrixPathsList(directory)*, per ogni tolleranza e per ogni metodo, va a richiamare il metodo *solve()* e stampa a video le informazioni richieste di ogni esecuzione.

Il file **readMatrix.py** consente di leggere le matrici col metodo *read()*.

La classe **iterativeMethod** è il core della libreria, al cui interno troviamo i seguenti metodi:

- *solve()*: metodo che esegue la risoluzione del sistema lineare
- *__init__()*: costruttore per il salvataggio di diverse strutture, tra cui
    - matA, ovvero la matrice in input
    - il vettore exactX, cioè il vettore per la soluzione esatta (utilizzato in seguito per il calcolo dell'errore)
    - il vettore x per il calcolo della soluzione approssimata, di dimensione N rispetto alla matrice matA di dimensione N x N in input
    - il vettore b, calcolato come prodotto matrice-vettore tra matA e exactX
    - tol, ovvero la tolleranza in input
    - time, ovvero la variabile per il salvataggio del tempo di esecuzione della risoluzione del sistema lineare
    - nIterations, variabile che permette di salvare il numero di iterazioni eseguite
    - maxIter, variabile fissata a 20000, numero massimo di iterazioni per la convergenza di ogni metodo
    - d, vettore che salva al suo interno il residuo scalato, utilizzato per il metodo del gradiente coniugato
    - tril, matrice triangolare utilizzata con il metodo Gauss-Seidel
    - errore, vettore per il salvataggio dell'errore relativo ad ogni iterata, utilizzato per il plot nella GUI
    - denominator, cioè il calcolo del denominatore nel criterio di arresto (norma del vettore b)
- *residuoNum()*: metodo per il calcolo del residuo scalato
- *stopIter()*: metodo che restituisce un valore booleano per fermare le iterazioni dei metodi. Al suo interno è presente l'algoritmo che implementa il criterio d'arresto.
- *update()*: metodo che effettua l'aggiornamento del vettore x ad ogni iterata
- *absoluteError()*, *relativeError()* e *percentageError()*: metodi per il calcolo dell'errore assoluto, relativo e percentuale (relativo)

Per quanto riguarda i files denominati **jacobi.py, gaussSeidel.py, gradiente.py e gradienteConiugato.py**, vengono utilizzati per aggiornare il vettore ad ogni iterata, ognuno secondo lo specifico metodo.

1.3 Interfaccia grafica – GUI

Oltre a quanto richiesto nella specifica del progetto, abbiamo deciso di sviluppare una GUI (Graphical User Interface) per migliorare la user experience dell'utente.
L'interfaccia grafica consente di scegliere le matrici che si vogliono usare come input e la tolleranza da utilizzare, inoltre è possibile inserire, grazie al pulsante **Open,** delle nuove matrici su cui eseguire tutte le operazioni, si veda Figura 1.

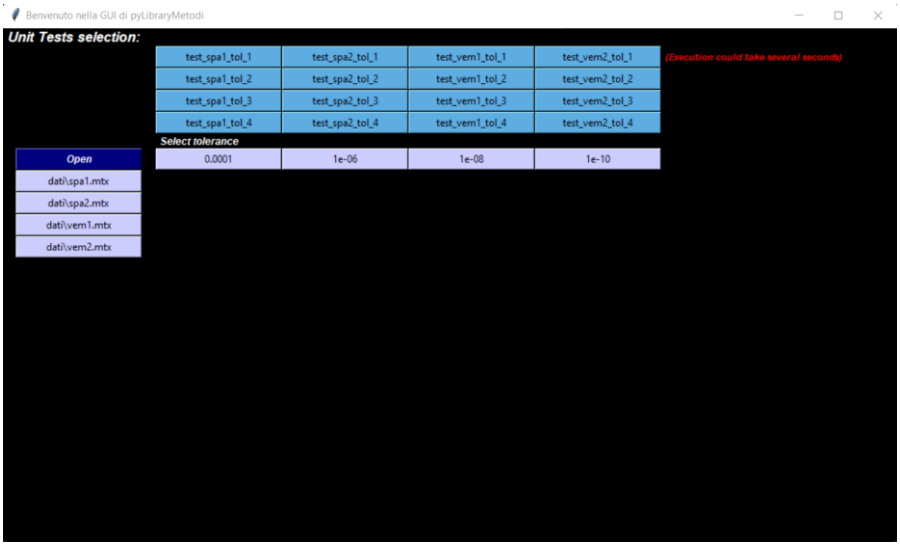


Figura 1: Interfaccia grafica (GUI)

Una volta scelta la tolleranza, la GUI completerà l'esecuzione e saranno visualizzati i risultati sull'interfaccia, che aprirà una nuova finestra in cui saranno presentati i plot delle performance ottenute. In questi plot è possibile notare l'errore relativo sull'asse delle ordinate e il numero di iterazioni svolte sulle ascisse, si veda Figura 2.

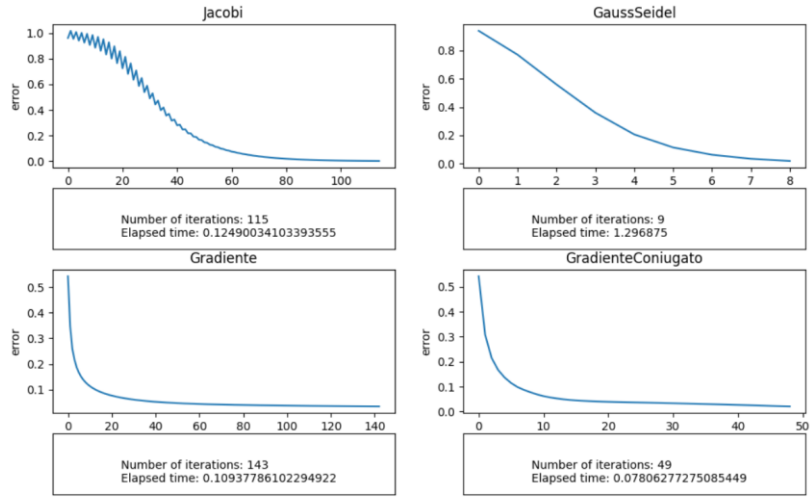


Figura 2: risultati esecuzione metodi e andamento dell'errore relativo

Inoltre, nell'interfaccia grafica, sono presenti dei bottoni che permettono di eseguire gli unit test, riferiti ad una singola matrice per una specifica tolleranza. Al termine dell'esecuzione si aprirà una finestra che mostrerà i risultati ottenuti, si veda Figura 3.

