Università degli Studi di Milano Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica - LM 18

Metodi del Calcolo Scientifico Progetto 1

Algebra lineare numerica

Sistemi lineari con matrici sparse simmetriche e definite positive

Componenti del gruppo:
Bancora Davide - M.905588
Donato Benedetta - M.905338
Dubini Emanuele - M.904078

Anno Accademico 2022-2023

Indice

Elenco delle figure

1 Introduzione

Lo scopo di questo progetto è quello di studiare ed analizzare l'implementazione del **metodo di Cholesky** per la risoluzione di sistemi lineari associati a matrici sparse, simmetriche e definite positive confrontando l'implementazione realizzata in **MATLAB®** con quella in ambienti di programmazione open source: **R**, **Java** e **Python**. Il confronto sarà effettuato eseguendo il progetto sulla stessa macchina virtuale utilizzando sia il sistema operativo **Windows** che **Linux** e verterà su criteri come il tempo di esecuzione, l'accuratezza, l'utilizzo di memoria, la facilità d'uso e la documentazione delle librerie testate.
inoltre le **matrici sparse** presentano un gran numero di entrate nulle in cui, spesso, la quantità di elementi diversi da zero su ciascuna riga è relativamente bassa, indipendentemente dalla dimensione della matrice stessa: numerosi problemi nel campo del calcolo scientifico richiedono la soluzione di uno o più sistemi lineari con matrici di questo tipo. Le matrici sparse permettono di essere memorizzate in modo compatto, tenendo conto solo degli elementi diversi da zero. Ad esempio, è sufficiente memorizzare la posizione (i, j) di ciascun elemento diverso da zero insieme al suo valore a[i,j] ignorando semplicemente gli elementi nulli.
Per risolvere i sistemi lineari sarà utilizzato il **metodo di Cholesky**, il quale può essere applicato solo a **matrici simmetriche** e **definite positive**. Quelle da noi utilizzate fanno parte della SuiteSparse Matrix Collection, che raccoglie matrici sparse

provenienti da applicazioni di problemi reali in diverse aree (ingegneria strutturale, fluidodinamica, elettromagnetismo, termodinamica, computer graphics/vision, network e grafi).

In particolare, le matrici simmetriche e definite positive che verranno analizzate in questo progetto includono:

- Flan 1565
- StocF-1465
- cfd2
- cfd1
- G3 circuit
- parabolic fem
- apache2
- shallow water1
- ex15

Per l'implementazione del progetto sono stati utilizzati diversi linguaggi di programmazione con l'intento di confrontare l'implementazione del **metodo di Cholesky** utilizzando **MATLAB®**, la **libreria Matrix** per il linguaggio R, la **libreria EJML** per il linguaggio Java e la **libreria scikit-sparse** per il linguaggio Python.
Le tre implementazioni sono state analizzate sotto i seguenti aspetti:

- Data una matrice A, il **tempo in secondi** necessario per calcolare la sua fattorizzazione R tramite il metodo di Cholesky e trovare successivamente la soluzione x data da $A \cdot x = b$, con b termine noto calcolato in precedenza come:

  $$b = A \cdot x_e$$

  con $x_e = [1\ 1\ 1\ 1\ 1\ 1\ ...]$

- L'**errore relativo** tra la soluzione esatta x_e e quella calcolata x, definito da:

  $$errore_{rel} = \frac{||x - x_e||_2}{||x_e||_2}$$

- La **memoria necessaria** per eseguire la fattorizzazione di Cholesky e risolvere successivamente il sistema lineare, cioè l'aumento di memoria utilizzata dal programma appena dopo aver letto la matrice fino al termine della risoluzione del sistema.

É possibile approfondire le specifiche relative alla macchina utilizzata per i test e i vari sistemi operativi impiegati nel progetto all'interno della sezione Ambiente di Sviluppo della relazione.

1.1 Fattorizzazione di Cholesky

L'obiettivo principale del progetto consiste nell'analisi e nel confronto di diverse implementazioni ed esecuzioni della fattorizzazione di Cholesky. Questa sezione fornisce una descrizione concisa e dettagliata della metodologia applicata.
La fattorizzazione di Cholesky rappresenta una particolare tipologia di decomposizione di matrici, applicabile esclusivamente a matrici con specifiche caratteristiche. In particolare, si richiede che esse siano:

- simmetriche (ossia che soddisfino l'uguaglianza $A = A^T$);
- definite positive (tali che gli tutti gli autovalori siano strettamente maggiori di zero).

Formalmente, data una matrice simmetrica definita positiva $A \in \mathbb{R}^{n \times n}$, la fattorizzazione di Cholesky permette di esprimerla come il prodotto di una matrice triangolare superiore $R \in \mathbb{R}^{n \times n}$ e la sua trasposta:

$$A = R^T R,$$

dove la matrice $R \in \mathbb{R}^{n \times n}$, ottenuta dalla fattorizzazione di Cholesky, è una matrice triangolare superiore in cui gli elementi sulla diagonale principale sono positivi.
Inoltre, se la matrice A è definita positiva, allora la matrice R appena specificata è unica, rendendo l'algoritmo vantaggioso in termini di efficienza computazionale e precisione. Il costo computazionale della fattorizzazione di Cholesky è $O(n^3)$, dove n rappresenta il numero di righe (o colonne) della matrice A.
Una volta ottenuta la decomposizione $A = R^T R$, è possibile calcolare la soluzione di un sistema lineare $Ax = b$ mediante i seguenti passi:

1. Calcolo di y tramite la sostituzione in avanti con l'equazione $R^T y = b$, dove b rappresenta il vettore dei termini noti.

2. Calcolo di x tramite la sostituzione all'indietro con l'equazione $Rx = y$.

1.2 Ambiente di Sviluppo

Il progetto è stato sviluppato e testato su una macchina con le seguenti specifiche tecniche:

- **Processore**: AMD Ryzen 5 5600H with Radeon Graphics, 6 core(s), 12 processori logici, Cache L1: 384 KB, Cache L2: 3,0 MB, Cache L3: 16,0 MB
- **Memoria fisica installata (RAM)**: 2x8 GB SODIMM 3200MHz
- **Unità disco**: 475GB SSD
- **Architettura**: x64

Per l'esecuzione dei diversi algoritmi implementati nel progetto e per l'analisi dei risultati, sono state create due macchine virtuali utilizzando VirtualBox.
Il tutto sarà eseguito sia in ambiente Windows che Linux in modo da sottolineare le differenze in termini di tempo, accuratezza e impiego della memoria. A tale scopo, la prima macchina virtuale è stata configurata con **Windows 11 (64 bit)**, mentre la seconda macchina virtuale è stata creata utilizzando l'immagine di **Linux Mint versione 21.1**.
Entrambe le macchine virtuali sono state allocate con **10240 MB** di memoria di base e **6 processori** assegnati.

È importante sottolineare che entrambe le macchine virtuali sono state configurate in modo da avere accesso alle stesse risorse hardware, inclusa la stessa quantità di CPU e RAM. Questo ha permesso di garantire un ambiente di esecuzione controllato e confrontabile tra i due sistemi operativi, consentendo un confronto equo delle prestazioni delle diverse implementazioni.

2
MATLAB®

Questa Sezione presenta in modo dettagliato l'implementazione del **metodo di Cholesky** per la risoluzione di sistemi lineari utilizzando matrici sparse, simmetriche e definite positive, nell'ambiente **MATLAB®**.
Esso è un software proprietario ampiamente utilizzato per il calcolo numerico e l'analisi statistica, rilasciato per la prima volta nel 1984 da MathWorks Inc.
In questa sezione della relazione verrà descritto il codice realizzato fornendo una panoramica del suo funzionamento e delle funzioni native dell'ambiente MATLAB® utilizzate nel progetto.

2.1 Implementazione in MATLAB®

MATLAB, acronimo di **Matrix Laboratory**, è un potente ambiente per il calcolo numerico e l'analisi statistica, sviluppato utilizzando una combinazione di diversi linguaggi di programmazione, tra cui C e l'omonimo linguaggio MATLAB.
Per l'utilizzo di MATLAB è richiesta una licenza d'uso, ma nonostante la sua natura non `open-source`, gode di una vasta popolarità e viene utilizzato da milioni di persone in tutto il mondo.
Una delle caratteristiche distintive di MATLAB è la sua flessibilità come linguaggio di programmazione, poiché supporta diversi paradigmi, tra cui la programmazione

imperativa, procedurale, funzionale e, più recentemente, anche la programmazione orientata agli oggetti.
É disponibile per diversi sistemi operativi, tra cui Microsoft Windows e Linux, consentendo una vasta diffusione e accessibilità a un'ampia gamma di utenti. Uno dei suoi punti di forza risiede nel fatto che è un linguaggio interpretato, il che significa che le istruzioni vengono eseguite direttamente senza la necessità di una fase di compilazione in linguaggio macchina rendendo il processo di sviluppo e test più efficiente e rapido.
Un'altra caratteristica di MATLAB è la sua natura debolmente tipizzata, che significa che non è necessario specificare il tipo di una variabile durante la dichiarazione. Questa flessibilità semplifica la scrittura del codice e rende più agile l'interazione con i dati.
Grazie alla sua versatilità e alle sue potenti funzionalità, MATLAB è ampiamente utilizzato in ambito accademico, scientifico, ingegneristico e industriale per una vasta gamma di applicazioni, come la simulazione, l'elaborazione di segnali, l'analisi statistica, la modellazione matematica e altro ancora.

2.1.1 Documentazione delle funzioni utilizzate

All'interno del progetto sono state sfruttate funzioni *built-in* nativamente implementate dall'ambiente MATLAB, senza la necessità di librerie aggiuntive per completare i task richiesti dal progetto.
In particolare, riportiamo le principali funzioni utilizzate nei due script implementati:

- La funzione **load** viene utilizzata per caricare dati, da un file specificato, all'interno del *workspace* di MATLAB. Come utilizzato nello script `Main.m` , se il file specificato è di tipo MAT allora, verrà eseguita la funzione `load(filename)` che caricherà le variabili dal file

- Le funzioni **tic** e **toc** vengono utilizzate in coppia per misurare il tempo trascorso durante l'esecuzione di una porzione di codice che si desidera monitorare. All'interno dello script `CholeskySolve.m`, sfruttando la funzione **tic**, viene registrato il tempo corrente, mentre la funzione **toc** utilizza il valore registrato per calcolare il tempo trascorso durante la risoluzione del sistema lineare usando la decomposizione di Cholesky.

- la funzione **chol** di MATLAB è stata utilizzata all'interno del file `CholeskySolve.m` per effettuare la decomposizione di Cholesky della matrice data e risolvere il

sistema lineare **Ax=b** associato. La funzione `chol` è parte integrante delle librerie di MATLAB ed è ottimizzata per offrire elevate prestazioni computazionali.
La sua documentazione dettagliata è disponibile sul sito ufficiale di MathWorks.

- La funzione \ ("**Mldivide**") viene utilizzata per risolvere il sistema lineare **Ax=b** definito dalla matrice **A** e dal termine noto **b**.

- La funzione **norm** calcola la norma euclidea del vettore specificato per parametro durante la sua invocazione. Nel progetto è utilizzata all'interno del file `CholeskySolve.m` per calcolare l'errore relativo tra la **soluzione calcolata** x e la **soluzione esatta** xe

- La funzione **ispc** permette di determinare se la versione dell'ambiente MATLAB utilizzata sia per piattaforme Windows. Di fatto, viene utilizzata all'interno dello script `CholeskySolve.m` ed è una funzione logica che restituisce `true` se il sistema operativo con cui viene eseguito MATLAB è Microsoft Windows (PC), altrimenti restituisce `false`.

- La funzione **memory** permette di visualizzare le informazioni relative alla quantità di memoria disponibile sul computer in uso e la quantità di memoria utilizzata da MATLAB. È importante notare che la funzione `memory` è disponibile solo su piattaforme **Microsoft Windows**. Ciò significa che questa funzione è utilizzabile solo se si sta eseguendo il programma su un sistema operativo Windows.

- La funzione **system** in MATLAB permette di eseguire i comandi di sistema specificati come parametro richiamando il sistema operativo in uso. Come presente nello script `CholeskySolve.m`, tramite "**free -b | grep Mem**" è possibile eseguire un comando di sistema per ottenere informazioni relative alla memoria disponibile sul computer utilizzando. Nello specifico, "**free -b**" restituisce informazioni sulle statistiche di memoria del sistema in byte, mentre eseguendo l'istruzione "**grep Mem**" filtra e restituisce solo le informazioni riguardanti la memoria totale, utilizzata, libera e cache (tutte quelle righe che contengono al loro interno la parola "Mem").

L'utilizzo delle funzioni native offerte da MATLAB offre notevoli vantaggi in termini di efficienza e prestazioni nel contesto del calcolo numerico e scientifico. L'ambiente MATLAB, nascendo con l'obbiettivo di ottimizzare l'esecuzione delle operazioni

matematiche sfruttando al meglio le risorse hardware disponibili, ed è riconosciuto per la sua capacità di fornire strumenti ottimizzati per il calcolo numerico tramite l'utilizzo di librerie sono appositamente sviluppate che garantiscono elevate prestazioni computazionali a chi le utilizza.

Per una visione completa della documentazione di tutte le funzioni disponibili in MATLAB è possibile visitare il sito: https://it.mathworks.com/help/MATLAB/.

2.2 Descrizione del Codice MATLAB

In questa sezione verrà presentato in dettaglio il codice MATLAB sviluppato per l'analisi di matrici sparse e simmetriche mediante la decomposizione di Cholesky. Il codice è stato strutturato in due file principali: `Main.m` e `CholeskySolve.m`, con l'esecuzione che parte dal file `Main.m`. L'obiettivo del codice è di valutare le prestazioni e i risultati ottenuti per ciascuna delle matrici analizzate tenendo in considerazione i seguenti parametri:

- **Tempo di Elaborazione:** il codice calcola e memorizza il tempo necessario per la risoluzione del sistema lineare associato alla matrice utilizzando la decomposizione di Cholesky.
- **Errore Relativo:** durante l'esecuzione dello script viene memorizzato l'errore relativo tra la soluzione calcolata `x` e la soluzione esatta `xe` calcolato utilizzando la norma euclidea dei vettori.
- **Memoria Utilizzata:** per rilevare la quantità di memoria RAM necessaria per la risoluzione del sistema viene misurato l'utilizzo di memoria prima e dopo la sua risoluzione. La differenza rappresenta la quantità di memoria necessaria per risolvere il sistema.

2.2.1 File `Main.m`

Il file `Main.m` rappresenta il punto di partenza per l'esecuzione dell'intero programma.
Durante la sua esecuzione, dopo aver definito all'interno dell'array `matrixNames` i nomi delle matrici da analizzare e una volta inizializzati i vari array necessari per la registrazione dei risultati ottenuti per ciascuna matrice, il codice esegue un ciclo

`for` per analizzare ciascuna delle matrici specificate precedentemente.
Per ogni iterazione, il codice esegue i seguenti passaggi:

1. Caricamento della matrice all'interno dello spazio di lavoro stampandone il nome e la dimensione.

```
% Carica la matrice dal file
tmp = load(['Matrici/', matrixNames{i}]);
matrix = tmp.Problem.A;
clear tmp

% Stampa il nome del file e le dimensioni della matrice
fprintf('\n---------------------------- %s
    ----------------------------\n\n', matrixNames{i})
```

Listing 2.1: Caricamento della matrice all'interno del *workspace* di Matlab®

2. Invocazione della funzione `CholeskySolve` per risolvere il sistema lineare associato alla matrice ottenendo il vettore delle soluzioni del sistema, il tempo di elaborazione, l'errore relativo e l'utilizzo di memoria prima e dopo la risoluzione del sistema lineare.

```
%funzione risoluzione sistema lineare
[x, time, errore_relativo, memory_used_preResolution,
    memory_used_postResolution] = CholeskySolve(matrix);
```

Listing 2.2: Richiamo funzione `CholeskySolve()`

3. Dopo la risoluzione del sistema lineare associato a ciascuna matrice, i risultati ottenuti vengono registrati in appositi array.

```
array_time(i) = time;
array_error(i) = errore_relativo;
array_memoryPre(i) = memory_used_preResolution;
array_memoryPost(i) = memory_used_postResolution;
array_memoryDiff(i) = memory_used_postResolution -
    memory_used_preResolution;
```

Listing 2.3: Memorizzazione dei risultati

Una volta completata l'analisi di tutte le matrici, i risultati ottenuti vengono opportunamente salvati all'interno in un file CSV. Questo file conterrà al suo interno una tabella in cui sono presenti i nomi delle matrici analizzate, le dimensioni di ciascuna matrice, la quantità di memoria utilizzata durante l'elaborazione, i tempi necessari per la risoluzione e gli errori relativi della soluzione di ciascun sistema lineare. Infine, il file CSV generato verrà esportato e successivamente utilizzato per la creazione di alcuni grafici mediante un apposito script in Python.