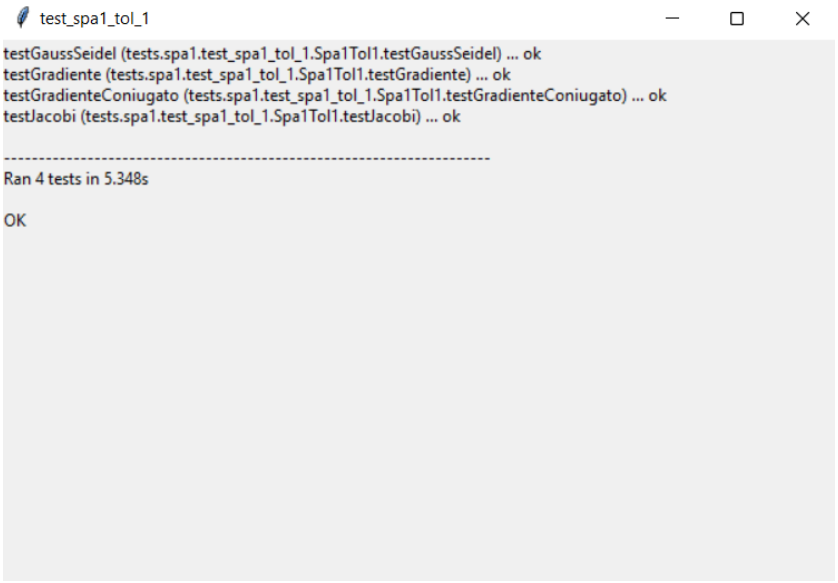


Figura 3: Unit test di spa1 con tolleranza 10^-4

1.4 Unit Test

I file di test presentano dei test case volti a valutare la corretta esecuzione dei metodi, basandoci sul numero di iterazioni, errore e tempo di esecuzione dei metodi. Per l'implementazione è stato sfruttato il modulo **unittest,** il quale fornisce strumenti per costruire e lanciare test.
È stato usato il metodo *setUp()* per fornire istruzioni da eseguire prima di lanciare ogni test. In ognuno di essi, viene eseguito il metodo *solve()* e si fanno determinate asserzioni per verificare i risultati attesi.
Gli unit test, come per il file *main,* possono essere lanciati sia da terminale sia da interfaccia grafica, si veda Figura 1.

2. Esecuzione del codice

Il codice può essere eseguito in due modalità:

- **Da Terminale**: eseguendo il file *main* dal terminale
- **Dall'interfaccia grafica**: selezionando la matrice che si vuole analizzare, saranno mostrati dei nuovi bottoni per la selezione della tolleranza, con cui eseguire i metodi e sarà eseguito il *main.*

Lanciando il file *main,* la libreria esegue la risoluzione dei sistemi lineari in riferimento alle 4 matrici fornite. Avendo 4 metodi da eseguire, con 4 tolleranze su 4 matrici, il nostro codice esegue tutte le combinazioni possibili di esecuzione.
Avremo perciò 4^3 combinazioni di esecuzioni, ovvero 64, che saranno presentati nei capitoli successivi.
Di seguito è riportata una porzione del file di log ottenuto dal completamento di tutte le esecuzioni, si veda Figura 4, per mostrare la visualizzazione dei risultati da terminale.

```
2023-05-24 22:08:24,481 INFO     Program started
2023-05-24 22:08:24,482 INFO     --------------------------------------------------
2023-05-24 22:08:24,482 INFO     Solving linear system with matrix A at path: dati\spa1.mtx
2023-05-24 22:08:24,817 INFO     --------------------------------------------------
2023-05-24 22:08:24,817 INFO     Tolerance selected: 0.0001
2023-05-24 22:08:24,817 INFO     --------------------------------------------------
2023-05-24 22:08:24,817 INFO     Iterative method selected: Jacobi
2023-05-24 22:08:24,981 INFO     Number of iterations: 115
2023-05-24 22:08:24,982 INFO     Relative error: 0.17708118132156309
2023-05-24 22:08:24,982 INFO     Percentage error: 0.18%
2023-05-24 22:08:24,982 INFO     Elapsed time: 0.16 seconds
2023-05-24 22:08:24,982 INFO     --------------------------------------------------
2023-05-24 22:08:24,982 INFO     Iterative method selected: GaussSeidel
2023-05-24 22:08:26,512 INFO     Number of iterations: 9
2023-05-24 22:08:26,512 INFO     Relative error: 1.819346991107023
2023-05-24 22:08:26,512 INFO     Percentage error: 1.82%
2023-05-24 22:08:26,512 INFO     Elapsed time: 1.53 seconds
2023-05-24 22:08:26,513 INFO     --------------------------------------------------
2023-05-24 22:08:26,513 INFO     Iterative method selected: Gradiente
2023-05-24 22:08:26,613 INFO     Number of iterations: 143
2023-05-24 22:08:26,613 INFO     Relative error: 3.4612824950369645
2023-05-24 22:08:26,613 INFO     Percentage error: 3.46%
2023-05-24 22:08:26,614 INFO     Elapsed time: 0.1 seconds
2023-05-24 22:08:26,614 INFO     --------------------------------------------------
2023-05-24 22:08:26,614 INFO     Iterative method selected: GradienteConiugato
2023-05-24 22:08:26,674 INFO     Number of iterations: 49
2023-05-24 22:08:26,674 INFO     Relative error: 2.0800035393044363
2023-05-24 22:08:26,674 INFO     Percentage error: 2.08%
2023-05-24 22:08:26,674 INFO     Elapsed time: 0.06 seconds
2023-05-24 22:08:26,674 INFO     --------------------------------------------------
2023-05-24 22:08:26,674 INFO     Tolerance selected: 1e-06
2023-05-24 22:08:26,674 INFO     --------------------------------------------------
2023-05-24 22:08:26,674 INFO     Iterative method selected: Jacobi
2023-05-24 22:08:26,858 INFO     Number of iterations: 181
2023-05-24 22:08:26,858 INFO     Relative error: 0.0017979247342011 48
2023-05-24 22:08:26,859 INFO     Percentage error: 0.0%
2023-05-24 22:08:26,859 INFO     Elapsed time: 0.18 seconds
2023-05-24 22:08:26,859 INFO     --------------------------------------------------
2023-05-24 22:08:26,859 INFO     Iterative method selected: GaussSeidel
```

Figura 4: Log file

3. Analisi dei risultati

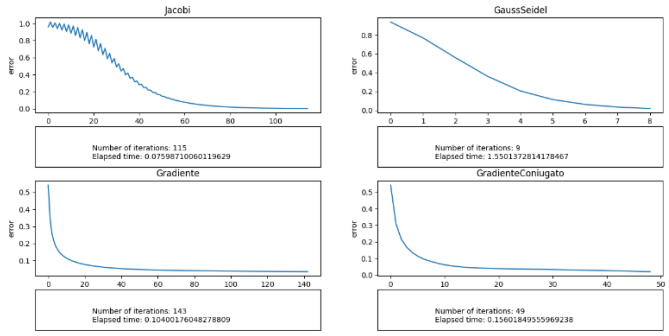
Le 4 matrici fornite sono in forma sparsa e presentano le seguenti caratteristiche raccolte nella seguente Tabella 1:

	SPA1	SPA2	VEM1	VEM2
Righe	1.000	3.000	1.681	2.601
Colonne	1.000	3.000	1.681	2.601
Dati non-zero	182.434	1.633.298	13.385	21.225
Dati Totali	1.000.000	9.000.000	2.825.761	6.765.201
% Dati sparsi	0,1824%	0,1814%	0,0047%	0,0031%

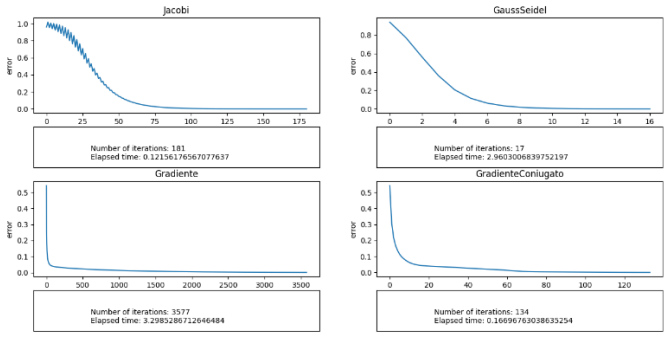
Tabella 1: Analisi Matrici

Di seguito sono riportati i grafici generati dall'esecuzione dei metodi per le singole matrici con una tolleranza specifica.