2.2.2 File `CholeskySolve.m`

La funzione `CholeskySolve` è fondamentale per la risoluzione dei sistemi lineari utilizzando la decomposizione di Cholesky. Questa funzione riceve in input una matrice `A` e restituisce il vettore delle soluzioni `x` associato alla matrice , il tempo di elaborazione, l'errore relativo e l'utilizzo di memoria per la risoluzione del sistema. Il codice all'interno dello script `CholeskySolve.m` esegue i seguenti passaggi:

1. Memorizza all'interno della variabile `memory_used_preResolution` la memoria utilizzata prima di effettuare la risoluzione del sistema.

```
memory_used_preResolution = get_memory_usage();
fprintf('\n-- Stato memoria pre risoluzione sistema: %d byte\n'
    , memory_used_preResolution)
```

Listing 2.4: lettura memoria utilizzata pre risoluzione

Per leggere la quantità di memoria utilizzata viene eseguita la funzione `get_memory_usage()` implementata come di seguito.

```
function [memory_used] = get_memory_usage()
    if ispc()
        % Se il sistema operativo e' Windows
        m = memory();
        memory_used = m.MemUsedMATLAB;
    else
        % Se il sistema operativo e' Linux
        [~, result] = system('free -b | grep Mem');
        memory_info = strsplit(result);
        memory_used = str2double(memory_info{3});
    end
```

Listing 2.5: Implementazione funzione `get_memory_used()`

2. Inizializza la variabile `n` che rappresenta il numero di righe presenti nella matrice da analizzare e la variabile `b` che costituisce il vettore dei termini noti inizializzato mediante il prodotto tra la matrice e un vettore di tutti 1.

```
%---risoluzione sistema lineare---
% Numero di righe della matrice
n = size(matrix, 1);

% Inizializza il termine noto b come un vettore dato dal
    prodotto tra A e un vettore di tutti 1
b = matrix*ones(size(matrix,1),1);

tic
```

Listing 2.6: definizione e iniziliazzazione variabili `b` ed `n`

3. Risolve il sistema lineare utilizzando la decomposizione di Cholesky sfruttando le funzioni *built-in* di Matlab `tic` e `toc` con il fine di memorizzare il tempo necessario per la risoluzione del sistema lineare all'interno della variabile `time`.

```
tic

% Calcola la matrice triangolare inferiore R di A utilizzando
    il metodo di Cholesky
%A = R' R con R triangolare sup.
R = chol(matrix);
%If S is a symmetric (or Hermitian), positive definite, sparse
    matrix, the statement R = chol(S) returns a sparse, upper
    triangular matrix R so that R'*R = S.

% Risolvi il sistema R'y = b utilizzando \
%fare sostituzione in avanti perche' R' e' tringolare inferiore
y = R'\b;

% Risolvi il sistema R'x = y utilizzando \
%fare sostituzione in indietro perche' R e' tringolare
    superiore
x = R\y;

time = toc;
```

Listing 2.7: Risoluzione del sistema lineare tramite la decomposizione di Cholesky

4. Calcola la quantità di memoria utilizzata al termine della risoluzione del sistema.

```
% Misura l'utilizzo di memoria RAM
% Serve per controllare mem su linux
memory_used_postResolution = get_memory_usage();
fprintf('\n-- Stato memoria post risoluzione sistema: %d byte\n
    ', memory_used_postResolution)
```

Listing 2.8: Calcolo memoria utilizzata

5. Calcola l'errore relativo tra la soluzione ottenuta dalla risoluzione `x` e la soluzione esatta `xe`.

```
% Calcola l'errore relativo tra x e xe utilizzando la norma
    euclidea
% Assumiamo xe come un vettore di tutti 1 per semplicita'
xe = ones(n, 1);
% Calcola la norma euclidea di xe (vettore soluzioni esatte)
norm_xe = norm(xe, 2);
% Calcola l'errore relativo
errore_relativo = norm(x - xe, 2) / norm_xe;
% Stampa dell'errore relativo
fprintf('\n-- Errore relativo tra x e xe:%s\n ', num2str(
    errore_relativo));
```

Listing 2.9: Calcolo errore relativo

6. Restituisce i parametri calcolati durante la risoluzione.

Durante l'esecuzione del punto 3, utilizzando la decomposizione di Cholesky si sfruttano le proprietà della matrice A per semplificare la risoluzione del sistema lineare ad essa associato. Di fatto, la matrice A viene rappresentata come il prodotto di due matrici triangolari: `A = R' * R`, dove R è una matrice triangolare superiore e la matrice R' è triangolare inferiore. Nel codice, la matrice `A` viene prima decomposta utilizzando la funzione `chol(matrix)` ottenendo la matrice triangolare superiore R. Successivamente, il sistema lineare R'y = b viene risolto utilizzando l'operatore \, dove R' è la matrice R trasposta. Questa risoluzione viene eseguita in avanti, poiché R' è triangolare inferiore. Successivamente, viene risolto un ulteriore sistema lineare `R * x = y`. Anche in questo caso, l'operatore \ viene utilizzato per risolvere il sistema applicando una sostituzione all'indietro, poiché R' è triangolare superiore.

In sostanza, la decomposizione di Cholesky permette di suddividere la risoluzione del sistema lineare in due passaggi più semplici rispetto alla risoluzione diretta del sistema `A * x = b`. Questo è particolarmente vantaggioso quando la matrice A è simmetrica, sparsa e definita positiva, sfruttando le sue caratteristiche per ottimizzare il processo di risoluzione.

Inoltre, sempre all'interno del file `CholeskySolve.m`, è possibile notare come la funzione `get_memory_usage` è utilizzata per ottenere la quantità di memoria necessaria per la risoluzione del sistema lineare. Infatti, in base al sistema operativo, questa funzione calcola l'utilizzo di memoria tramite il comando `memory()` per sistemi operativi Windows e il comando `free -b | grep Mem` su Linux.
Infine, è importante sottolineare che ai fini del progetto, il programma è stato rispettivamente eseguito su sistemi operativi Linux e Windows, con la creazione di file di output in formato CSV distinti per ciascun sistema operativo in modo da confrontare le prestazioni e l'utilizzo della memoria tra le due piattaforme analizzate.

Il codice sorgente appena descritto è disponibile all'indirizzo GitHub: https://github.com/EmanueleDubini/ProgettoCalcoloScientifico_Matlab.

2.2.3 Piano di supporto

Per quanto riguarda la prima parte del progetto è stata impiegata la piattaforma MATLAB, nella versione **R2023a**. Esso è un software commerciale prodotto da MathWorks Inc. sottoposto a frequente manutenzione accompagnata dal rilascio di *release* semestrali fornendo un ambiente specializzato nel calcolo scientifico e numerico ampiamente utilizzato in ambito scientifico e di ricerca grazie alla sua capacità di fornire strumenti efficienti ed ottimizzati per il calcolo numerico.

Il sito web ufficiale di MATLAB offre una vasta gamma di risorse, tra cui la documentazione completa delle funzioni del software, esempi di codice e guide per l'implementazione delle diverse funzionalità offerte in progetti specifici. Inoltre, è disponibile un forum di supporto, chiamato MATLAB Answers, dove gli utenti possono intervenire inserendo domande specifiche ed ottenere assistenza nel risolvere eventuali problemi o dubbi riguardanti l'implementazione delle funzioni.
MathWorks, l'azienda produttrice di MATLAB, fornisce inoltre un supporto tecnico diretto attraverso consulenti esperti che possono garantire assistenza personalizzata nell'affrontare sfide complesse o situazioni critiche. Questo supporto professionale

può rivelarsi particolarmente utile quando si lavora su progetti di elevate dimensioni o con requisiti specifici.

La scelta di utilizzare un software a pagamento come MATLAB offre diversi vantaggi, inclusi alti standard di affidabilità e sicurezza. MATLAB è sottoposto a rigorosi controlli di qualità e frequenti aggiornamenti per garantire stabilità e prestazioni ottimali delle funzionalità offerte. Tuttavia, è essenziale considerare anche i potenziali svantaggi che comporta l'adozione di software a pagamento dati i significativi costi non sostenibili soprattutto per aziende o organizzazioni con budget limitati. Pertanto, è quindi importante valutare attentamente il bilancio disponibile e confrontarlo con i vantaggi offerti da MATLAB prima di prendere una decisione. Esistono anche alternative `open-source`, come librerie Python per il calcolo numerico, che possono essere economicamente vantaggiose e altrettanto efficaci per molte applicazioni. Queste librerie offrono spesso una vasta comunità di sviluppatori, ma potrebbero mancare alcune funzionalità avanzate e il supporto personalizzato fornito da software a pagamento.
In sintesi, la scelta dell'ambiente di sviluppo dipende da diversi fattori, tra cui l'affidabilità, la sicurezza, il supporto tecnico, il budget disponibile e le specifiche esigenze del progetto. È fondamentale prendere in considerazione tutti questi aspetti prima di selezionare l'ambiente di programmazione più adatto alle necessità.

2.2.4 Matrici testate e supportate

La **SuiteSparse Matrix Collection** è una collezione di matrici sparse derivanti da applicazioni di problemi reali in vari settori. Essa offre una vasta gamma di matrici **simmetriche** e **definite positive** che sono ampiamente utilizzate per scopi di ricerca e sviluppo nell'ambito del calcolo scientifico e numerico. Per il progetto, una volta applicato il metodo di Cholesky alle matrici, non tutte hanno prodotto i risultati attesi. Diverse di queste, soprattutto quelle di elevate dimensioni, durante l'esecuzione del programma hanno portato alla saturazione della memoria RAM (usufruibile dal calcolatore) riportando l'errore di ***"Out of Memory"***, situazione che si verifica in seguito all'esaurimento delle risorse disponibili.
Di seguito è presente un' immagine che indica quali di queste matrici sono state correttamente elaborate dallo script MATLAB:

Nomi Matrici	Stato	Dimensione (in MB)
ex15.mat	✅	0,555
shallow_water1.mat	✅	2,263
apache2.mat	Out of memory ❌	8,302
parabolic_fem.mat	Out of memory ❌	13,116
G3_circuit.mat	Out of memory ❌	13,833
cfd1.mat	✅	14,164
cfd2.mat	✅	23,192
StocF-1465.mat	Out of memory ❌	178,368
Flan_1565.mat	Out of Memory ❌	292,858

Figura 2.1: Matrici supportate da MATLAB®

Maggiori informazioni sono disponibili alla pagina GitHub del progetto.

3 Librerie Open-Source

In questa sezione della relazione, verranno presentate le diverse implementazioni delle librerie `open-source` implementate all'interno del progetto con l'obbiettivo comune legato alla risoluzione di sistemi lineari associati a matrici simmetriche definite positive utilizzando la decomposizione di Cholesky.

3.1 Implementazione in R - Librerie

Il programma ha l'intento di analizzare una serie di matrici sparse e condurre un'analisi accurata delle performance riguardanti la decomposizione di Cholesky e la risoluzione di sistemi lineari.
Il suo sviluppo è stato guidato dalla finalità di valutare l'efficienza computazionale di tali processi, avvalendosi principalmente delle funzioni di base di R e delle librerie scientifiche `library(pryr)`, `library(here)`, `library(R.matlab)`.
L'obiettivo centrale consiste nella manipolazione di matrici sparse scientifiche evitando la conversione in matrici dense, che avrebbe comportato un consumo eccessivo della memoria.
La selezione delle librerie `open-source` è stata motivata dalla loro capacità di gestire in modo efficiente le matrici sparse e risolvere sistemi lineari in modo rapido ed efficiente, garantendo al contempo un utilizzo ottimale delle risorse del sistema.

Il codice sorgente del progetto è disponibile all'indirizzo GitHub: https://github.com/BenedettaDonato/ProgettoCalcoloScientifico_R

3.1.1 Descrizione delle librerie utilizzate

La libreria `pryr`, disponibile in R, è uno strumento per gli sviluppatori che offre un insieme di funzioni e strumenti per il monitoraggio e l'analisi della memoria durante l'esecuzione di calcoli complessi, consentendo agli utenti di diagnosticarla e ottimizzarla nei loro script.
È particolarmente adatta quando si lavora con grandi insiemi di dati in quanto aiuta a individuare potenziali problemi di memoria e fornisce informazioni utili a migliorare l'efficienza del codice.
Inoltre, questa libreria svolge un ruolo importante nel garantire che i programmi siano ottimizzati per l'esecuzione su sistemi con risorse limitate, migliorando così la riproducibilità e l'efficienza delle analisi scientifiche condotte.

La libreria `here`, importata nello script R, è uno strumento essenziale per semplificare la gestione delle directory all'interno dei progetti.
`here` può essere descritta come uno strumento fondamentale per migliorare la riproducibilità e la portabilità delle analisi condotte; di fatto facilita la gestione dei percorsi all'interno dei progetti, semplifica la condivisione del codice e migliora la trasferibilità delle analisi tra sistemi operativi.
Questo contribuisce notevolmente ad aumentare la robustezza e l'affidabilità delle ricerche scientifiche condotte, assicurando che gli altri utilizzatori possano facilmente riprodurre i risultati ottenuti.

La libreria `R.matlab` è uno strumento fondamentale per l'interazione e lo scambio di dati tra R e il formato file MATLAB (.mat).
`R.matlab` può essere descritta come una libreria essenziale per l'integrazione di dati provenienti da software e ambienti di calcolo diversi offrendo "un ponte" tra R e MATLAB e consentendo agli utenti di sfruttare al meglio le potenzialità di entrambi i linguaggi.

Oltre alle librerie illustrate precedentemente, ne sono state utilizzate delle altre, disponibili nativamente in R, per l'analisi e la manipolazione dei dati; così facendo, è

stato possibile sfruttare le potenzialità del linguaggio R semplificando la gestione dei dati provenienti da diverse fonti, condurre analisi avanzate e risultati confrontabili.

3.1.2 Codice Sorgente

In questa sezione verranno illustrate le principali funzioni create nel progetto, reperibili nel file `Utils.R`.
Il codice di seguito fornito definisce una funzione chiamata `load_matrix_from_file` che ha il compito di estrarre una matrice sparsa da un file MATLAB e restituirla come un oggetto matrix in R.
È in grado di riconoscere automaticamente la locazione della matrice all'interno del file dato e fornisce un messaggio di conferma se la matrice è effettivamente sparsa.

```
#Per caricare la matrice dal file mat e restituire la matrice sparsa
load_matrix_from_file <- function(filepath) {
  mat_data <- readMat(filepath)
  problem_elements <- mat_data$Problem
  A <- NULL
  for (element in problem_elements) {
    if (inherits(element, "dgCMatrix")) {
      A <- element
      break
    }
  }
#Verifica se la matrice e' sparsa
  if (is.sparseMatrix(A)) {
    cat("La matrice A e' sparsa \n")
  }
  return(A)
}
```

Listing 3.1: Funzione `load_matrix_from_file`

Lo scopo della funzione `is_symmetric` è verificare se una data matrice 'A' è simmetrica calcolando la sua trasposta tramite il metodo t() nativo di R e verifica se tutti gli elementi delle due matrici sono uguali.
In caso affermativo, la funzione restituirà 'TRUE', altrimenti 'FALSE'.

```
is_symmetric <- function(A) {
  A_transpose <- t(A)
  return(all(A@x == A_transpose@x))
}
```

Listing 3.2: Funzione `is_symmetric`

La funzione denominata `create_b_vector` prende in input la matrice ‘A‘, caricata precedentemente, e restituisce un vettore ‘b‘ calcolato come il prodotto matrice-vettore tra ‘A‘ e un vettore contenente tutti gli elementi pari a uno.

```
create_b_vector <- function(A) {
  n <- nrow(A)
  ones_vector <- rep(1, n)
  b <- A %*% ones_vector
  return(b)
}
```

Listing 3.3: Funzione `create_b_vector`

La funzione `cholesky_decomposition` calcola la fattorizzazione di Cholesky di ‘A‘ e restituisce la matrice risultante insieme alle informazioni sulla memoria utilizzata durante l’operazione.
Inoltre, fornisce un messaggio informativo sulla positività della matrice o sulla presenza di errori durante la fattorizzazione.
Qui di seguito è illustrato nel dettaglio il suo funzionamento:

- `memoria_iniziale <- mem_used()`: In questa riga, viene misurata la quantità di memoria utilizzata prima di eseguire la fattorizzazione di Cholesky, utile per monitorare quanto memoria viene allocata durante il processo.

- `factor <- Cholesky(A)`: calcola la fattorizzazione di Cholesky della matrice ‘A‘ utilizzando la funzione ‘Cholesky()‘ nativa in R.
  Nel caso in cui il processo termini, il programma informa l’utente che la matrice caricata è positiva.

- `memoria_finale <- mem_used()`: Dopo aver eseguito la fattorizzazione, viene nuovamente misurata la quantità di memoria utilizzata.
  Sottraendo questo nuovo valore alla memoria allocata prima di eseguire l’operazione, è possibile visualizzare la memoria totale richiesta dal processo.

```
cholesky_decomposition <- function(A) {
  # Ottieni la memoria usata prima della fattorizzazione di Cholesky
  memoria_iniziale <- mem_used()

  # Calcola la fattorizzazione di Cholesky
  tryCatch({
    factor <- Cholesky(A)
    cat("La matrice A e' definita positiva \n")
```

```
  }, error = function(err) {
    cat("La matrice A non e' definita positiva \n")
  })

  # Ottieni la memoria usata dopo la fattorizzazione di Cholesky
  memoria_finale <- mem_used() #usare memory.size() con windows

  # Calcola la memoria utilizzata dalla funzione
  memoria_utilizzata_chol <- memoria_finale - memoria_iniziale

  return(list(factor = factor,
    memoria_utilizzata_chol = memoria_utilizzata_chol))
}
```

Listing 3.4: Funzione `cholesky_decomposition`

La funzione denominata `solve_linear_system`, che richiede come argomento una matrice fattorizzata e il vettore b, risolve il sistema lineare corrispondente e restituisce la soluzione insieme alle informazioni sulla memoria utilizzata durante l'operazione.
Di seguito viene illustrato nel dettaglio il suo funzionamento:

- `memoria_iniziale <- mem_used()`: viene misurata la quantità di memoria utilizzata prima di risolvere il sistema lineare.
  Questo è utile per monitorare quanta memoria viene allocata prima del processo.

- `x <- solve(factor, b)`: risolve il sistema lineare utilizzando la funzione `solve()`.
  `factor` rappresenta la matrice ottenuta dalla fattorizzazione di Cholesky, e `b` è il vettore dei termini noti calcolato precedentemente.
  Nel caso in cui il processo termini senza errori viene restituito un messaggio di conferma.

- `memoria_finale <- mem_used()`: Dopo aver risolto il sistema, viene nuovamente misurata la quantità di memoria attualmente in uso da R e sottraendo questo valore alla memoria utilizzata prima del processo, è possibile visualizzare la memoria totale richiesta dalla funzione.

```
solve_linear_system <- function(factor, b) {
  # Ottieni la memoria usata prima della fattorizzazione di Cholesky
  memoria_iniziale <- mem_used()
```

```
  x <- solve(factor, b)

  # Ottieni la memoria usata dopo la fattorizzazione di Cholesky
  memoria_finale <- mem_used()
  cat("\n\nSistema Risolto\n\n")

  # Calcola la memoria utilizzata dalla funzione
  memoria_utilizzata_sistemaLin <- memoria_finale - memoria_iniziale

  return(list(x = x,
    memoria_utilizzata_sistemaLin = memoria_utilizzata_sistemaLin))
}
```

Listing 3.5: Funzione `solve_linear_system`

La funzione `compute_relative_error` accetta un vettore x e calcola l'errore relativo rispetto a un vettore di riferimento x_esatto, restituendo il risultato come output. L'errore relativo è una metrica comune utilizzata in problemi numerici per valutare quanto una soluzione approssimata si discosta dalla soluzione esatta, permettendo così di capire quanto accurata sia la soluzione restituita dal processo.

```
compute_relative_error <- function(x) {
  n <- length(x)
  x_esatto <- rep(1, n)
  errore_relativo <- norm(x - x_esatto,
    type = c("2")) / norm(x_esatto, type = c("2"))
  return(errore_relativo)
}
```

Listing 3.6: Funzione `compute_relative_error`

3.1.3 Documentazione e piano di supporto

La documentazione delle librerie `pryr`, `here` e `R.matlab` sono disponibili sui seguenti siti web:
https://cran.r-project.org/web/packages/pryr/
https://cran.r-project.org/web/packages/here/ e
https://cran.r-project.org/web/packages/R.matlab/index.html
Ogni sito web sopra elencato contiene informazioni dettagliate sull'installazione, sull'utilizzo dei metodi e delle funzioni che presenta tale libreria.
Gli autori, ripetitivamente Hadley Wickham, Kirill Müller e Henrik Bengtsson, sono coloro che si occupano anche di mantenere aggiornata le librerie insieme ai loro

collaboratori.
I siti dispongono anche di una voce in cui sono inserite tutte le versioni precedenti della libreria confermando i continui aggiornamenti a cui è sottoposta.
Inoltre, in una sezione presente nei rispettivi siti, è possibile riportare eventuali bug o problemi che gli utilizzatori riscontrano tenendo quindi monitorati eventuali anomalie da risolvere.