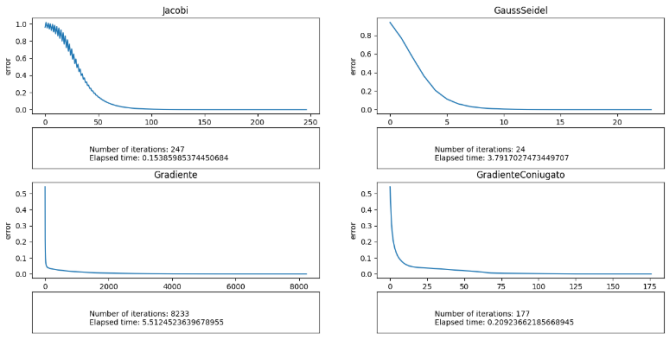
spa1.mtx tol=10e-04



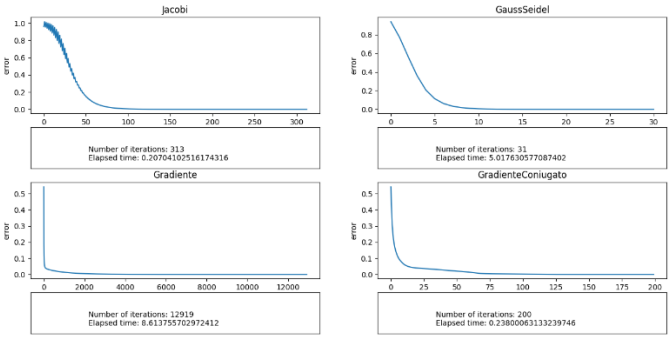
spa1.mtx tol=10e-06



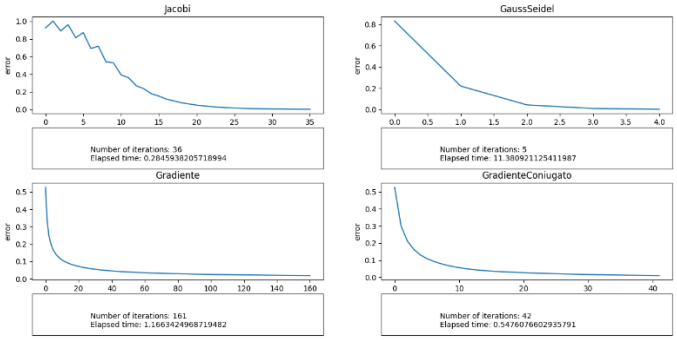
spa1.mtx tol=10e-08



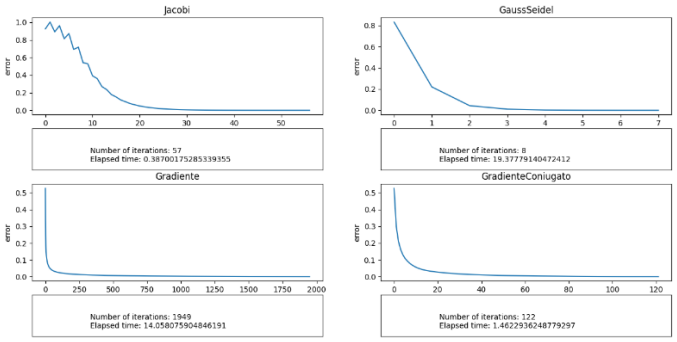
spa1.mtx tol=10e-10



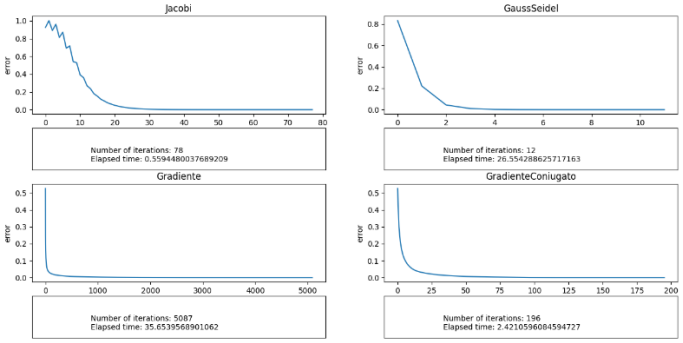
spa2.mtx tol=10e-04



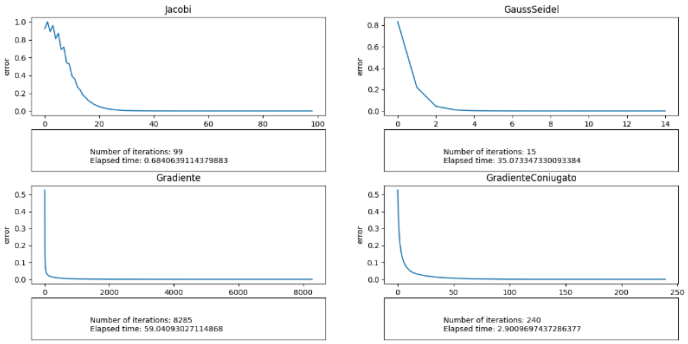
spa2.mtx tol=10e-06



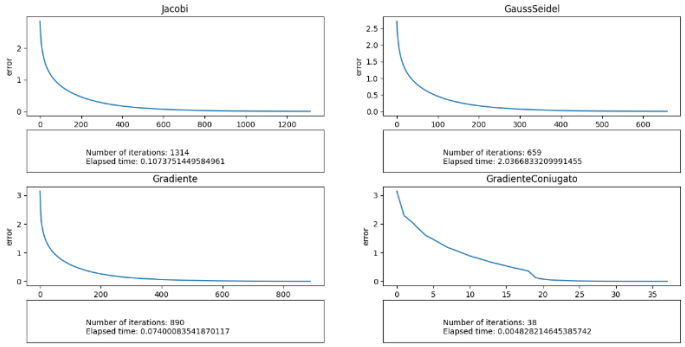
spa2.mtx tol=10e-08



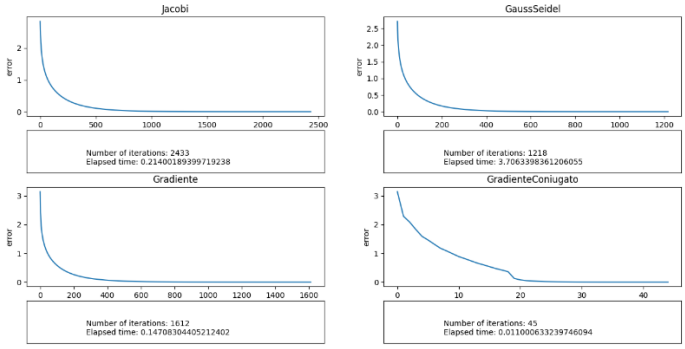
spa2.mtx tol=10e-10



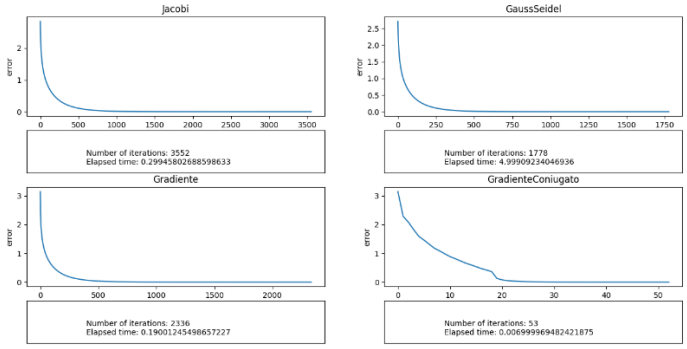
vem1.mtx tol=10e-04



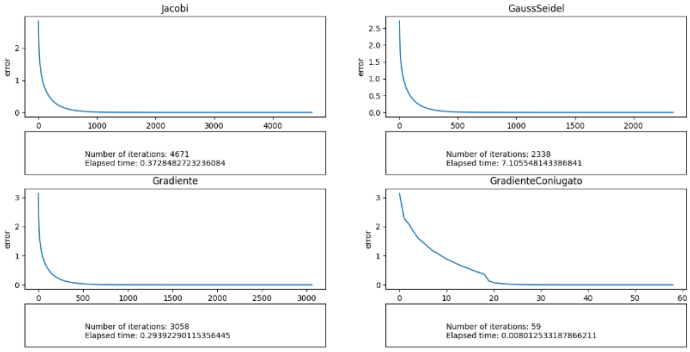
vem1.mtx tol=10e-06



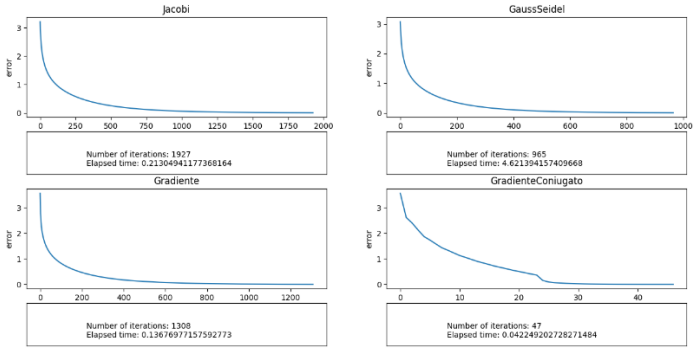
vem1.mtx tol=10e-08



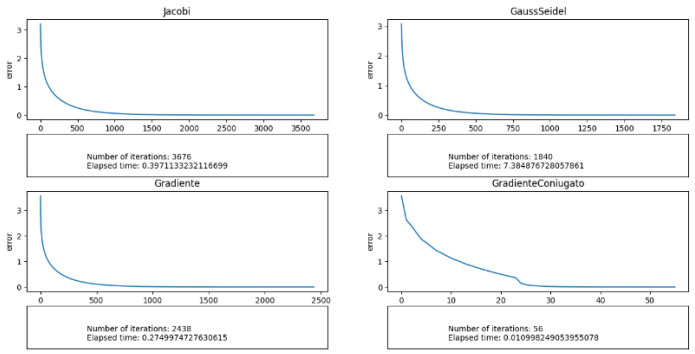
vem1.mtx tol=10e-10



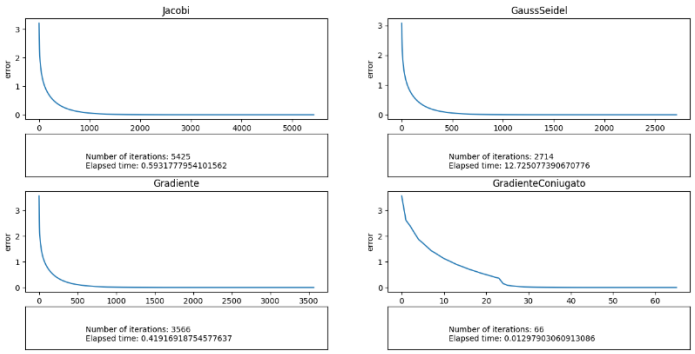
vem2.mtx tol=10e-04



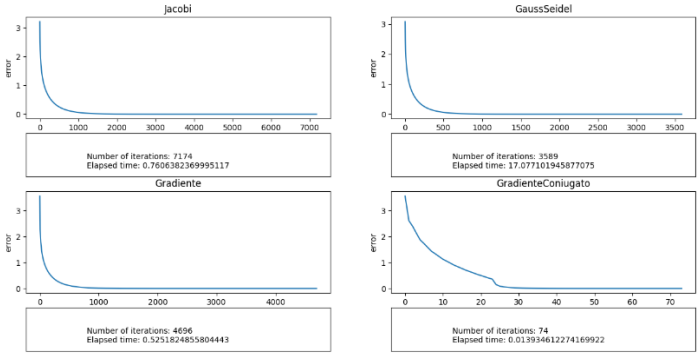
vem2.mtx tol=10e-06



vem2.mtx tol=10e-08



vem2.mtx tol=10e-10

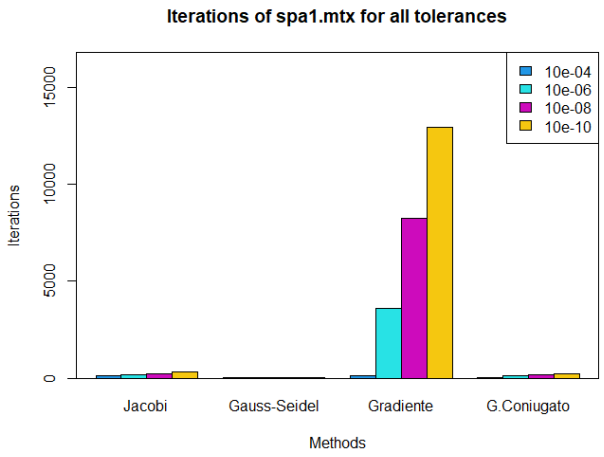


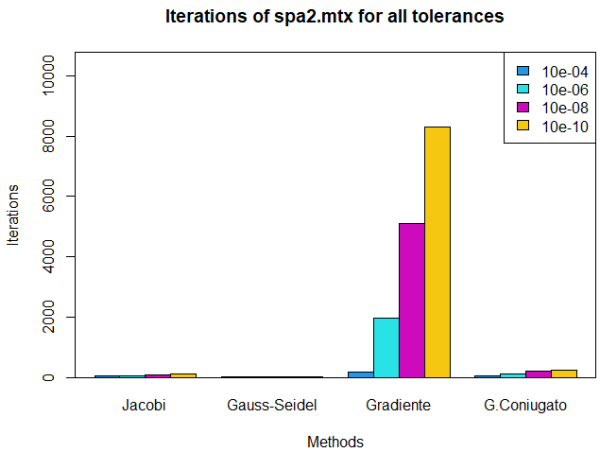
Dai risultati ottenuti dai grafici precedenti possiamo notare che esiste una forte correlazione tra il crescere delle iterazioni e la riduzione dell'errore, questo è osservabile soprattutto nelle prime iterazioni dei metodi, dove la riduzione dell'errore è particolarmente accentuata.

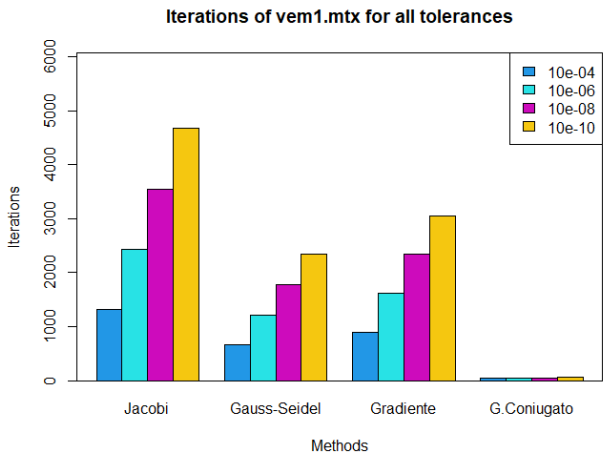
Per quanto riguarda le matrici spa1 e spa2, possiamo notare come il metodo di Jacobi abbia un andamento irregolare nella prima fase di iterazioni (potremmo dire per le prime 40 iterazioni circa per spa1 e le prime 10 circa per spa2). Per andamento irregolare si intende che l'errore aumenta e diminuisce via via fino a stabilizzarsi.
Questo comportamento si nota soprattutto con spa1, mentre con spa2 è molto meno accentuato.
Inoltre, i metodi di Jacobi e Gauss-Seidel condividono la particolarità di dimezzare l'errore percentuale molto più "lentamente" in termini di iterazioni, rispetto ai metodi del gradiente e del gradiente coniugato che, al contrario, dimezzano quasi immediatamente l'errore, di fatto stabilizzandosi a uno 0.% dopo circa 10 iterazioni.
Altre metriche interpretabili da questi plot, sempre per quanto riguarda spa1 e spa2, sono il tempo di esecuzione e il numero di iterazioni in termini assoluti. Il metodo di Jacobi risulta essere il più veloce in termini di tempo (nell'ordine dei secondi), mentre il metodo di Gauss-Seidel presenta il numero più basso di iterazioni.

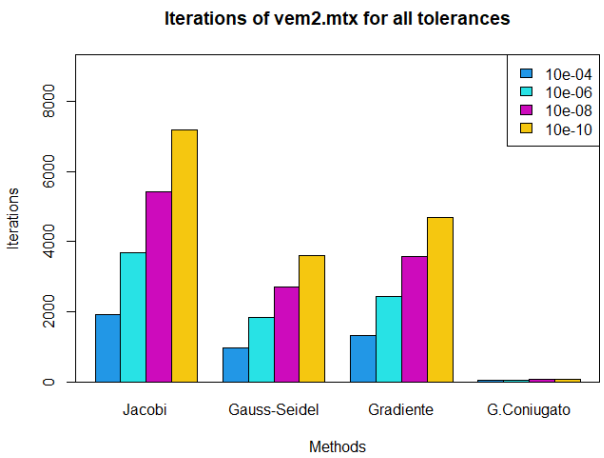
Per quanto riguarda, invece, le matrici vem1 e vem2, possiamo notare che i metodi di Jacobi, Gauss-Seidel e del Gradiente condividono lo stesso andamento, mentre il metodo del Gradiente Coniugato presenta diverse irregolarità ma stessa direzioni degli altri metodi precedentemente citati.
Nonostante ciò, il metodo del Gradiente Coniugato risulta essere il più veloce sia in termini di tempo sia in termini di iterazioni.
Nei seguenti barplot abbiamo generato per ogni matrice dei grafici esplicativi delle performance, che dimostrano le osservazioni e analisi precedentemente fatte.