3.1.4 Matrici testate

All'interno del programma sono state testate tutte le matrici richieste dalla specifica del progetto.
Tuttavia, non tutte vengono supportate dalle librerie da noi utilizzate in quanto quelle di dimensioni maggiori o con una percentuale di elementi non nulli alta, esauriscono velocemente le risorse del sistema.
Qui di seguito è mostrata una tabella che elenca tutte le matrici da noi testate affiancate da un indicatore indicante l'esito (positivo o negativo) del processo di fattorizzazione e risoluzione del sistema lineare:

Matrici	Stato	Dimensione (in MB)
ex15.mat	✅	0,555
shallow_water1.mat	✅	2,263
apache2.mat	✅	8,302
parabolic_fem.mat	✅	13,116
G3_circuit.mat	✅	13,833
cfd1.mat	✅	14,164
cfd2.mat	✅	23,192
StocF-1465.mat	Out of memory ❌	178,368
Flan_1565.mat	Out of Memory ❌	292,858

Figura 3.1: Matrici supportate da R

3.2 Implementazione in Java - Libreria EJML

Nel campo dell'algebra lineare computazionale, la risoluzione efficiente di sistemi lineari associati a matrici sparse, simmetriche e definite positive è un problema di grande rilevanza.
In questo contesto, una parte del progetto è stata sviluppata basandosi sulla libreria `open-source` **EJML** (Efficient Java Matrix Library). Essa è una libreria scritta interamente in Java e rilasciata con licenza **Apache v2**, progettata per fornire strumenti chiari e concisi per la manipolazione di matrici e vettori reali o complessi, sia densi che sparsi.
Offre un'ampia gamma di funzionalità matematiche, tra cui la risoluzione di sistemi lineari, il calcolo di autovalori e autovettori, la decomposizione QR, la fattorizzazione LU, la decomposizione di Cholesky e la fattorizzazione SVD.
Gli obiettivi principali di *EJML* sono:

1. Efficienza computazionale e gestione ottimizzata della memoria per matrici di varie dimensioni.

2. Accessibilità sia per principianti che per esperti nel campo dell'algebra lineare.

Il codice sorgente del progetto è disponibile all'indirizzo GitHub: https://github.com/EmanueleDubini/ProgettoCalcoloScientifico_EJML.

3.2.1 Descrizione delle librerie utilizzate

La libreria EJML offre una vasta gamma di funzionalità per l'algebra lineare, tra cui la risoluzione di sistemi lineari associati a matrici sparse, il calcolo degli autovalori e degli autovettori di una matrice, la fattorizzazione LU per la risoluzione di sistemi lineari e la decomposizione di Cholesky per matrici simmetriche definite positive.
Di seguito sono riportate tutte le funzionalità per l'algebra lineare implementate dalla libreria EJML.

Capabilities

Linear Algebra Capabilities

	Dense Real	Fixed real	Dense Complex	Sparse Real
Basic Arithmetic	X	X	X	X
Element-Wise Ops	X	X	X	X
Transpose	X	X	X	X
Determinant	X	X	X	X
Norm	X		X	X
Inverse	X	X	X	X
Solve m=n	X		X	X
Solve m>n	X		X	X
LU	X		X	X
Cholesky	X		X	X
QR	X		X	X
QRP	X			
SVD	X			
Eigen Symm	X			
Eigen General	X			

Figura 3.2: Funzionalità algebra lineare libreria EJML

Alcune di queste funzionalità sono state utilizzate all'interno del programma per risolvere il sistema lineare $Ax=b$ associato alle matrici prese in esame.
Sul sito ufficiale della libreria è possibile trovare la sezione "*Example Code*" che fornisce una tabella contenente vari esempi di codice per i più comuni problemi di algebra lineare risolvibili utilizzando la libreria in questione, oltre all'elenco di tutte le funzioni supportate dalla libreria stessa, rendendola quindi di facile implementazione.

3.3 Descrizione del Codice Java

In questa sezione è contenuta la parte del progetto da noi implementata in **Java** per studiare ed analizzare l'implementazione del metodo di **Cholesky** per la risoluzione

di sistemi lineari associati a matrici sparse, simmetriche e definite positive.
L'intero codice è contenuto all'interno del file `Main.java` tramite il quale è possibile eseguire quanto implementato.
Anche in questo caso il codice è stato sviluppato con l'obiettivo di valutare le prestazioni e i risultati ottenuti tramite l'elaborazione di ciascuna matrice analizzata.
Osservando il **tempo** necessario per risolvere il sistema lineare tramite la decomposizione di Cholesky, l'**errore relativo** tra la soluzione calcolata `x` e la soluzione esatta `xe` e l'utilizzo di **memoria RAM** necessaria per risolvere il sistema è possibile utilizzare queste metriche per successivi confronti con altre piattaforme disponibili e programmi da noi sviluppati.

3.3.1 File `Main.java`

Di seguito è riportato il codice completo dell'applicazione Java, da cui è possibile eseguire l'intero programma.
Durante l'esecuzione del programma Java, vengono eseguiti i seguenti passaggi:

- All'inizio dell'esecuzione del programma vengono inizializzati gli `Arraylist`: `MatrixName`, `Size`, `MemoryPre`, `MemoryPost`, `MemoryDiff`, `Time` ed `Error`, utilizzati successivamente per memorizzare i risultati ottenuti in seguito all'elaborazione di ciascuna matrice. Inoltre, mediante un array di oggetti di tipo `File`, vengono lette e memorizzate le matrici da analizzare successivamente.

Listing 3.7: Inizializzazione degli `Array` e memorizzazione delle matrici

```
1  // Inizializzazione degli array per memorizzare i dati di ogni
       matrice
2      ArrayList<String> MatrixName = new ArrayList<>();
3      ArrayList<Long> Size = new ArrayList<>();
4      ArrayList<Long> MemoryPre = new ArrayList<>();
5      ArrayList<Long> MemoryPost = new ArrayList<>();
6      ArrayList<Long> MemoryDiff = new ArrayList<>();
7      ArrayList<Double> Time = new ArrayList<>();
8      ArrayList<Double> Error = new ArrayList<>();
9
10     // Percorso della cartella contenente le matrici
11     String path = "src/main/java/org/BDD/Matrici/";
12     File matriciFolder = new File(path);
13     // Lista di tutti i file nella cartella Matrici
```

```
14    File[] files = matriciFolder.listFiles();
```

- Viene eseguito un ciclo `for` che, ad ogni iterazione, elabora una singola matrice specificando, tramite un apposita stampa a *console*, quale matrice è attualmente utilizzata.
- Tramite il metodo `readFromFile` offerto dalla libreria `MAT File Library` viene importata la matrice sparsa, simmetrica e definita positiva da analizzare nell'iterazione corrente memorizzandola all'interno della matrice `A` di tipo `DMatrixSparseCSC`.

Listing 3.8: Importazione delle matrici dal formato `.mat` a matrici sparse

```
1
2     // Importazione della matrice sparsa simmetrica e definita
          positiva A
3     Sparse value = Mat5.readFromFile(file.getAbsolutePath())
4         .getStruct("Problem")
5         .getSparse("A");
6
7     DMatrixSparseCSC A = new DMatrixSparseCSC(value.getNumRows(),
          value.getNumCols());
8     A = Mat5Ejml.convert(value, A);
```

- Vengono salvati all'interno dell'Arraylist `Size` la dimensione dell'intero file contenete la matrice analizzata e il suo nome all'interno di `MatrixName`.

Listing 3.9: Importazione delle matrici dal formato `.mat` a matrici sparse

```
1
2     File matriceA = new File("src/main/java/org/BDD/Matrici/" +
          file.getName());
3     long dimensionA = matriceA.length();
4     Size.add(dimensionA);
5     System.out.println("Dimensioni matrice A: " + dimensionA / (1024
          * 1024) + " MB");
6
7     System.out.println("\n---> Inizio elaborazione matrice " +
          file.getName() + " \n");
```

```
8     MatrixName.add(file.getName());
```

- Vengono eseguiti specifici controlli per determinare se la matrice analizzata nell'iterazione corrente sia definita positiva e simmetrica stampando a *console* il tempo impiegato per i controlli. Questi controlli sono resi possibili dai metodi `isPositiveDefinite(A) e isSymmetric(A, 1e-8)` offerti dalla classe `MatrixFeatures_DSCC` appartenente alla libreria `EJML`.

Listing 3.10: Controllo matrice positivita' e simmetricita' della matrice

```
1
2     if (MatrixFeatures_DSCC.isPositiveDefinite(A)) {
3         System.out.println("La matrice A e' definita positiva");
4     } else {
5         System.err.println("La matrice A non e' definita positiva");
6         System.exit(1);
7     }
8
9     if (MatrixFeatures_DSCC.isSymmetric(A, 1e-8)) {
10        System.out.println("La matrice A e' simmetrica");
11    } else {
12        System.err.println("La matrice A non e' simmetrica");
13        System.exit(1);
14    }
```

- Viene misurato l'utilizzo iniziale della memoria e registrato nell'array `MemoryPre`.

Listing 3.11: Controllo matrice positivita' e simmetricita' della matrice

```
1
2     // Misura la memoria iniziale
3     long memoriaIniziale = Runtime.getRuntime().totalMemory() -
          Runtime.getRuntime().freeMemory();
4     MemoryPre.add(memoriaIniziale);
```

- Viene calcolata la dimensione della matrice analizzata e viene creato un vettore `B` definito come prodotto della matrice `A` e un vettore composto da soli uno.

Listing 3.12: Calcolo della dimensione della matrice analizzata e definizione del vettore B

```
1
2    // Calcola la dimensione della matrice A
3    //la matrice e' simmetrica quindi n = m
4    int n = A.numCols;
5
6    // Crea il vettore B di modo che x = [1,1,...,1]
7    // tmp e' un vettore colonna, va creato il vettore di tutti 1 e
         poi moltiplicato per la matrice A per creare B
8    DMatrixSparseCSC tmp = new DMatrixSparseCSC(n, 1);
9    for (int j = 0; j < n; j++) {
10       tmp.set(j, 0, 1);
11   }
12
13   // Moltiplicazione tra il vettore di tutti 1 tmp e la matrice A,
         il risultato viene salvato in B
14   //B = A*tmp eseguito in multi-thread
15   DMatrixSparseCSC B = CommonOps_MT_DSCC.mult(A, tmp, null);
```

- Viene creato un vettore `x` rappresentante il vettore soluzione per poi memorizzare l'istante di inizio del processo di risoluzione del sistema lineare.

Listing 3.13: Definizione del vettore soluzione `x`

```
1
2    // Crea il vettore x
3    //x e' un vettore colonna con tutti gli elementi uguali a zero
4    DMatrixSparseCSC x = new DMatrixSparseCSC(n, 1);
5
6    // Registra il tempo d'inizio
7    long startTime = System.currentTimeMillis();
```

- Tramite la libreria `LinearSolverFactory_DSCC` viene istanziato un oggetto che permette la risoluzione di sistemi lineari mediante la decomposizione di Cholesky. Inoltre, specificando tramite il metodo `setA()` la matrice `A` su cui applicare la decomposizione di Cholesky, viene risolto il sistema lineare `Ax = B` memorizzando il risultato all'interno del vettore `x`.

Listing 3.14: Risoluzione del sistema lineare

```
1
2     LinearSolverSparse<DMatrixSparseCSC, DMatrixRMaj> solver =
          LinearSolverFactory_DSCC.cholesky(FillReducing.NONE);
3             solver.setA(A);
4             solver.solveSparse(B, x);
5
6     System.out.println("\nSistema risolto\n");
```

- Terminata la risoluzione del sistema lineare, si prosegue con il calcolo della memoria utilizzata e del tempo impiegato per la risoluzione.

  Listing 3.15: Calcolo memoria utilizzata e tempo impiegato dal programmma per la risoluzione

```
1
2     // Misura la memoria finale
3     long memoriaFinale = Runtime.getRuntime().totalMemory() -
          Runtime.getRuntime().freeMemory();
4     MemoryPost.add(memoriaFinale);
5
6     // Calcola la memoria utilizzata
7     long memoriaUtilizzata = memoriaFinale - memoriaIniziale;
8     MemoryDiff.add(memoriaUtilizzata);
9     System.out.println("Memoria iniziale: " + (memoriaIniziale /
          (1024F * 1024F)) + " MB");
10    System.out.println("Memoria finale: " + (memoriaFinale / (1024F
          * 1024F)) + " MB");
11    System.out.println("Memoria utilizzata nella risoluzione: " +
          (memoriaUtilizzata / (1024F * 1024F)) + " MB");
12
13
14    // Registra il tempo di fine
15    long stopTime = System.currentTimeMillis();
16    // Calcola il tempo impiegato in millisecondi
17    double elapsedTimeSeconds = (stopTime - startTime) / 1000.0;
18    Time.add(elapsedTimeSeconds);
19    System.out.println("Tempo di esecuzione: " + elapsedTimeSeconds
          + " s");
```

- Viene calcolato l'errore relativo tra la soluzione calcolata `x` e la soluzione esatta `xe` per poi salvarlo all'interno dell'Arraylist `Error` e stamparlo a *console*.

Listing 3.16: Calcolo errore relativo soluzione

```
1
2     //----------CALCOLO ERRORE RELATIVO----------
3     // Definisco vettore soluzione xe esatta di modo che xe =
          [1,1,....,1]
4     DMatrixSparseCSC xe = new DMatrixSparseCSC(n, 1);
5     for (int z = 0; z < n; z++) {
6         xe.set(z, 0, 1);
7     }
8
9     // Calcolo norma || x -xe||
10    DMatrixSparseCSC x_diff_xe = new DMatrixSparseCSC(n, 1);
11    x_diff_xe = CommonOps_DSCC.add(1, x, -1, xe, x_diff_xe, null,
          null);
12    double norm_diff = NormOps_DSCC.normF(x_diff_xe);
13    // Calcolo norma || xe||
14    double norm_xe = NormOps_DSCC.normF(xe);
15    //Calcolo errore relativo
16    double relative_error = norm_diff / norm_xe;
17
18    Error.add(relative_error);
19    System.out.println("Errore relativo: " + relative_error);
```

- Una volta completato il ciclo `for` precedente, i risultati ottenuti relativi alla risoluzione del sistema lineare associato a ciascuna delle matrici sparse, simmetriche e definite positive, vengono registrati all'interno di un file `CSV` sfruttando la funzione `write` da noi implementata. Il nome del file viene generato in base al sistema operativo corrente, e vengono salvati i valori di `MatrixName`, `Size`, `MemoryPre`, `MemoryPost`, `MemoryDiff`, `Time` ed `Error`. Il salvataggio di questi dati permetterà di generare dei grafici, mediante un apposito script in Python, permettendo il confronto tra i dati ottenuti con le prestazioni offerte da altri linguaggi di programmazione `open-source`.

3.3.2 Documentazione e Supporto della Libreria EJML

La libreria *EJML* è stata concepita seguendo un approccio modulare che consente agli sviluppatori di integrare specifiche parti della libreria in base alle loro esigenze. Questo design modulare facilita la manutenzione e l'aggiornamento del codice nel tempo, in quanto permette di modificare o aggiornare singole parti della libreria senza dover manipolare l'intero sistema.
Essendo un progetto `open-source`, *EJML* beneficia del supporto attivo di una vasta comunità di sviluppatori i quali contribuiscono al progetto segnalando eventuali bug, proponendo nuove funzionalità e migliorando la documentazione. Questo ciclo di feedback e collaborazione aiuta a mantenere la libreria aggiornata e a risolvere tempestivamente eventuali problemi riscontrabili.

La pagina GitHub di EJML, accessibile al seguente link: https://github.com/lessthanoptimal/ejml, offre una vasta gamma di informazioni e risorse per tutti gli utenti. Al suo interno è possibile trovare, oltre ad una panoramica iniziale del progetto, l'accesso al manuale utente e alla JavaDoc, . Inoltre, vengono fornite istruzioni dettagliate su come integrare EJML nei propri progetti Maven in aggiunta al codice sorgente completo della libreria.

Per quanto riguarda il manuale della libreria, disponibile al link: https://ejml.org/wiki/index.php?title=Manual, è strutturato in quattro sezioni principali:

- The Basics (Nozioni Base): Fornisce un'introduzione a EJML, compresa la lista delle operazioni standard e le funzionalità principali della libreria. Descrive come sviluppare applicazioni utilizzando EJML, fornisce una sezione FAQ e una bacheca come punto di riferimento per le domande degli utenti.

- Tutorials (Tutorial): Offre tutorial mirati a vari problemi di algebra lineare e le decomposizioni di matrici. Questi tutorial dimostrano l'utilizzo di diverse parti di EJML aiutando gli utenti a risolvere diversi problemi di algebra lineare.

- Example Code (Esempi di Codice): Fornisce esempi concreti di problemi di algebra lineare implementati utilizzando EJML. Vengono indicati i vari approcci disponibili tramite le diverse interfacce offerte dalla libreria, inclusi esempi di utilizzo di matrici sparse, filtro di Kalman, analisi delle componenti principali e molto altro.

- External References (Riferimenti Esterni): Consiglia materiale di lettura per gli utenti interessati ad approfondire i concetti di algebra lineare e l'utilizzo della libreria EJML.

3.3.3 Matrici testate e supportate

All'interno del programma sono state testate tutte le matrici che sono state richieste dalla consegna del progetto, ma non tutte vengono elaborate; le matrici di dimensione maggiore (o con una percentuale di zeri maggiore) esauriscono le risorse del sistema.
Di seguito viene riportata una tabella che contiene tutte le matrici testate e l'indicazione sul supporto delle stesse:

Nomi Matrici	Stato	Dimensione (in KB)
ex15.mat	✅	555
shallow_water1.mat	✅	2263
apache2.mat	Out of memory ❌	8302
parabolic_fem.mat	Out of memory ❌	13116
G3_circuit.mat	Out of memory ❌	13833
cfd1.mat	✅	14164
cfd2.mat	✅	23192
StocF-1465.mat	Out of memory ❌	178368
Flan_1565.mat	Out of Memory ❌	292858

Figura 3.3: Matrici supportate da Java

3.4 Implementazione in Python - Libreria SciKit-sparse

Come mostrato nelle sezioni precedenti, il programma è stato progettato per analizzare una serie di matrici sparse conducendo un'analisi delle prestazioni relative alla risoluzione di sistemi lineari Ax=b mediante la decomposizione di Cholesky . L'obiettivo è di valutare l'efficienza computazionale di tali operazioni utilizzando diverse librerie `open-source` come la libreria scientifica SciPy e SciKit-Sparse disponibili in Python.
É importante sottolineare che, durante l'esecuzione del codice implementato, le matrici elaborate saranno manipolate sotto forma di matrici sparse evitando la conversione in formato denso e l'eccessivo utilizzo delle risorse del sistema. Le librerie `open-source` sono state scelte per la loro affidabilità, flessibilità e diffusione nella comunità di Python. Di fatto, consentono di gestire matrici sparse e risolvere sistemi lineari in modo rapido ed efficiente, evitando l'esaurimento delle risorse computazionali.

Il codice sorgente del progetto sviluppato in Python è disponibile all'indirizzo GitHub: https://github.com/dbancora/ProgettoCalcoloScientifico_Python.

3.4.1 Descrizione delle librerie utilizzate

SciPy è una libreria `open-source` per il calcolo scientifico e tecnico in Python. Essa si basa su un'altra libreria chiamata *NumPy*, che offre supporto sempre nello stesso ambito. **SciPy** è un'importante risorsa per la comunità scientifica e tecnica poiché offre strumenti potenti e flessibili per l'analisi e la manipolazione di dati scientifici.

Per quanto riguarda la libreria **SciKit-sparse**, anch'essa di tipo `open-source`, fornisce un'interfaccia che permette agli utenti di utilizzare le funzionalità offerte dal package Cholmod. **SciKit-sparse** è progettata per operare con matrici sparse ed offre una vasta gamma di funzionalità per manipolare e risolvere problemi di algebra lineare avanzati. Uno dei suoi package principali è *sksparse.cholmod* che consente di eseguire operazioni algebriche come la decomposizione di matrici sparse e risolvere sistemi lineari ad esse associati.
Come detto precedentemente, le matrici sparse sono matrici con molti elementi nulli diffuse in problemi di grandi dimensioni in cui risulta inefficiente rappresentare le matrici in forma densa. **SciKit-sparse** offre un'implementazione efficiente per la gestione e la manipolazione di tali matrici, consentendo di eseguire operazioni

algebriche complesse in modo rapido ed efficiente. La libreria può essere utilizzata in diversi ambiti, come l'analisi dei dati, l'apprendimento automatico (machine learning), l'elaborazione delle immagini e molti altri. Essa fornisce agli utenti una gamma di strumenti per risolvere problemi che coinvolgono matrici sparse, consentendo di ottimizzare le prestazioni e ridurre la complessità computazionale delle operazioni svolte.

3.4.2 Presentazione del Codice Sorgente

Il programma da noi implementato esegue diverse operazioni su una serie di file `.mat` contenenti matrici sparse e definite positive, con lo scopo di calcolare il tempo di esecuzione, la memoria utilizzata e l'errore relativo ottenuti durante la decomposizione di Cholesky e la risoluzione di un sistema lineare $Ax = b$. Nello specifico, osservando il codice sorgente, inizialmente viene caricata una matrice caricata da un file in formato `.mat` tramite la funzione `load_matrix_from_file(filename)` e viene salvata in una variabile.
In seguito, il programma verifica se la matrice letta è sparsa e simmetrica utilizzando la funzione `is_symmetric(A)`.
Una volta caricata la matrice, viene creato il vettore dei termini noti b (`create_b_vector(A)`) affinché la soluzione `x` del sistema lineare $Ax = b$ sia un vettore composto interamente da 1. Il vettore `b` viene definito come il risultato tra il prodotto della matrice A ed un vettore composto da tutti 1.
Una volta creato il vettore `b` e caricato la matrice, il programma prosegue eseguendo la decomposizione di Cholesky sfruttando la funzione `cholesky_decomposition(A)` la quale, al suo interno, presenta anche le righe di codice che permettono di calcolare la memoria utilizzata durante il processo di risoluzione. Di seguito viene riportato il codice:

Listing 3.17: Decomposizione di Cholesky

```
1
2   def cholesky_decomposition(A):
3       # Ottieni la memoria usata prima della fattorizzazione di Cholesky
4       memoria_iniziale = None
5
6       process = psutil.Process()
7       memoria_iniziale = process.memory_info().rss / (1024 * 1024)
8
9       # Calcola la fattorizzazione di Cholesky
```

```
        try:
            factor = cholmod.cholesky(A)
            print("La matrice A e' definita positiva")


        # Cattura l'eccezione nel caso in cui la matrice non sia definita
            positiva
        except cholmod.CHOLMODNotPositiveDefinite:
            print("La matrice A non e' definita positiva")

        # Ottieni la memoria usata dopo la fattorizzazione di Cholesky
        memoria_finale = None

        process = psutil.Process()
        memoria_finale = process.memory_info().rss / (1024 * 1024)

        # Calcola la memoria utilizzata dalla funzione
        memoria_utilizzata_chol = memoria_finale - memoria_iniziale

        return factor, memoria_utilizzata_chol
```

Eseguita la decomposizione, è ora possibile applicare la funzione `solve_linear_system(factor, b)` per risolvere il sistema lineare considerando, anche in questo caso, la memoria utilizzata.
Di seguito viene riportato il codice della funzione appena descritta:

Listing 3.18: Risoluzione del sistema lineare

```

    def solve_linear_system(factor, b):
        # Ottieni la memoria usata prima della risoluzione del sistema
        memoria_iniziale = None

        process = psutil.Process()
        memoria_iniziale = process.memory_info().rss / (1024 * 1024)

        x = factor(b)

        #DEBUG: print(x)

```

```
        # Ottieni la memoria usata dopo la fattorizzazione di Cholesky
        memoria_finale = None

        process = psutil.Process()
        memoria_finale = process.memory_info().rss / (1024 * 1024)

        print("\n\nSistema Risolto\n")
        # Calcola la memoria utilizzata dalla funzione
        memoria_utilizzata_sistemaLin = memoria_finale - memoria_iniziale

        return x, memoria_utilizzata_sistemaLin
```

Una volta terminate le operazioni di decomposizione e di risoluzione del sistema lineare, il programma prosegue calcolando l'errore relativo mediante il confronto tra la soluzione x ottenuta dalla funzione di risoluzione del sistema lineare e il vettore $xEsatto$, definito come $x_Esatto = [1\ 1\ 1\ 1\ 1\ 1\ ...]$. Queste operazioni sono descritte nella funzione `compute_relative_error(x)`.

Una volta che il programma esegue tutte le funzioni sopra descritte applicandole alla prima matrice da analizzare, proseguirà eseguendo gli stessi passaggi per ogni matrice presente nella directory principale. Tutti i risultati ottenuti vengono salvati in un array che successivamente sarà inserito all'interno di un file `.csv`, creato al termine dell'esecuzione del programma, in modo da poter essere esportato ed utilizzato per la generazione dei grafici di confronto e rappresentazione dei risultati ottenuti.

3.4.3 Documentazione e Piano di supporto

SciPy

La documentazione ufficiale di **SciPy** è disponibile sul sito web https://docs.scipy.org/doc/scipy/ e contiene informazioni dettagliate riguardanti l'installazione e l'utilizzo dei vari moduli e funzioni.
Per quanto riguarda il sito della libreria, è suddiviso in diverse sezioni tra cui: una guida introduttiva per gli utenti meno esperti, una guida per utenti che desiderano approfondire le funzionalità offerte dalla libreria, una sezione di riferimento tecnico dedicata agli sviluppatori che intendono utilizzare **SciPy** ed infine una guida realizzata per i collaboratori interessati a contribuire al progetto `open-source`. La

libreria è costantemente aggiornata e sviluppata dagli autori permettendo inoltre, ai suoi utilizzatori, di segnalare bug e migliorarne l'efficienza con lo scopo di fornire una piattaforma di calcolo di alto livello per la comunità scientifica.

SciKit-sparse

La documentazione della libreria `open-source` utilizzata in questa sezione del progetto, è disponibile al sito https://scikit-sparse.readthedocs.io/en/latest/. Essa è divisa in varie sezioni come:

- *Overview*, che contiene link utili per l'installazione della libreria, le informazioni generali e i contatti personali degli sviluppatori. Inoltre, sempre all'interno della stessa sezione sono specificati i requisiti necessari per il corretto funzionamento della libreria come NumPy, SciPy, Cython e Cholmod (incluso in SuiteSparse).
- *Sparse.cholmod*, contiene le informazioni per l'utilizzo dei metodi utili ad applicare la decomposizione di Cholesky su matrici sparse. É inoltre disponibile una sotto-sezione relativa ai possibili errori riscontrabili durante l'utilizzo della funzione stessa.
- *Changes*, contiene tutte le versioni della libreria che sono state distribuite: per ognuna di queste è possibile verificare il *changelog* delle modifiche apportate.

Il team di sviluppo di SciKit-sparse è composto da volontari provenienti da diverse organizzazioni e istituti di ricerca. Questi sviluppatori dedicano il loro tempo e le loro competenze per mantenere e migliorare la libreria, risolvere bug, implementare nuove funzionalità e garantire la stabilità e l'affidabilità di quest'ultima. Infine, osservando la pagina GitHub ufficiale della libreria, è possibile notare come essa sia aggiornata costantemente introducendo migliorie o correggendo eventuali anomalie presenti nel progetto.

3.4.4 Matrici testate e supportate

Anche per quanto riguarda le librerie Python abbiamo testato tutte le matrici, richieste dalla consegna, presenti nella SuiteSparse Matrix Collection osservando come le librerie `open-source` da noi implementate siano in grado di risolvere la maggior

parte dei sistemi lineari $Ax=b$ associati alle matrici sparse prese in considerazione. Di seguito viene riportata una tabella riassuntiva contenente tutte le matrici testate:

Matrici	Stato	Dimensione (in MB)
ex15.mat	✅	0,555
shallow_water1.mat	✅	2,263
apache2.mat	✅	8,302
parabolic_fem.mat	✅	13,116
G3_circuit.mat	✅	13,833
cfd1.mat	✅	14,164
cfd2.mat	✅	23,192
StocF-1465.mat	Out of memory ❌	178,368
Flan_1565.mat	Out of Memory ❌	292,858

Figura 3.4: Matrici supportate da Python

4
Risultati Ottenuti

In Questa sezione della relazione verranno analizzati i risultati ottenuti dall'esecuzione delle librerie introdotte nella sezione 2 e sezione 3 del documento.
Come anticipato precedentemente, gli aspetti cardine utilizzati come linee guida durante il confronto per valutare la migliore implementazione rispetto alle tre librerie e linguaggi analizzati sono:

- **Memoria utilizzata**: all'interno dei file CSV generati al termine dell'esecuzione di ciascun progetto sviluppato, è riportata la memoria utilizzata in `byte` dal sistema per effettuare la fattorizzazione delle matrice sparsa mediante la decomposizione di Cholesky e la memoria necessaria per la successiva risoluzione il sistema lineare $Ax = b$.
  É importante notare che la memoria misurata per ciascun programma testato considera esclusivamente la porzione di memoria RAM allocata per eseguire i due compiti appena descritti.

- **Tempo di esecuzione**: all'interno dei programmi implementati, il tempo di esecuzione viene misurato effettuando due misurazioni riferite unicamente ai secondi necessari per effettuare la decomposizione di Cholesky e risoluzione del sistema non considerando quanto viene impiegato dal sistema per allocare quanto necessario per la risoluzione del sistema.

Così come per la memoria utilizzata, anche il tempo di esecuzione viene salvato all'interno dello stesso file CSV.

- **Errore relativo**: al termine della risoluzione di ciascun sistema viene calcolato l'errore relativo tra la soluzione `x` ottenuta e il vettore esatto `xe`, definito come $x_e = [1\ 1\ 1\ 1\ 1\ 1\ ...]$. Anche in questo caso i risultati ottenuti vengono salvati all'interno del file CSV.

Per ogni linguaggio e sistema operativo, al termine dell'esecuzione di ciascun programma, viene generato uno specifico file CSV rinominato a seconda del linguaggio specifico e del sistema operativo utilizzato per l'esecuzione. Oltre a contenere al suo interno la memoria utilizzata, il tempo di esecuzione e l'errore relativo per la risoluzione di ciascun sistema, ogni file CSV ottenuto presenta anche il nome di ciascuna matrice analizzata e supportata dalla libreria, affiancata dalla dimensione della stessa (sempre in `byte`).

Questi file saranno successivamente analizzati al fine di condurre un confronto esaustivo e accurato mediante la generazione di opportuni grafici di confronto generati tramite uno script Python, da noi sviluppato, disponibile al seguente link GitHub: https://github.com/dbancora/ProgettoCalcoloScientifico_Grafici.
Lo script permette di generare grafici specifici per ciascuna delle tre metriche sopra descritte misurate per ogni matrice analizzata suddividendo ciascun grafico in base al linguaggio di programmazione utilizzato e mettendo a confronto i risultati ottenuti tra i due sistemi operativi tesati, ossia `Windows` e `Linux`. Vengono generati ulteriori grafici ognuno riferito ad una precisa metrica prestazionale ed ad uno specifico sistema operativo utilizzato, che confrontano i risultati ottenuti da ciascun linguaggio di programmazione preso in esame.

4.1 MATLAB

In questa sezione, vengono presentati i risultati ottenuti eseguendo il programma da noi sviluppato in MATLAB.
É importante sottolineare che le matrici supportate dal programma sono `ex15.mat`, `shallow_water1.mat`, `cfd1.mat` e `cfd2.mat`.
Le ulteriori matrici da elaborare: `Flan_1565.mat`, `Stocf-1465.mat`, `G3_circuit.mat`, `parabolic_fem.mat` e `apache2.mat` richieste dalla consegna del progetto non vengono analizzate in quanto, durante l'esecuzione della decomposizione di Cholesky, viene visualizzato l'errore *Out of Memory* esaurendo le risorse disponibili alla macchina virtuale utilizzata per i test.

Riportiamo di seguito le metriche prestazionali rilevate dall'esecuzione degli script in ambienti *Windows* e *Linux*:

MatrixName	Size	MemoryDiff	Time	Error
ex15.mat	568315	80953344	1.4177961	8.21380186712579e-07
shallow_water1.mat	2316415	379895808	3.3899692	3.07688798314864e-16
cfd1.mat	14503159	1281593344	6.769914	2.16641067660319e-13
cfd2.mat	23748539	2488291328	11.3603334	6.19326456426195e-13

Figura 4.1: Risultati ottenuti da MATLAB su Windows

MatrixName	Size	MemoryDiff	Time	Error
ex15.mat	568315	5230592	0.60678	8.21380186712579e-07
shallow_water1.mat	2316415	378171392	3.586064	3.07688798314864e-16
cfd1.mat	14503159	1251893248	8.704733	2.16641067660319e-13
cfd2.mat	23748539	2488057856	15.274341	6.19326456426195e-13

Figura 4.2: Risultati ottenuti da MATLAB su Linux

Procediamo ora all'analisi e al confronto della memoria utilizzata per eseguire il processo di decomposizione di Cholesky e risoluzione del sistema lineare $Ax=b$.
A tal proposito, a supporto delle nostre analisi, viene inserito di seguito un grafico a barre che mostra la memoria necessaria per eseguire le operazioni sopra descritte, confrontando le risorse utilizzate da *Windows* con quelle utilizzate da *Linux*:

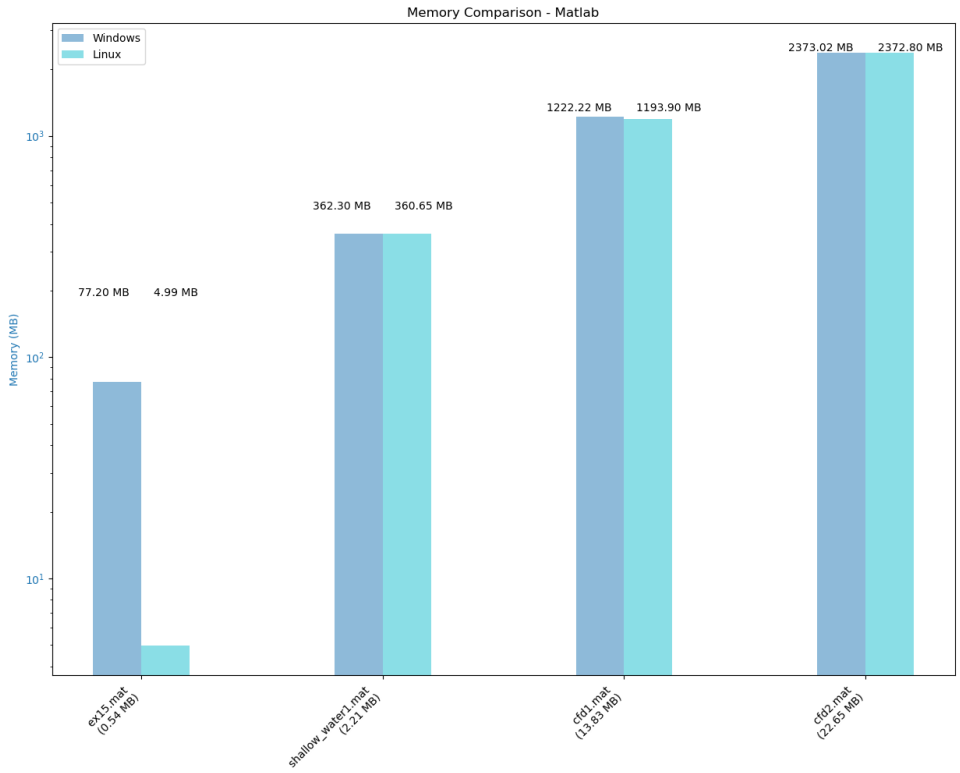

Figura 4.3: Grafico memoria occupata da MATLAB

Per quanto riguarda il grafico, notiamo che la quantità di memoria utilizzata è espressa in scala logaritmica in base 10, il che consente un chiaro confronto tra i risultati mostrati e mette in evidenza le differenze significative nella quantità di memoria richiesta da ciascuna matrice.
Questo aspetto è particolarmente utile per normalizzare efficacemente i risultati ottenuti valutando visualmente la quantità di memoria utilizzata da MATLAB durante la risoluzione dei sistemi lineari e facilitando il confronto tra i due sistemi operativi utilizzati. Analizzando quanto espresso dal grafico è possibile notare come non siano presenti grosse differenze tra i due sistemi operativi testati anche se per quanto riguarda la matrice `ex15.mat`, che presenta dimensioni minori rispetto alle altre, in

ambiente Windows viene allocata una maggiore quantità di risorse per eseguire la decomposizione e la risoluzione del sistema lineare. All'aumentare della dimensione delle matrici analizzate, la differenza delle risorse utilizzate dai due sistemi operativi si riduce, fino ad arrivare alla matrice `cfd2.mat` in cui è presente una differenza irrisoria.
Inoltre è possibile notare come la dimensione in `MegaByte` di ciascuna matrice e la quantità di memoria utilizzata per la risoluzione del sistema associato ad essa siano direttamente proporzionali.