3.1 Iterazioni



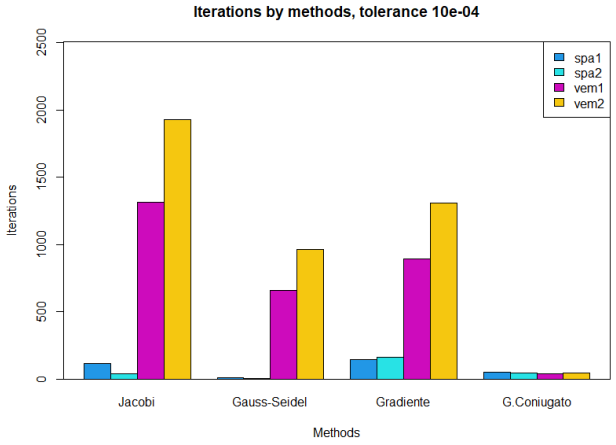


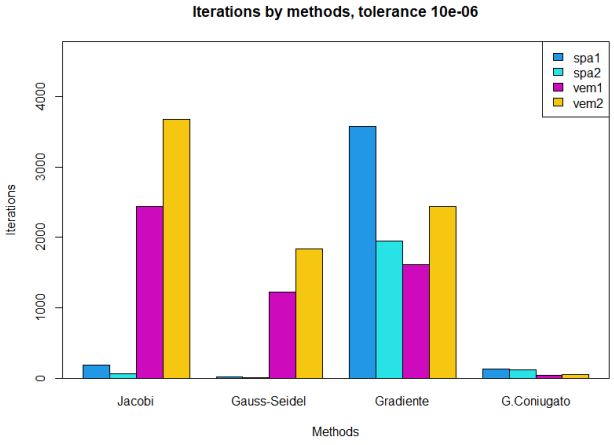


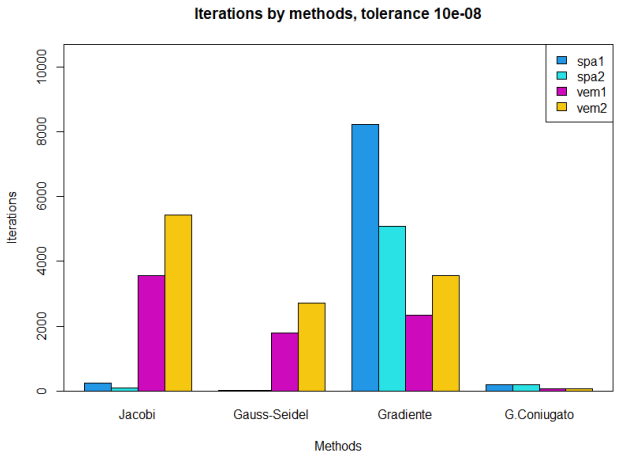


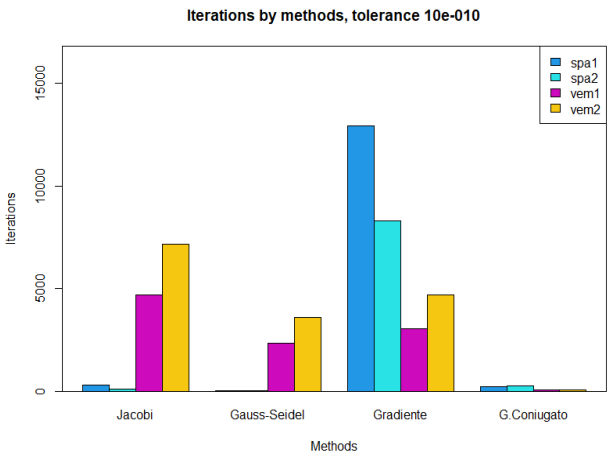
Si può notare inoltre che, per le matrici spa1 e spa2, il metodo del Gradiente converge a soluzione con un numero di iterazioni molto alto rispetto agli altri metodi.
Per quanto riguarda, invece, le matrici vem1 e vem2, i metodi di Jacobi, Gauss-Seidel e del Gradiente convergono a soluzione con numeri, in termini di iterazioni, abbastanza similari.
Tutte le matrici condividono, poi, il fatto che al diminuire della tolleranza, aumenta il numero di iterazioni per convergere a soluzione.

Invece, data un'analisi complessiva, in riferimento alla singola tolleranza, abbiamo ottenuto i seguenti dati per ogni matrice che esegue tutti i metodi ad una singola tolleranza.





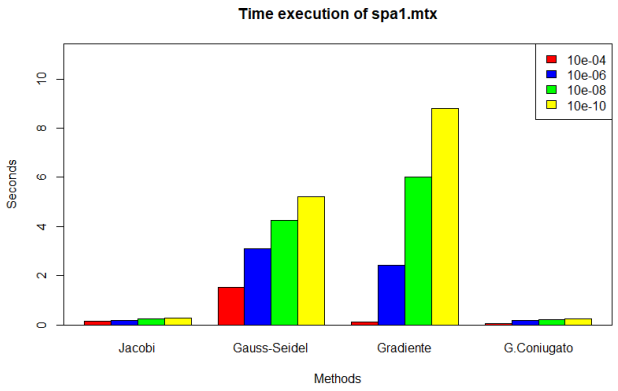


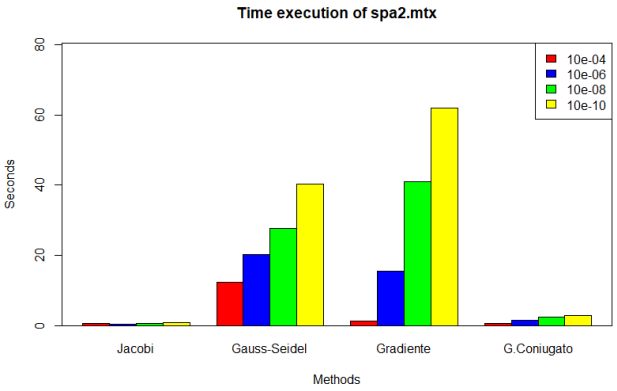


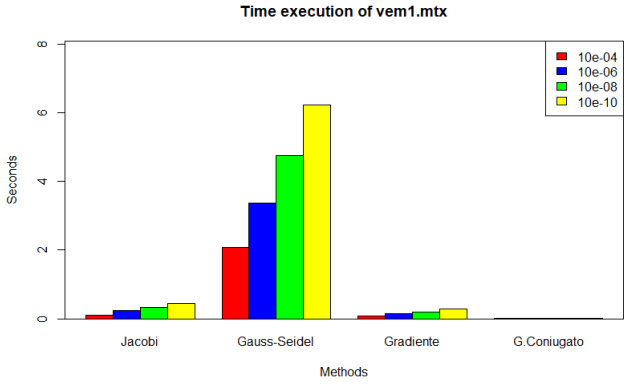
Possiamo notare che per ogni tolleranza, il metodo del Gradiente Coniugato ha una media di iterazioni nettamente migliore indipendentemente dalla matrice utilizzata.