Il secondo grafico da noi realizzato ha l'obbiettivo di confrontare, per ciascuna matrice analizzata, il tempo impiegato per eseguire la decomposizione di Cholesky e la successiva risoluzione del sistema nei due ambienti testati.

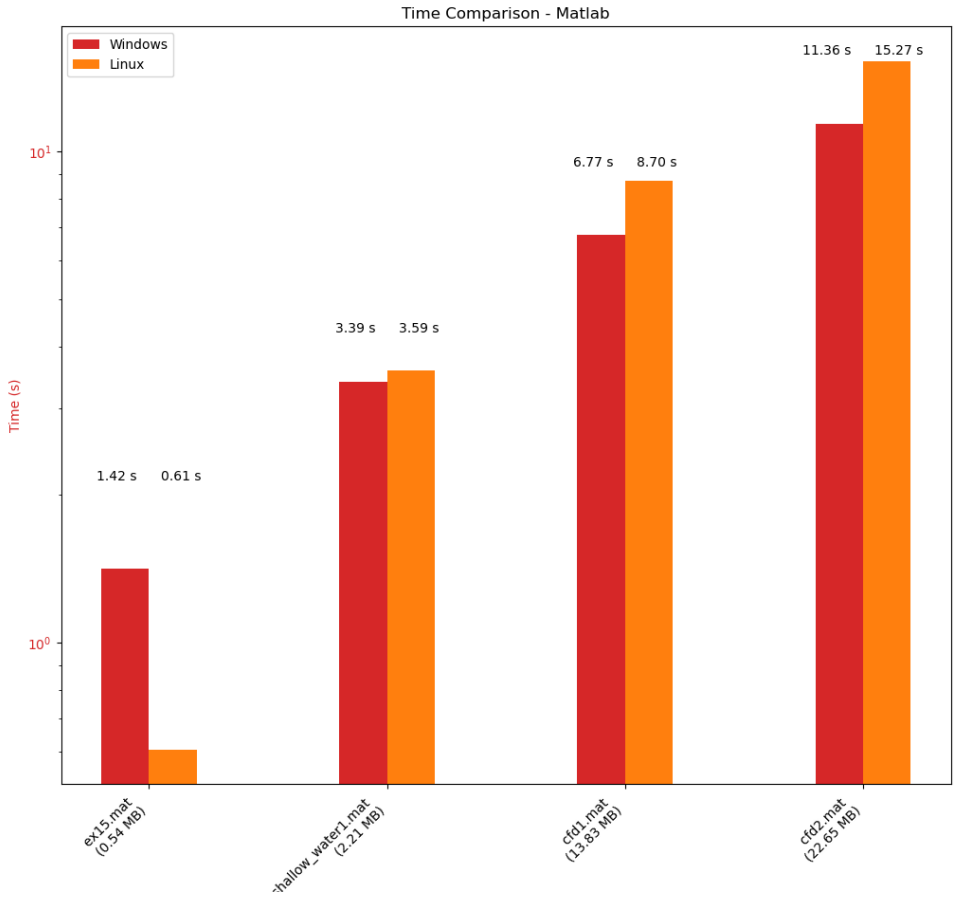

Figura 4.4: Grafico tempo impiegato da MATLAB

I dati mostrano come il tempo impiegato per risolvere il sistema lineare, in ambiente Linux, sia leggermente maggiore rispetto all'esecuzione in Windows per tutte le matrici analizzate ad eccezione della `ex15.mat`. Queste differenze sono da imputare alle diverse modalità con cui i due sistemi operativi gestiscono l'allocazione e la successiva liberazione della memoria allocata. Anche in questa occasione possiamo affermare che la dimensione in `MegaByte` di ciascuna matrice analizzata è direttamente proporzionale al tempo impiegato per la risoluzione del sistema lineare associato.

Il terzo grafico realizzato in ambiente MATLAB ha l'obbiettivo di confrontare, per i due sistemi operativi adottati e per ciascuna matrice analizzata, l'errore relativo calcolato tra la soluzione del sistema lineare `x` con la soluzione esatta `xe`.

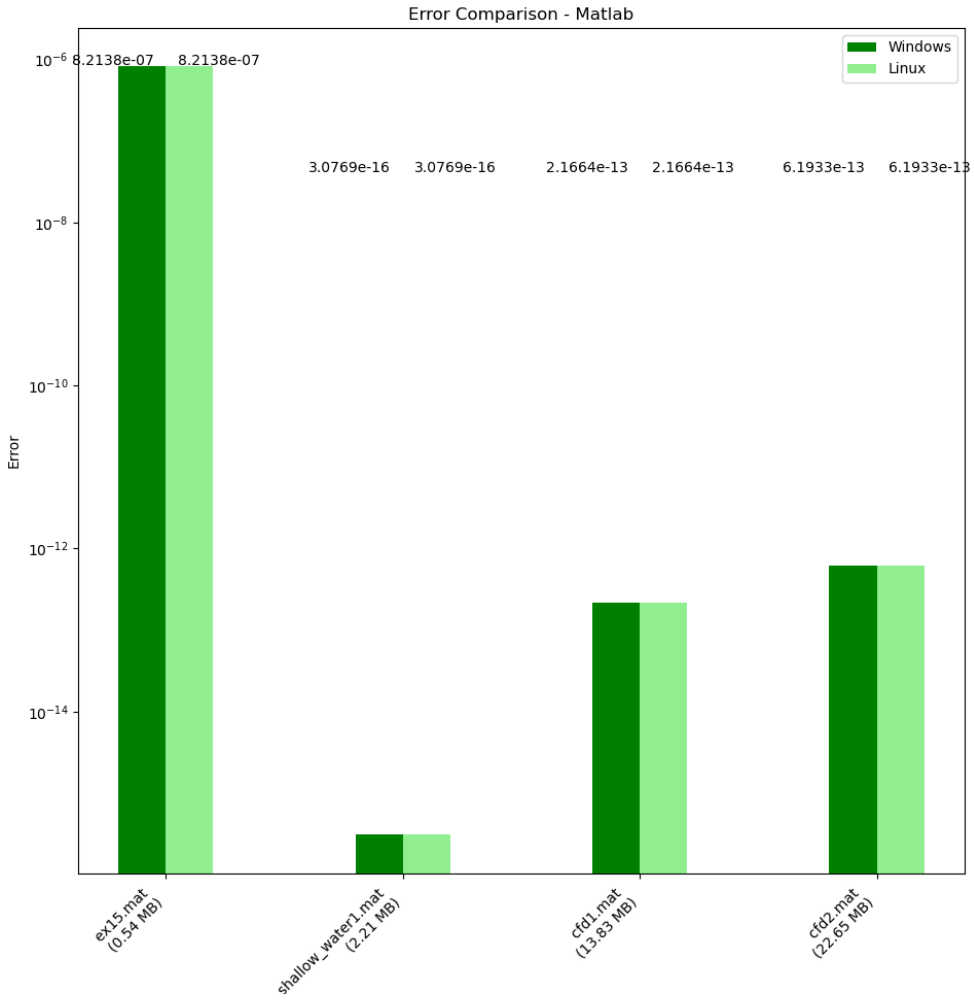

Figura 4.5: Grafico errore relativo ottenuto da MATLAB

É possibile osservare che l'andamento dell'errore associato alla risoluzione del sistema lineare per le matrici `shallow_water1.mat`, `cfd1.mat` e `cfd2.mat` è direttamente proporzionale alla dimensione delle stesse. Questa osservazione non è vera per la matrice `ex15.mat` in quanto, pur avendo dimensioni minori rispetto alle matrici sopra citate, presenta un errore relativo maggiore. Questo non è dovuto alla dimensione della matrice stessa ma è legato alla difficoltà con cui si risolve il sistema lineare definito dal numero di condizionamento. Osservando questo dato associato a ciascuna matrice analizzata:

MatrixName	ConditionNumber
ex15.mat	8.61e+12
shallow_water1.mat	3.51e+00
apache2.mat	3.07e+06
parabolic_fem.mat	2.10e+05
G3_circuit.mat	1.47e+07
cfd1.mat	3.39e+05
cfd2.mat	1.52e+06

Figura 4.6: Numeri di condizionamento delle matrici

notiamo che, in corrispondenza della `ex15.mat`, il numero di condizionamento assume un valore maggiore rispetto a tutte le altre matrici.
Anche nel caso della `shallow_water1.mat` notiamo che accade la stessa situazione in cui, essendo la matrice con il numero di condizionamento minore tra tutte le matrici analizzate, ad essa è riferito l'errore relativo più basso nella risoluzione del sistema lineare.

In conclusione possiamo affermare che non vi siano significative differenze tra i dati ottenuti utilizzando la piattaforma MATLAB con sistemi operativi Windows e Linux. Le uniche differenze rilevanti si riscontrano considerando la memoria utilizzata e i tempi di esecuzione notando che il sistema operativo Windows è in grado di utilizzare meno risorse ed impiegare tempo minore con matrici di grandi dimensioni. A differenza di Linux che si dimostra più conveniente in seguito all'utilizzo di matrici di piccole dimensioni.

4.2 R

In questa sezione, vengono presentati i risultati ottenuti eseguendo il programma da noi sviluppato in R. É importante sottolineare che non tutte le matrici sono supportate dal programma: `StocF-1465.mat` e `Flan_1565.mat`, richieste dalla consegna del progetto, non vengono analizzate in quanto, durante l'esecuzione della decomposizione di Cholesky, viene visualizzato l'errore *Out of Memory* esaurendo le risorse disponibili alla macchina virtuale utilizzata per i test.

Riportiamo di seguito le metriche prestazionali rilevate dall'esecuzione degli script `R` in ambienti *Windows* e *Linux*:

MatrixName	Size	MemoryDiff	Time	Error
ex15.mat	568315	3071984	0.559454917907715	7.52205594858616e-07
shallow_water1.mat	2316415	30924608	0.932286977767944	2.3005075780259e-16
apache2.mat	8501019	2150241104	162.815289020538	7.87130687601758e-11
parabolic_fem.mat	13430627	450420096	9.95148181915283	2.78163926009668e-12
G3_circuit.mat	14164910	2319963072	125.393973827362	8.20721162362439e-12
cfd1.mat	14503159	432792184	16.3143348693848	3.32987475656585e-12
cfd2.mat	23748539	889196024	76.2307629585266	6.38703575723732e-12

Figura 4.7: Risultati ottenuti da R su Windows

MatrixName	Size	MemoryDiff	Time	Error
ex15.mat	568315	3210624	0.728338003158569	7.52205594858616e-07
shallow_water1.mat	2316415	30913984	1.22969079017639	2.3005075780259e-16
apache2.mat	8501019	2150241104	159.061979293823	7.87130687601758e-11
parabolic_fem.mat	13430627	450419888	10.7573566436768	2.78163926009668e-12
G3_circuit.mat	14164910	2319963072	153.620629310608	8.20721162362439e-12
cfd1.mat	14503159	432791976	19.3153836727142	3.32987475656585e-12
cfd2.mat	23748539	889195816	65.8250815868378	6.38703575723732e-12

Figura 4.8: Risultati ottenuti da R su Linux

Come per l'ambiente MATLAB sono stati realizzati tre grafici, ognuno associato ad una singola metrica prestazionale considerata e che permette di comparare l'andamento dei due sistemi operativi adottati

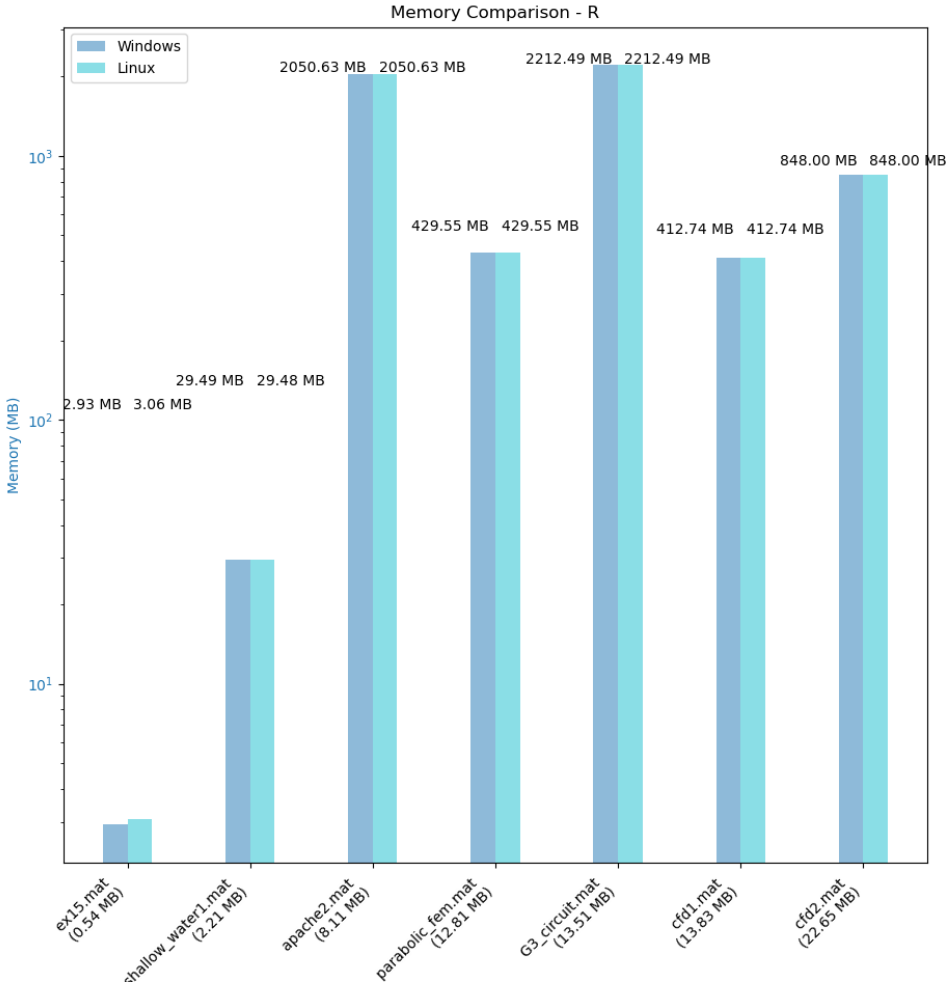

Figura 4.9: Grafico memoria occupata da R

Grazie ai dati raccolti dal grafico possiamo dedurre come sia presente una andamento crescente tra la dimensione della matrice analizzata in `MegaByte` e la quantità di memoria utilizzata durante la risoluzione del sistema.
L'unica eccezione è data della memoria utilizzata durante l'elaborazione delle matrici `apache2.mat` e `G3_circuit.mat` dove le risorse necessarie sono molto più elevate rispetto alle altre matrici.

Questo comportamento può essere giustificato dalla tabella sottostante in cui viene mostrato che il numero di elementi diversi da zero, all'interno delle due matrici appena citate, è anch'esso elevato e superiore a tutte le altre matrici analizzate.

MatrixName	NonZeroA
ex15.mat	9.87e+04
shallow_water1.mat	3.28e+05
apache2.mat	4.82e+06
parabolic_fem.mat	3.67e+06
G3_circuit.mat	7.66e+06
cfd1.mat	1.83e+06
cfd2.mat	3.09e+06

Figura 4.10: Numero di entrate diverse da 0 per ogni matrice analizzata

Per quanto riguarda la differenza tra la memoria utilizzata confrontando i due sistemi operativi invece, possiamo affermare che non siano presenti differenze significative e preferire un sistema operativo rispetto all'altro, non comporterebbe un significativo aumento delle prestazioni.

Osserviamo ora il grafico relativo ai tempi di elaborazione necessari per ciascun sistema operativo impiegato nell'analisi:

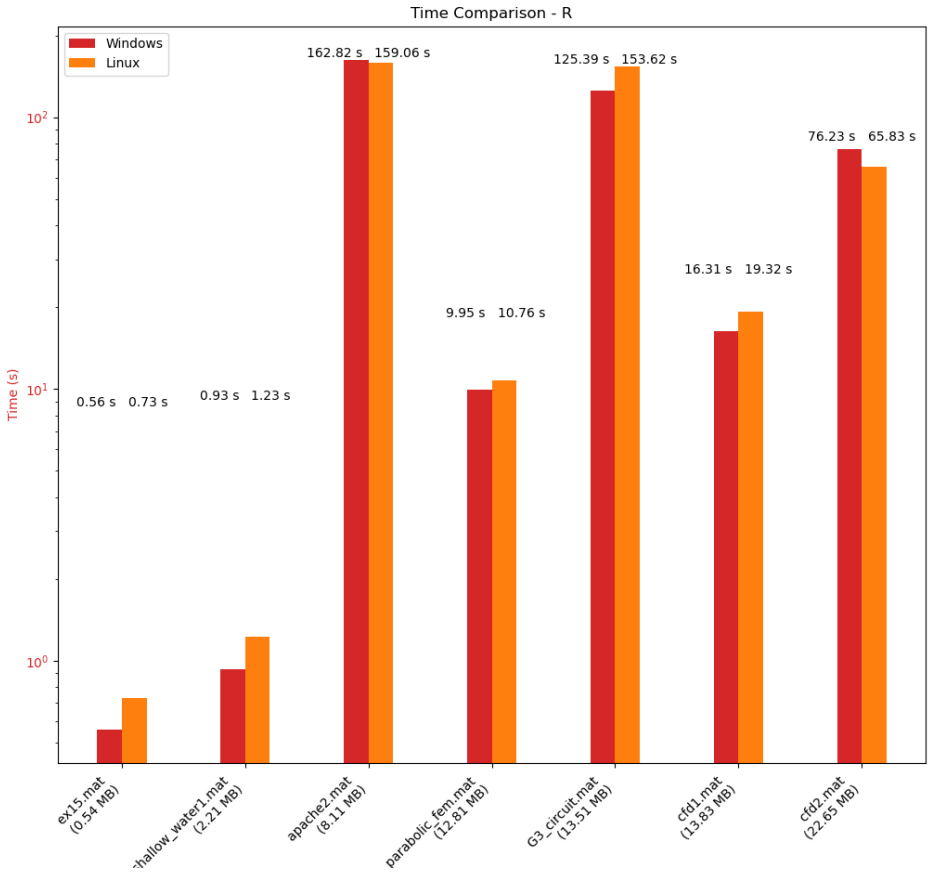

Figura 4.11: Grafico tempo impiegato da R

Come per la memoria, anche considerando il tempo di elaborazione come metrica prestazionale, si osserva un andamento crescente tra i secondi impiegati nella risoluzione del sistema associato alla matrice considerata e la dimensione in `MegaByte` della matrice stessa.
Anche in questo caso possiamo notare come le due matrici `apache2.mat` e `G3_circutit.mat` si distinguano da tale andamento necessitando tempi maggiori per la loro elaborazione pur avendo dimensioni minori rispetto a matrici di dimensioni maggiori come `parabolic_fem.mat` e `cfd1.mat`.
Questi valori anomali possono essere giustificati osservando la Figura 4.10.
Per quanto riguarda il confronto tra i due ambienti software notiamo come in Windows si osserva che, per la maggior parte delle matrici analizzate, i tempi risultano essere sensibilmente inferiori permettendoci di consigliarlo rispetto al sistema operativo Linux.

Per quanto riguarda il grafico mostrante l'errore relativo associato alla risoluzione di ciascun sistema lineare:

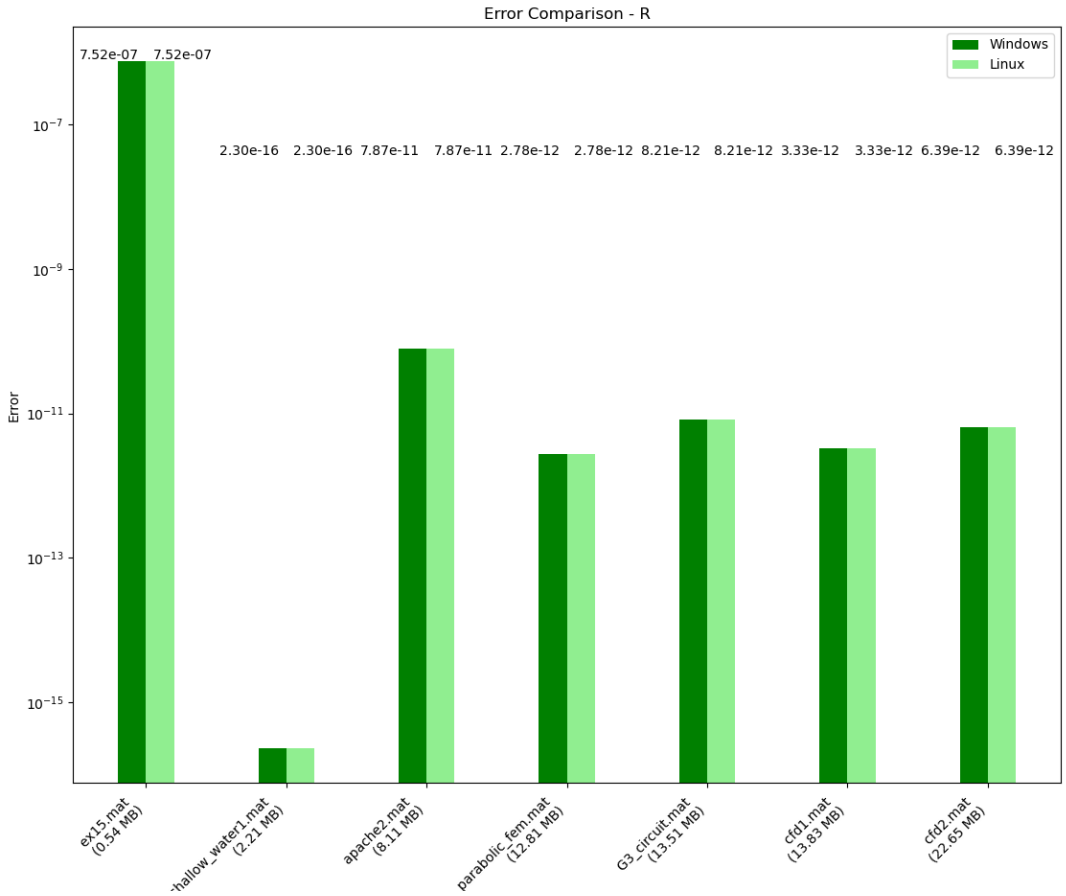

Figura 4.12: Grafico tempo impiegato da R

Per quanto riguarda l'errore relativo misurato tra la soluzione del sistema `x` e la soluzione esatta `xe`, si nota un alto valore associato alla matrice `ex15.mat` probabilmente dovuto al suo mal condizionamento, come già riscontrato in figura 4.6.
Per tutte le altre matrici abbiamo ottenuto un errore accettabile, ponendo particolare attenzione alla matrice `shallow_water1.mat` in cui si è registrato l'errore relativo minore dovuto al numero di condizionamento minimo rispetto alle altre matrici analizzate.

Osservando i risultati ottenuti in ambiente `R` confrontati su Windows e Linux, questi hanno mostrato un errore relativo e una memoria occupata simili.
La sostanziale differenza risiede nel tempo di esecuzione misurato: Windows sembra impiegare meno secondi per eseguire le operazioni richieste per la maggior parte delle matrici analizzate.

4.3 Java

In questa sezione, vengono presentati i risultati ottenuti eseguendo il programma da noi sviluppato in Java.
É importante sottolineare che, come in MATLAB, le matrici supportate dal programma sono `ex15.mat`, `shallow_water1.mat`, `cfd1.mat` e `cfd2.mat`.
Le ulteriori matrici da elaborare: `Flan_1565.mat`, `Stocf-1465.mat`, `G3_circuit.mat`, `parabolic_fem.mat` e `apache2.mat` richieste dalla consegna del progetto non vengono analizzate in quanto, durante l'esecuzione della decomposizione di Cholesky, viene visualizzato l'eccezione *Out of Memory* esaurendo le risorse disponibili alla macchina virtuale utilizzata per i test.

Riportiamo di seguito le metriche prestazionali rilevate durante l'esecuzione del programma `Java` in ambienti *Windows* e *Linux*:

MatrixName	Size	MemoryPre	MemoryPost	MemoryDiff	Time	Error
cfd1.mat	14503159	26370736	1012436776	986066040	49.444	4.525125569752032E-11
cfd2.mat	23748539	40559528	1968587672	1928028144	100.139	2.1927722216622213E-11
ex15.mat	568315	3530560	10854800	7324240	0.007	8.007992883722316E-7
shallow_water1.mat	2316415	8939184	302562024	293622840	6.437	2.9863681564206695E-16

Figura 4.13: Risultati ottenuti da Java su Windows

MatrixName	Size	MemoryPre	MemoryPost	MemoryDiff	Time	Error
shallow_water1.mat	2316415	11589576	303454928	291865352	9.079	2.9863681564206695E-16
cfd2.mat	23748539	42210224	1970236304	1928026080	93.307	2.1927722216622213E-11
cfd1.mat	14503159	27319520	1014388288	987068768	60.476	4.525125569752032E-11
ex15.mat	568315	5180256	11156288	5976032	0.009	8.007992883722316E-7

Figura 4.14: Risultati ottenuti da Java su Linux

Anche per il seguente linguaggio di programmazione utilizzato sono stati realizzati i medesimi grafici come quanto mostrato nella sezione precedente. Proseguiamo ad

una loro attenta analisi suddividendoli sempre in base alla *memoria utilizzata*, *tempo di esecuzione* e *errore relativo*:

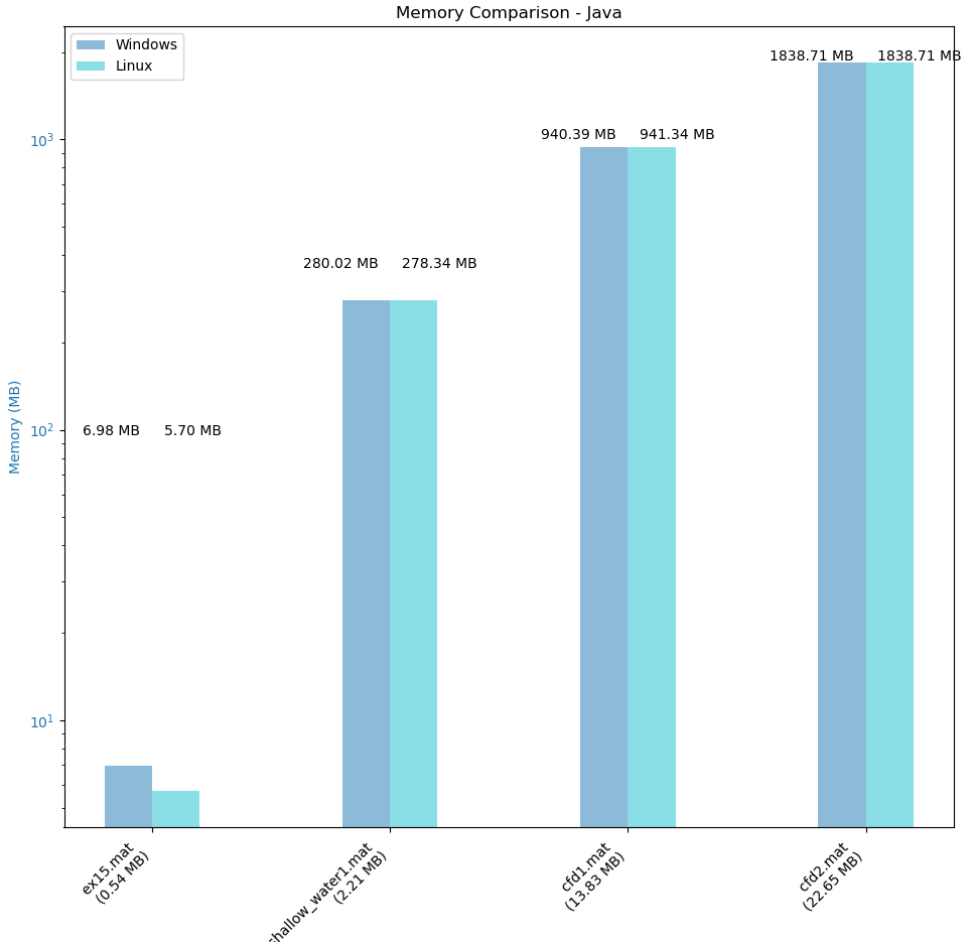

Figura 4.15: Grafico memoria occupata da Java

Osservando il grafico riferito alla memoria utilizzata con la libreria `EJML` in `Java`, è semplice dedurre che, all'aumentare della dimensione in `MegaByte` di ciascuna matrice analizzata, aumenta anche la memoria RAM necessaria per eseguire le operazioni richieste dalla consegna.
Al contrario di quanto riscontrato nell'ambiente MATLAB, osservando unicamente la memoria necessaria, la differenza tra i due sistemi operativi utilizzati è indifferente in quanto, sia in Windows che Linux, viene utilizzato lo stesso quantitativo di risorse per la risoluzione del sistema. L'unica eccezione la si può trovare osservando la matrice `ex15.mat` in cui sembra che l'ambiente Linux ottimizzi meglio le risorse

a disposizione, ma considerando lo standard di memoria RAM installata sulle macchine odierne la differenza è da considerarsi irrisoria.

Osserviamo ora il grafico relativo al tempo di esecuzione in ambiente `Java`:

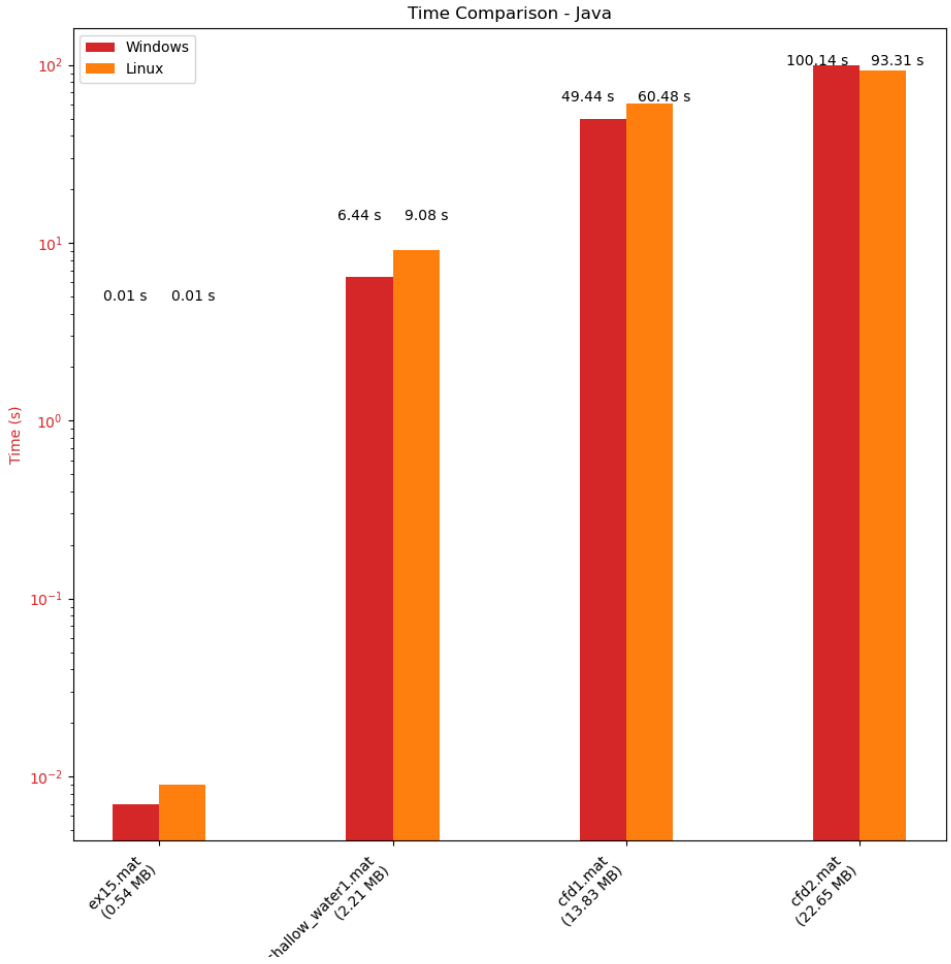

Figura 4.16: Grafico tempo impiegato da Java

Dai risultati ottenuti è facile osservare che all'aumentare della dimensione delle matrici analizzate, aumenta di conseguenza il tempo necessario per la loro elaborazione. Confrontando più a fondo i dati ottenuti notiamo che per quanto riguarda la matrice `ex15.mat` il tempo richiesto è pressoché uguale per i due ambienti Windows e Linux. Per quanto riguarda le due matrici `shallow_water1.mat` e `cfd1.mat` Windows ha la meglio mostrando tempi di calcolo minori nell'ordine dei secondi. Infine nel

caso della matrice `cfd2.mat` si ha un andamento inverso rispetto a quanto appena affermato con un tempo di elaborazione minore in Linux.

Per quanto riguarda il grafico dell'errore relativo:

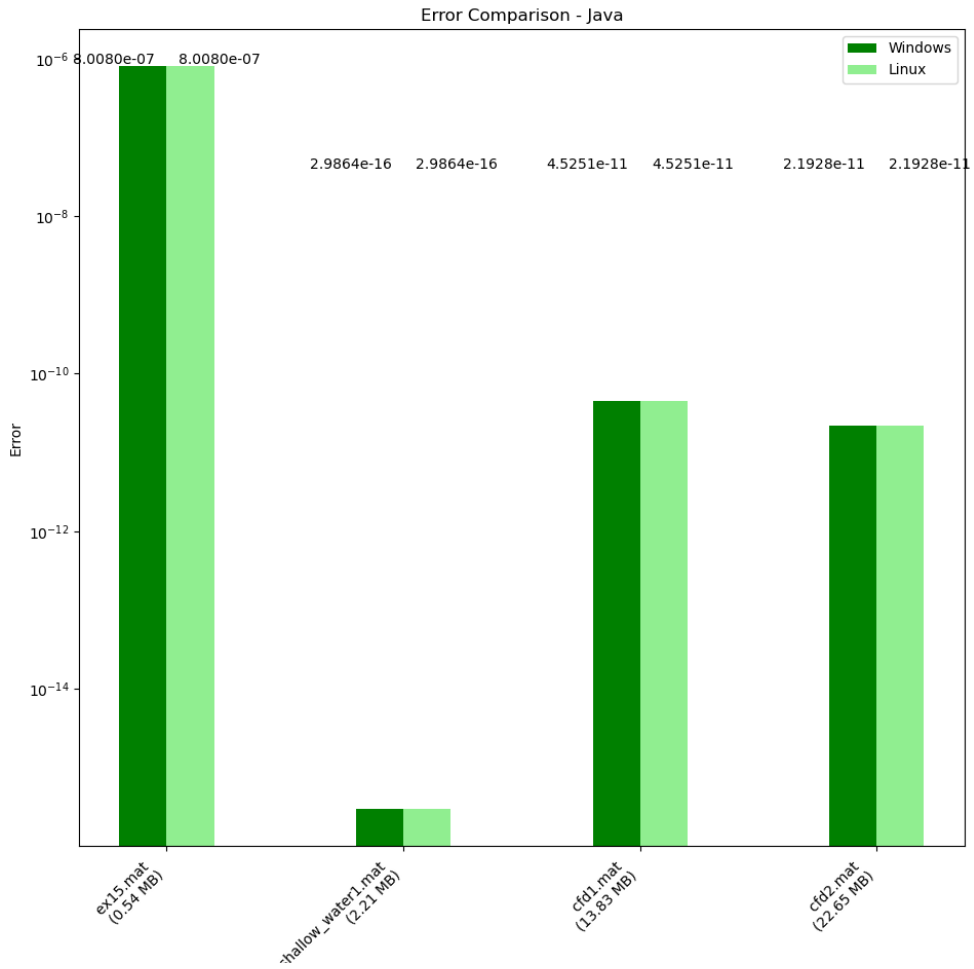

Figura 4.17: Grafico errore relativo ottenuto da Java

Notiamo che si manifesta un andamento simile a quanto riscontrato per l'ambiente di sviluppo MATLAB con la matrice `ex15.mat` che presenta l'errore relativo maggiore, causato da un numero di condizionamento alto, e la matrice `shallow_water1.mat` che riporta un errore relativo minimo rispetto alle altre matrici analizzate.
In merito al confronto tra le metriche prestazionali ottenute tra i due sistemi operativi Windows e Linux testati, possiamo affermare che i due ambienti non presentano

una differenza marcata ad eccezione della metriche riferita al tempo di elaborazione dove Windows sembra essere meglio ottimizzato per l'analisi di matrici di dimensioni minori. Per questo motivo consigliamo di utilizzare il sistema operativo Windows nel caso in cui si verifichi la necessità di elaborare sistemi lineari associati a matrici di piccole dimensioni.