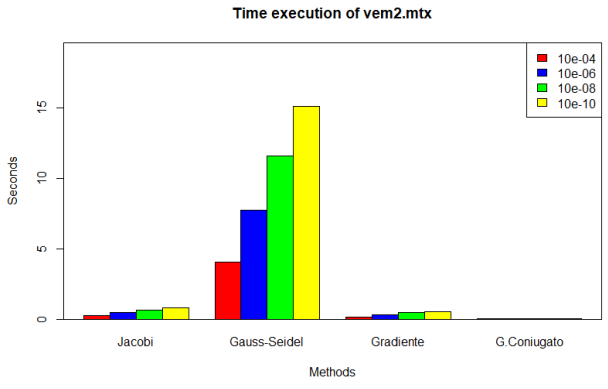
3.2 Tempi d'esecuzione

Dopo aver effettuato uno studio delle iterazioni, si è passati ad analizzare i tempi delle singole esecuzioni, che sono riportati nei seguenti barplot, seguendo le stesse modalità utilizzate precedentemente per il numero di iterazioni. Inoltre, riportiamo il dato dei tempi totali di esecuzione di tutte e 64 le combinazioni che è pari a **5 minuti e 38 secondi**.



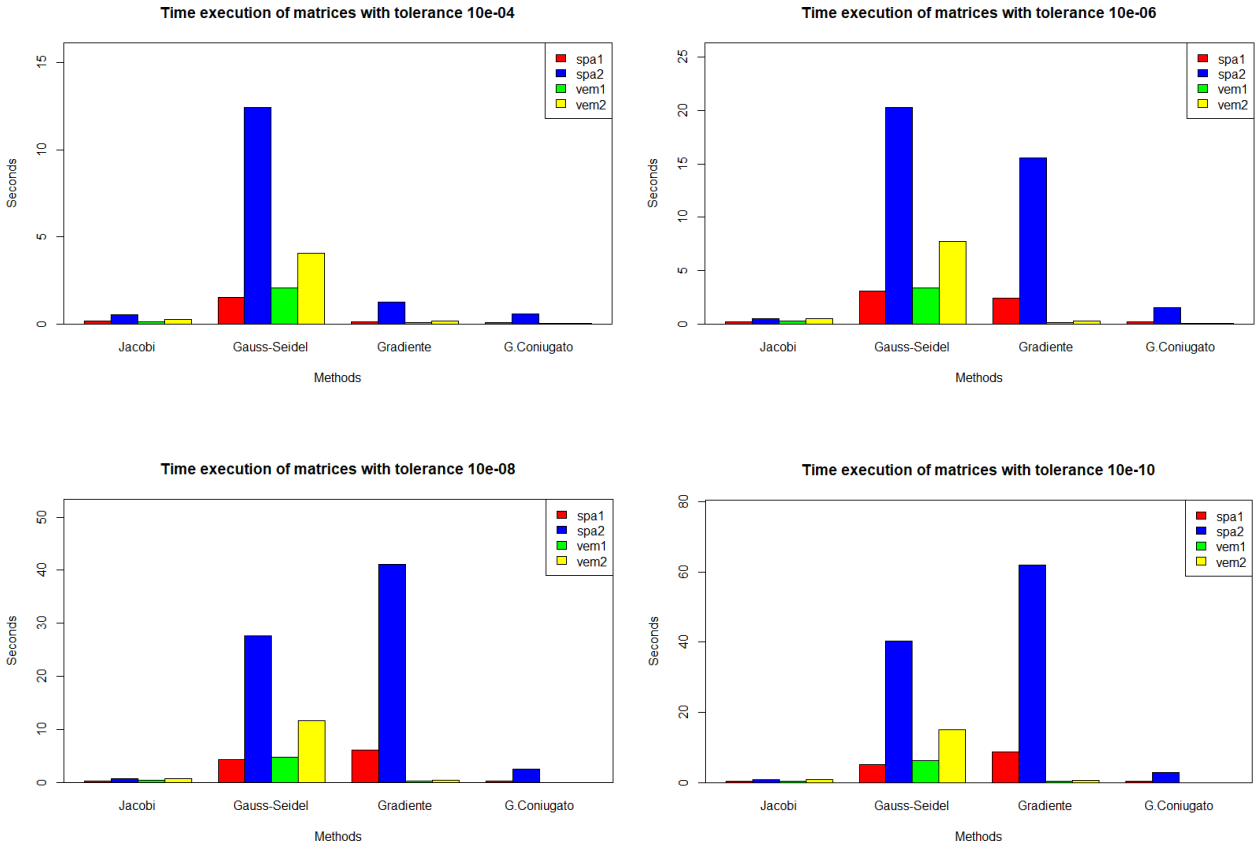






Vediamo che il tempo d'esecuzione di una matrice cresce alla riduzione della tolleranza e notiamo che il Gradiente Coniugato ha i migliori tempi di esecuzione in media, anche se per spa1 e spa2 le performance dei tempi sono molto simili a quelle del metodo Jacobi.

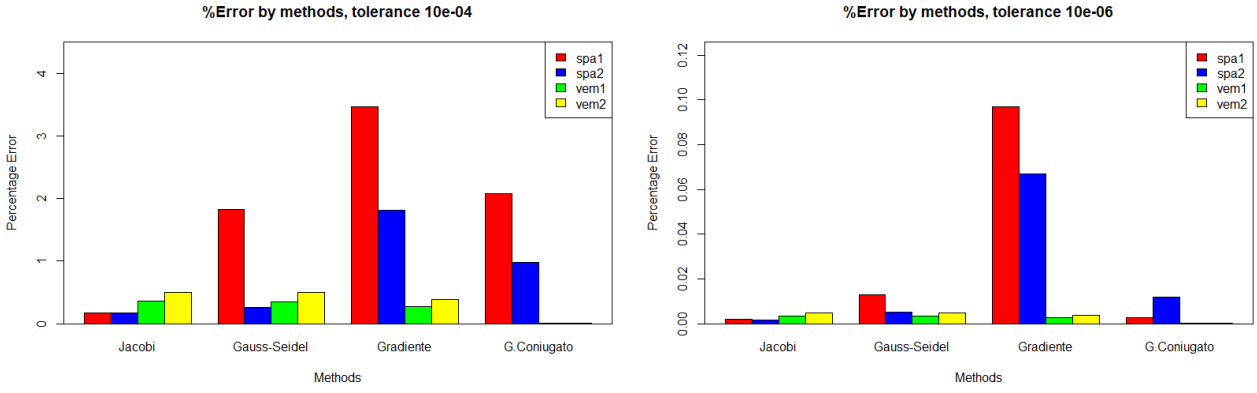
Invece nei prossimi barplot, riportiamo i tempi di esecuzione di tutte le matrici per una tolleranza fissata.