4.4 Python

In questa sezione, vengono presentati i risultati ottenuti eseguendo il programma da noi sviluppato in Python. É importante sottolineare che non tutte le matrici sono supportate dal programma: `StocF-1465.mat` e `Flan_1565.mat`, richieste dalla consegna del progetto, non vengono analizzate in quanto, durante l'esecuzione della decomposizione di Cholesky, viene visualizzato l'errore *Out of Memory* esaurendo le risorse disponibili alla macchina virtuale utilizzata per i test.

Riportiamo di seguito le metriche prestazionali ottenute durante l'esecuzione del programma `Python` in ambienti *Windows* e *Linux*:

MatrixName	Size	MemoryDiff	Time	Error
ex15.mat	568315	3727360	0.017224788665771484	6.373039892025257e-07
shallow_water1.mat	2316415	37830656	0.38373517990112305	2.4191432592763325e-16
apache2.mat	8501019	1813708800	83.10970187187195	4.053547131931853e-11
parabolic_fem.mat	13430627	422174720	7.327727317810059	1.2186342978855653e-12
G3_circuit.mat	14164910	2090758144	91.9291090965271	2.4374166113795984e-12
cfd1.mat	14503159	362459136	14.238816499710083	3.8009881841220814e-13
cfd2.mat	23748539	721260544	46.6806366443634	6.903172394444132e-13

Figura 4.18: Risultati ottenuti da Python su Windows

MatrixName	Size	MemoryDiff	Time	Error
ex15.mat	568315	4640768	0.17230844497680664	6.373039892025257e-07
shallow_water1.mat	2316415	43823104	0.49481678009033203	2.4191432592763325e-16
apache2.mat	8501019	1430745088	50.31600284576416	2.6439542358305498e-11
parabolic_fem.mat	13430627	388530176	4.722765207290649	1.2186342978855653e-12
G3_circuit.mat	14164910	1103335424	20.841373443603516	2.741966359964529e-12
cfd1.mat	14503159	187654144	3.135990619659424	1.8334919784499647e-14
cfd2.mat	23748539	350425088	7.625463008880615	3.5281971859737356e-13

Figura 4.19: Risultati ottenuti da Python su Linux

Analizziamo di seguito i grafici realizzati come mostrato nelle sezioni precedenti. Anche in questo caso l'analisi prenderà in considerazione la *memoria utilizzata*, *tempo di esecuzione* e *errore relativo*:

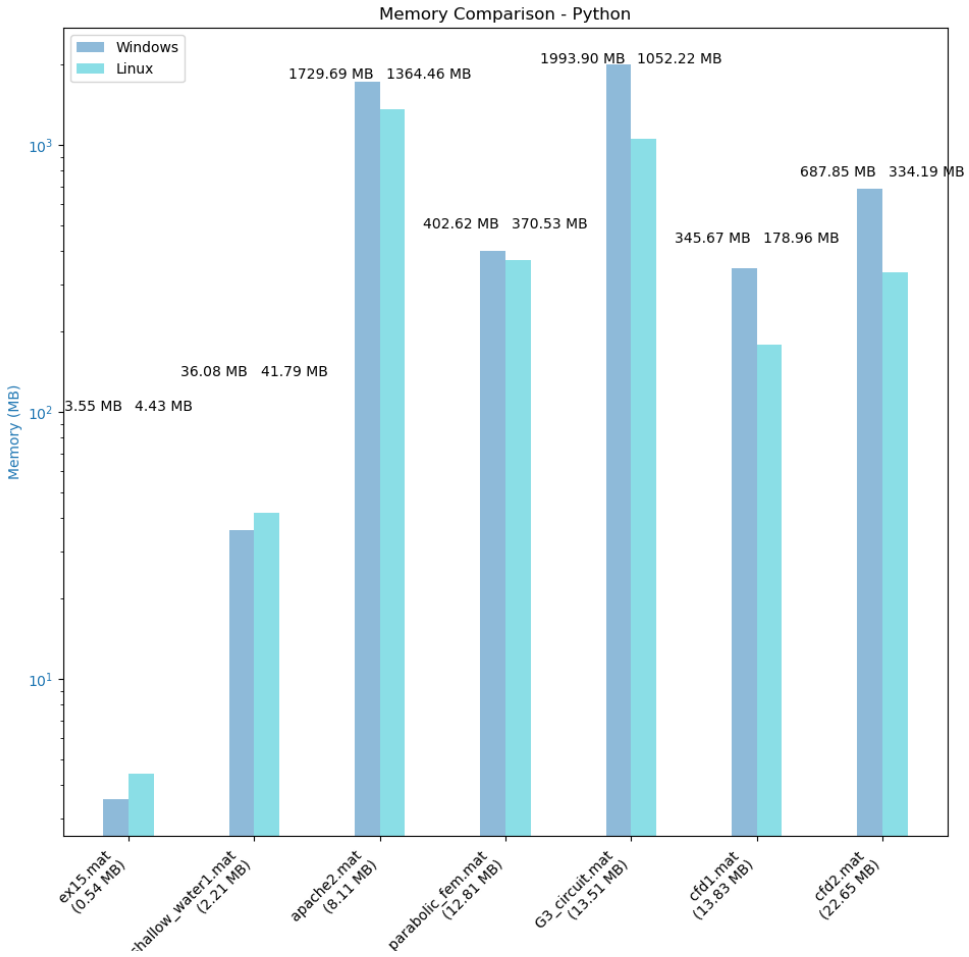

Figura 4.20: Grafico memoria occupata da Python

Il primo grafico realizzato permette di osservare la quantità di memoria utilizzata durante l'elaborazione di ciascuna matrice analizzata. in questo caso, a differenza dei risultati precedenti, non è ben definito un andamento crescente tra la dimensione della matrice in `MegaByte` e la quantità di memoria utilizzata per la risoluzione del sistema lineare associato. Come mostrato in figura 4.9, le matrici `apache2.mat` e `G3_circuit.mat` necessitano di un quantitativo superiore di memoria RAM rispetto alle altre matrici analizzate pur non appartenendo ai file `.mat` di dimensioni mag-

giori.
Per quanto riguarda il confronto tra i due sistemi operativi testati, è facile osservare come Windows utilizzi un quantitativo leggermente inferiore di memoria per l'elaborazione delle matrici `ex15.mat` e `shallow_water1.mat` a differenza di Linux. Di contro, quest'ultimo risulta essere ottimizzato nell'analisi di matrici di grandi dimensioni presentando un utilizzo di memoria per la loro elaborazione nettamente inferiore rispetto al sistema operativo Windows.
Osserviamo ora il grafico relativo al tempo di esecuzione in ambiente `Python`:

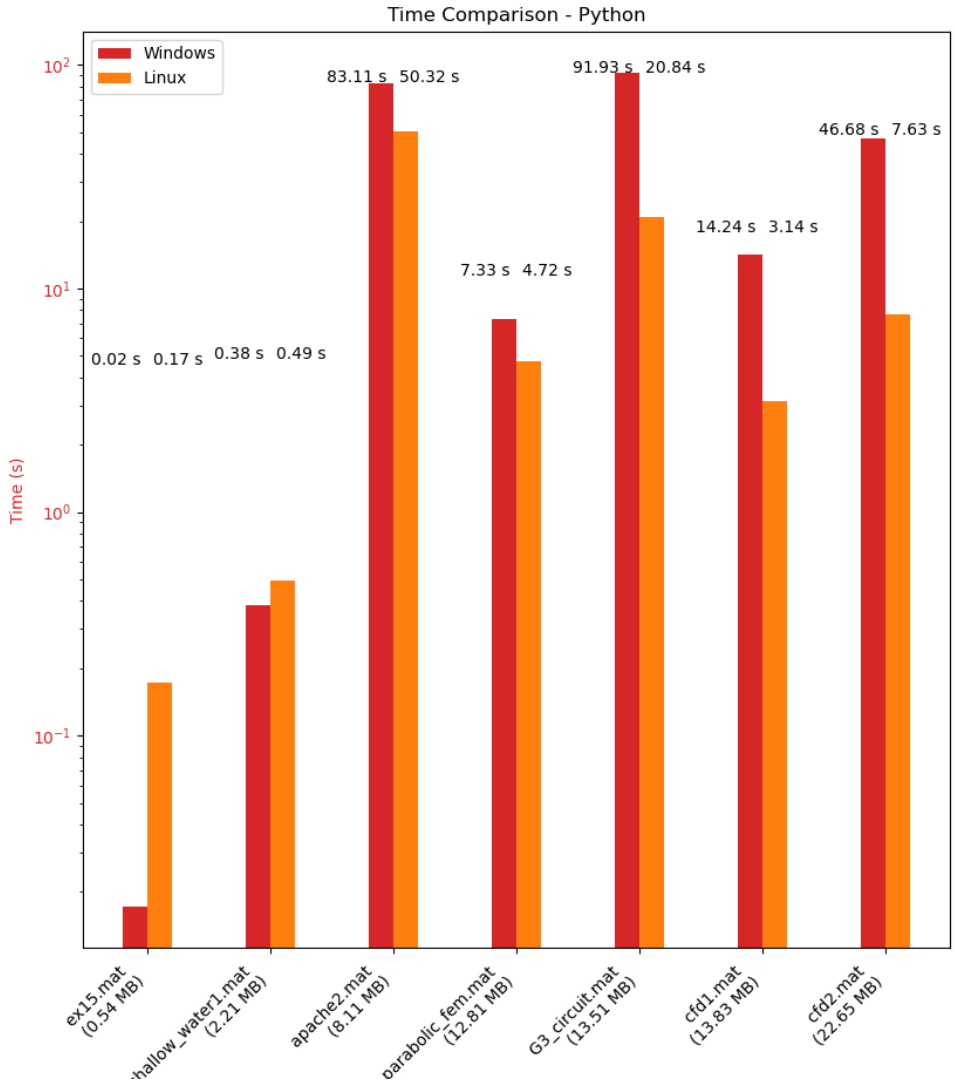

Figura 4.21: Grafico tempo impiegato da Python

Osservando attentamente il grafico che riporta i temi di elaborazione delle matrici prese in esame, si nota una netta discrepanza tra le prestazioni dei due sistemi operativi. Sebbene la differenza non sia marcata per i tempi delle matrici `ex15.mat` e `shallow_water1.mat`, in tutti gli altri casi Linux si dimostra molto più efficiente nella risoluzione dei sistemi lineari associati alle matrici prese in esame.

Per quanto riguarda il grafico dell'errore relativo:

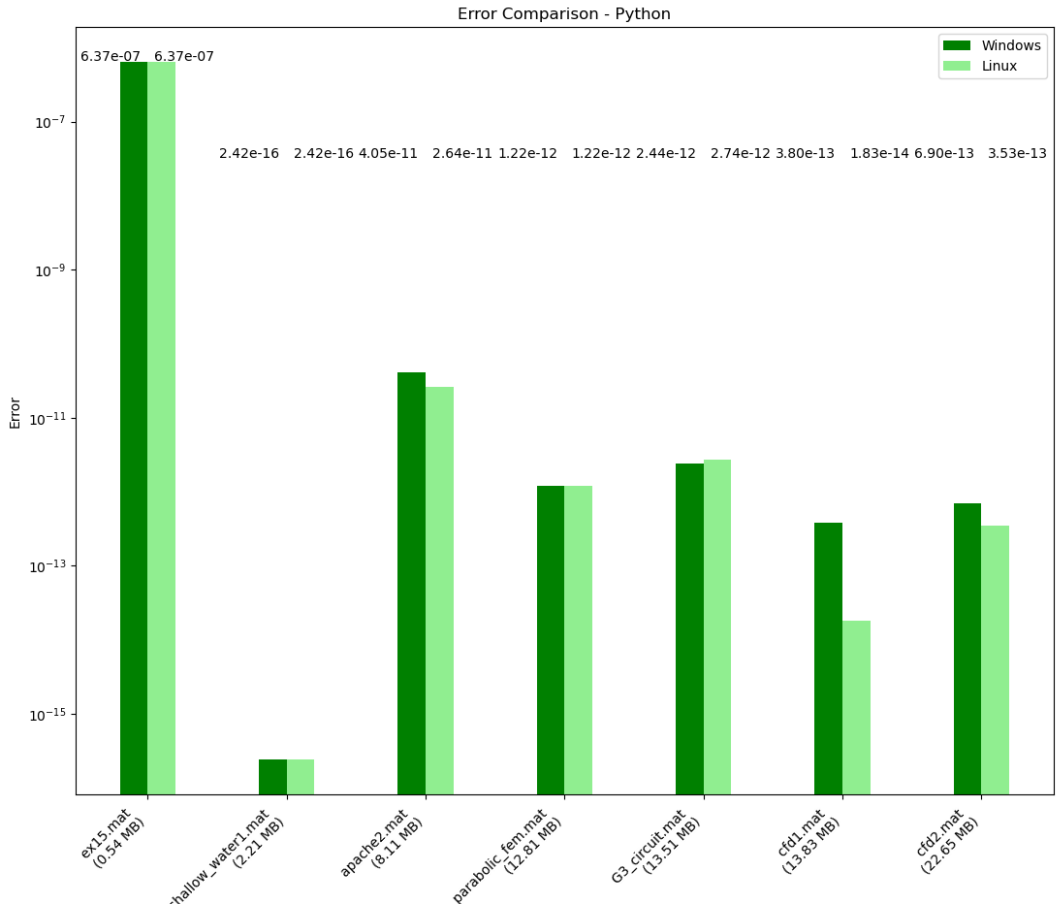

Figura 4.22: Grafico errore relativo ottenuto da Python

Anche in questo caso la matrice `ex15.mat` presenta l'errore relativo maggiore, sempre dovuto al suo mal condizionamento. il contrario accade per la matrice `shallow_water1.mat` dove l'errore relativo è minimo. Per quanto riguarda il confronto tra i due sistemi operativi, possiamo affermare che l'errore relativo rimane simile per tutte le matrici

ad eccezione delle matrici `cfd1.mat` e `cfd2.mat` che riportano un errore relativo inferiore utilizzando Linux.

In conclusione, date le discrepanze appena evidenziate, è facile concludere che il programma `Python` eseguito in ambiente Linux presenti prestazioni nettamente migliori in termini di memoria utilizzata e tempo necessario per l'esecuzione delle operazioni rispetto a Windows. Per quanto riguarda l'errore relativo, pur presentando una differenza minima tra i due ambienti, ancora una volta Linux offre prestazioni migliori.

4.5 Confronto tra i linguaggi di programmazione adottati

Al fine di ottenere un confronto completo delle metriche prestazionali ottenute dai quattro linguaggi di programmazione testati, all'interno di questa sezione analizziamo il loro comportamento utilizzando rispettivamente un sistema operativo Windows o Linux. All'interno dei grafici riportati di seguito è importante notare che le matrici `apache2.mat`, `parabolic_fem.mat` e `g3_circuit.mat` sono correttamente elaborate unicamente dai linguaggi di programmazione `Python` e `R` ma sono comunque riportati per avere un confronto completo del lavoro svolto.
I primi due grafici analizzati, uno per ciascun sistema operativo, confrontano il comportamento dei quattro linguaggi in base all'impiego di memoria necessaria per completare la risoluzione del sistema lineare mediante la decomposizione di Cholesky:

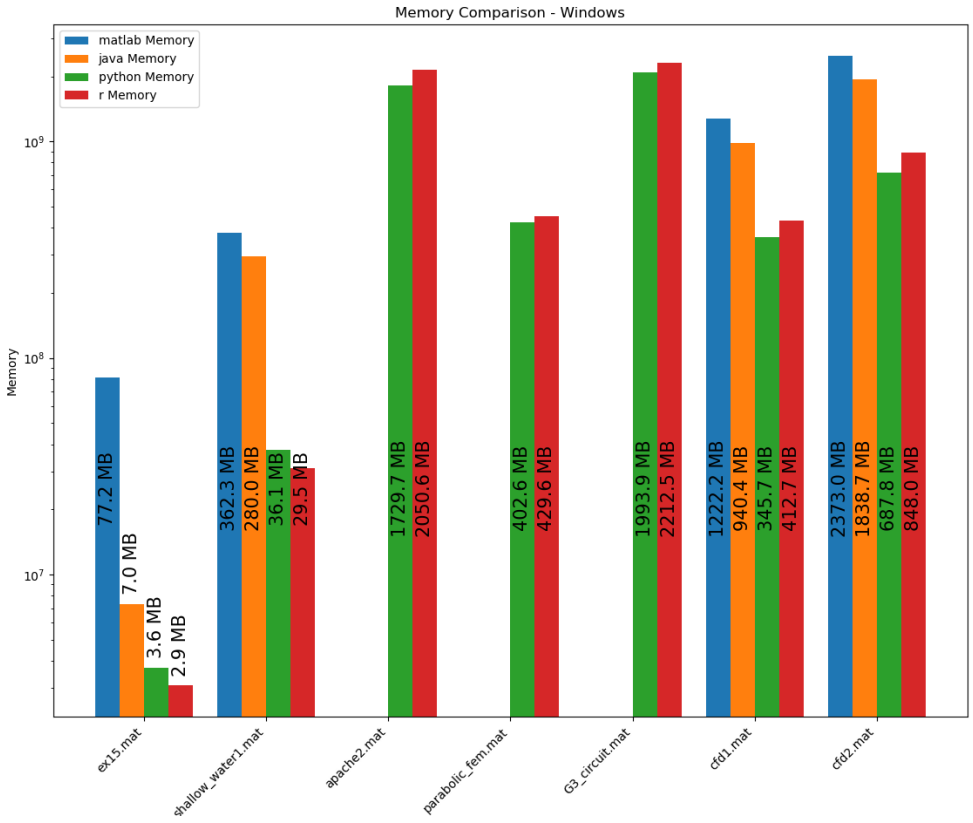

Figura 4.23: Grafico memoria utilizzata da Windows

Osservando il seguente grafico, che riporta il consumo di memoria utilizzata da Windows, è facile notare come MATLAB risulti essere il linguaggio più dispendioso in termini di risorse, sia nell'elaborare matrici di piccole che di grandi dimensioni. Il linguaggio Java, pur mostrando risultati leggermente migliori, tende a seguire l'andamento prestazionale dell'ambiente MATLAB. Per quanto riguarda i due linguaggi Python e R invece notiamo come la richiesta di memoria RAM sia inferiore rispetto ai due linguaggi appena citati. Nello specifico, in seguito all'elaborazione di matrici di piccole dimensioni, R si dimostra il linguaggio più efficiente. Invece, al crescere della grandezza dei file analizzati si inverte l'andamento appena descritto: Python risulta quindi essere il linguaggio più efficiente sul maggior numero di matrici in termini di memoria utilizzata.