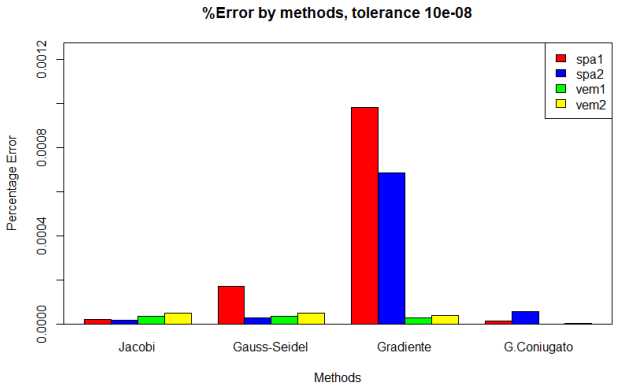


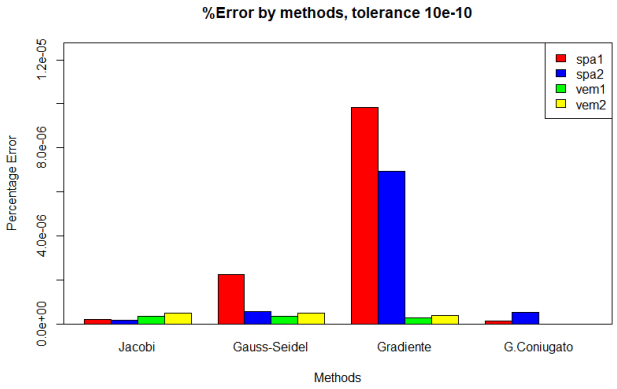
Osservando i tempi d'esecuzione per ogni tolleranza, notiamo che per la stessa tolleranza i metodi Jacobi e Gradiente Coniugato hanno mediamente le migliori performance, che si avvicinano molto.

3.3 Errore

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







Notiamo che l'errore, come ci si aspetta, al diminuire della tolleranza si riduce notevolmente, come si evince osservando le ascisse dei barplot, che presentano una riduzione drastica delle misurazioni, anche di alcuni ordini di grandezza, fino ad arrivare anche ad e^(-6).
Con la tolleranza di 10^(-4) abbiamo un errore percentuale ancora apprezzabile, che si attesta tra lo 0.2% ed il 3.5%. Passando alle tolleranze più basse tale percentuale di errore si abbassa allo 0.0% con dei picchi allo 0.1% solo nel caso di tolleranza pari a 10^(-6).

3.4 Analisi casi specifici

Da un'analisi globale, notiamo la presenza di casi specifici che ci permettono di svolgere ulteriori analisi del comportamento dei metodi.
In particolare, è osservabile il comportamento peculiare del metodo del Gradiente in relazione alla matrice spa1. Tale metodo presenta una "esplosione" di iterazioni dovuta alla diminuzione della tolleranza; infatti, a partire dalla tolleranza pari a 10^(-6), il numero delle iterazioni cresce rapidamente, passando da 143 iterazioni a 3577, per poi passare a 8233 e, infine, 12919.

Può essere fatta un'ulteriore riflessione sui tempi di esecuzione dei metodi. Tali tempi, come da previsione, si incrementano con la diminuzione della tolleranza.
Come possiamo notare, il metodo di Gauss-Seidel ha un impatto importante sui tempi di esecuzione di tutte e quattro le matrici, invece, per quanto riguarda il metodo del Gradiente notiamo un comportamento particolare.
Il metodo del Gradiente, infatti, per spa1 e spa2 ha un comportamento similare a quello notato per il metodo di Gauss-Seidel, invece, le matrici vem1 e vem2 durante l'esecuzione del metodo del Gradiente presentano un comportamento contrario rispetto al metodo di Gauss-Seidel, dove i tempi di esecuzione della stessa matrice si riducono notevolmente.

Osserviamo che la matrice spa2 presenta i tempi più onerosi, quando andiamo a paragonarne l'esecuzione parallelamente a quella delle altre matrici per le singole specifiche tolleranze. L'onerosità dei tempi è dovuta alle sue grosse dimensioni.

4. Riflessioni su Unit Test e GUI

Gli unit test implementati, dopo aver raccolto i risultati delle esecuzioni dei metodi, ci consentono di fornire dei parametri ai test case per valutare la compatibilità della libreria con l'hardware dell'utente. Quindi eseguendo i test su un determinato dispositivo, consentiamo all'utente finale di valutare se il suo apparecchio sia compatibile ad un utilizzo efficiente della nostra libreria.

Questo è svolto tramite la verifica delle asserzioni da noi formulate, sulla base dei risultati ottenuti, ovvero ottenendo la maggioranza/totalità dei test superati l'hardware sarà altamente performante con questa libreria di risoluzione per sistemi lineari.

La GUI permette l'esecuzione dei singoli test e dei metodi. Si è scelto di implementare una GUI per migliorare la user experience della libreria, così da renderlo un software fruibile a tutti.
Inoltre, la nostra scelta progettuale ha previsto anche la possibilità di aggiungere altre matrici, per venire incontro ad un caso d'uso reale. Ovviamente la GUI è molto essenziale in termini di funzionalità, le quali potrebbero essere espanse in aggiornamenti futuri.

5. Conclusioni

Di seguito una tabella che sintetizza, per ogni matrice, quale sia il miglior metodo risolutivo per una matrice, in termini di tempo e iterazioni.

	SPA1	**SPA2**	**VEM1**	**VEM2**
TEMPO	JACOBI	JACOBI	GRADIENTE CONIUGATO	GRADIENTE CONIUGATO
ITERAZIONI	GAUSS-SEIDEL	GAUSS-SEIDEL	GRADIENTE CONIUGATO	GRADIENTE CONIUGATO

Le principali difficoltà riscontrate nello sviluppo del progetto sono sicuramente quelle relative ai tentativi di abbassamento dei tempi di esecuzione dei metodi. Ci siamo accorti, infatti, che seguendo "alla lettera" gli algoritmi e le formule presentate a lezione, non avremmo ottenuto una libreria tanto performante quanto quella allo stato attuale, dove, riportiamo ancora il dato, tutte le 64 combinazioni possibili di matrici, metodi e tolleranze vengono eseguite entro un tempo totale di 5 minuti e 38 secondi.
I miglioramenti sono dovuti alle sezioni denominate "rapidità" presenti nelle note del docente e anche al salvataggio di determinati calcoli e parametri, in modo da non doverli ri-eseguire in secondi momenti.
Tra le difficoltà riscontrate si può citare, inoltre, la definizione di un'architettura che fosse pulita e ordinata. Infatti, ci siamo accorti che Python non sia la scelta più "saggia" se si vuole avere un approccio object-oriented allo sviluppo di un'applicazione software.