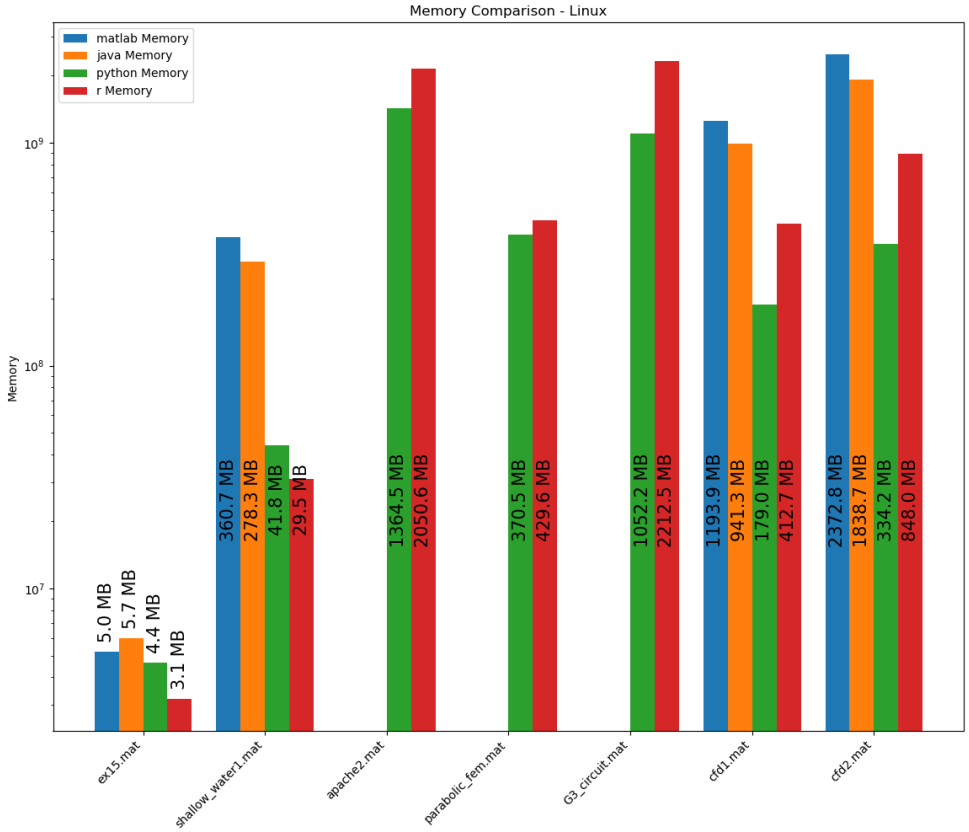

Figura 4.24: Grafico memoria utilizzata da Linux

Per quanto riguarda l'ambiente Linux l'andamento è molto simile a quanto osservato nel grafico precedente per i linguaggi MATLAB, Java e R. É importante notare

come la quantità di memoria utilizzata durante la risoluzione dei vari sistemi lineari in Python si riduce drasticamente soprattutto per l'analisi di matrici di grandi dimensioni facendolo risultare, ancora una volta, il linguaggio in grado di sfruttare al meglio le risorse a disposizione.

Proseguiamo analizzando i due grafici relativi al tempo di esecuzione degli ambienti analizzati:

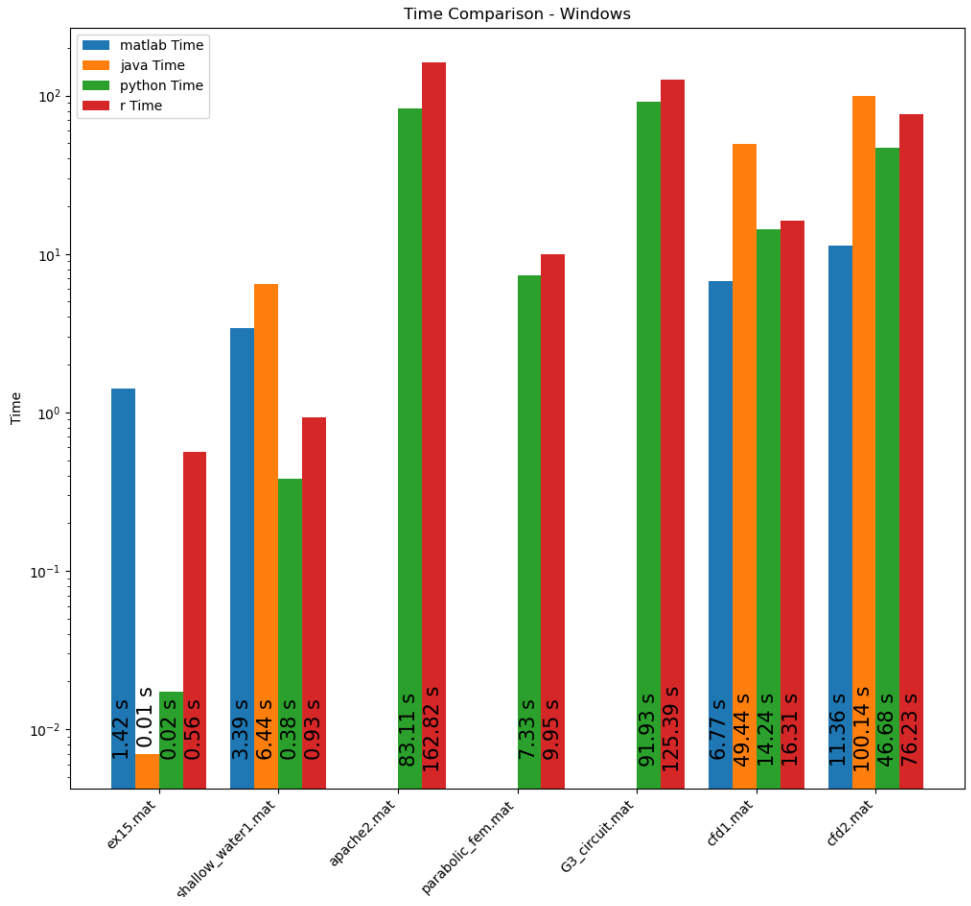

Figura 4.25: Grafico tempo impiegato da Windows

Osservando attentamente i risultati ottenuti possiamo notare come, utilizzando un sistema operativo Windows, il linguaggio MATLAB presenta il minor tempo di elaborazione per le matrici `cfd1.mat` e `cfd2.mat`. Il secondo linguaggio più efficiente risulta essere Python essendo in grado di elaborare il maggior numero di matrici richieste dalla consegna anche se, confrontandolo con i tempi di MATLAB nell'a-

nalizzare le matrici più grandi, risulta comunque esserci un distacco elevato. Per quanto riguarda i tempi misurati con il linguaggio R, questi risultano essere abbastanza equilibrati, permettendo di elaborare le stesse matrici di Python, nonostante i secondi di elaborazione maggiori. in termini di tempo, la libreria EJML risulta essere la peggiore nell'elaborare le matrici.

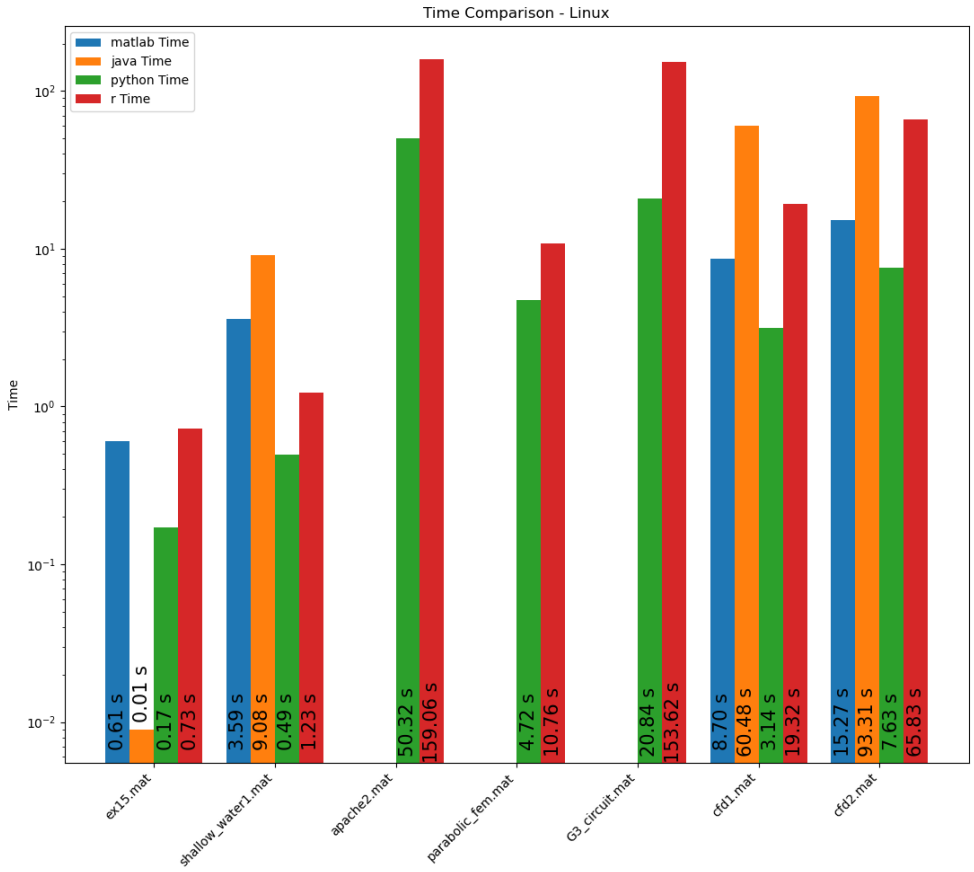

Figura 4.26: Grafico tempo impiegato da Linux

Analizzando i tempi ottenuti con Linux, notiamo che Python risulta essere il linguaggio di programmazione più ottimizzato, in termini di tempo, per la risoluzione di ciascun sistema lineare in tutte le matrici ad eccezione di `ex15.mat`. Pur analizzando un minor numero di matrici, MATLAB conferma la sua efficienza dimostrandosi il secondo linguaggio di programmazione più veloce in questo ambiente. I linguaggi R e Java mantengono lo stesso andamento mostrato nel sistema operativo Windows.

Infine, nelle figure 4.27 e 4.28 è possibile osservare l'errore relativo generato dai quattro linguaggi analizzati:

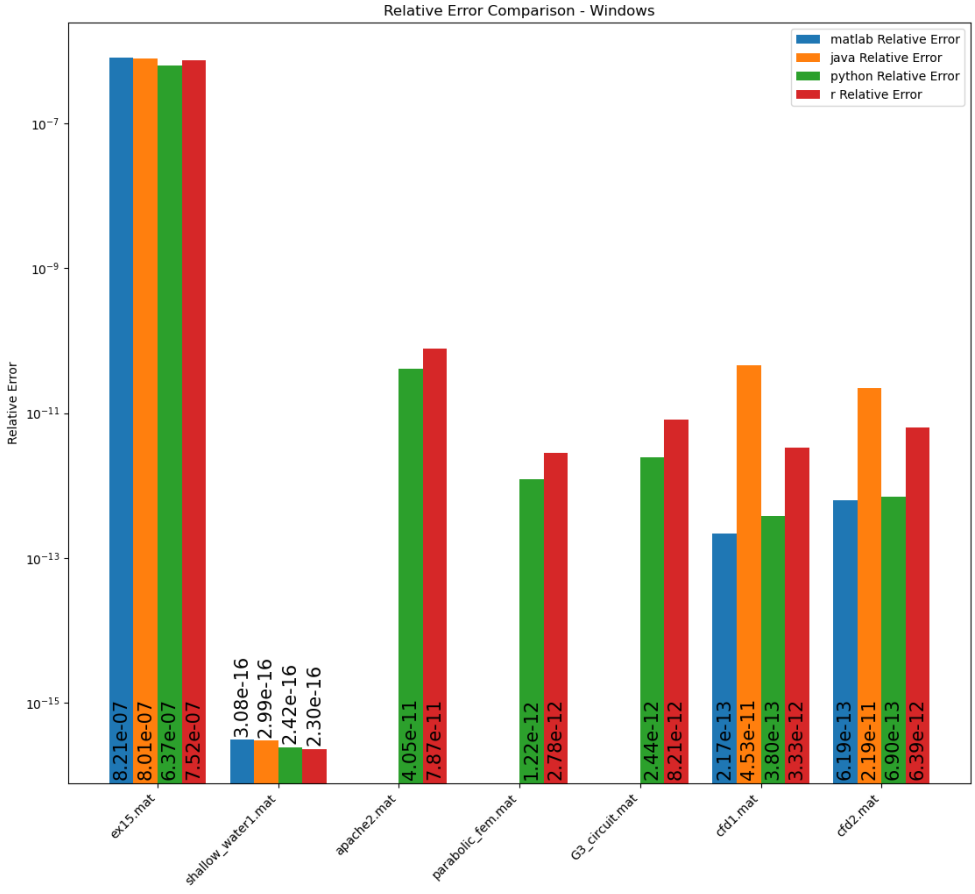

Figura 4.27: Grafico errore relativo ottenuto da Windows

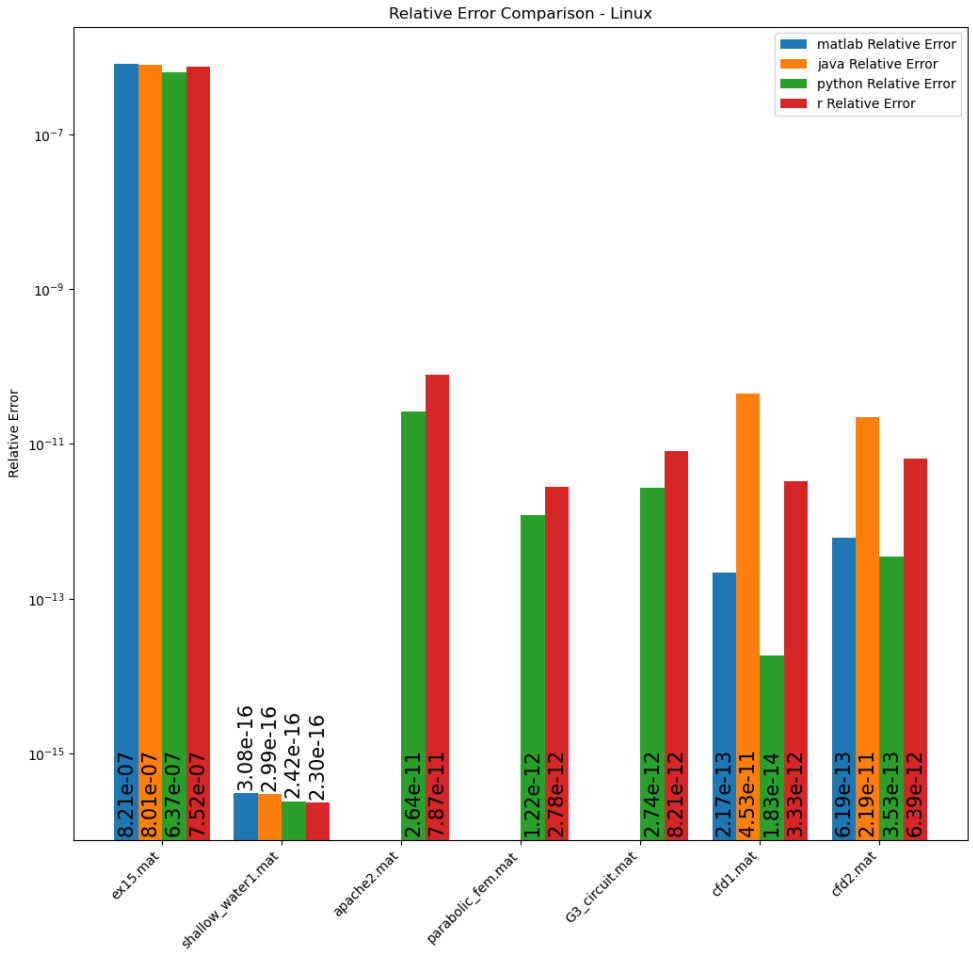

Figura 4.28: Grafico errore relativo ottenuto da Linux

In entrambi i sistemi operativi testati, possiamo vedere come i quattro linguaggi si sono comportati in maniera pressoché identica sulla matrice `ex15.mat`, ottenendo un alto errore relativo, dovuto al già citato mal condizionamento. Mentre, per quanto riguarda le restanti matrici, il linguaggio che si comporta mediamente meglio è Python, avendo ottenuto anche sulle due matrici più grandi un errore relativo significativamente più piccolo rispetto a quello ottenuto dagli altri tre linguaggi.

4.6 Facilità d'uso ed installazione degli ambienti testati

In questa sezione forniremo una panoramica generale toccando i principali aspetti, per ciascun linguaggio di programmazione testato, riguardanti la facilità di installazione e le difficoltà riscontrate nell'utilizzo dei vari ambienti.
Installare la libreria SciKit-sparse è stato difficile su Windows https://github.com/scikit-sparse/scikit-sparse. Su Linux no, perché bastava un semplice comando. R si utilizzano librerie interne, quindi semplice, MATLAB anche li semplice. Java, abbiamo utilizzato Maven, quindi semplice.

4.6.1 Ambiente Matlab

Pur non avendo mai avuto esperienze pregresse con l'utilizzo del linguaggio di sviluppo MATLAB, l'implementazione delle funzioni richieste dalla consegna e l'installazione dell'intero ambiente di lavoro utile per la realizzazione del progetto, non ha fatto emergere particolari problemi. Possiamo racchiudere la nostra esperienza in una serie di punti:

- **Licenza universitaria**: per poter accedere a tutte le funzionalità offerte da MATLAB è necessario possedere una licenza d'uso. In quanto studenti, fortunatamente, essa è offerta gratuitamente. In ambito lavorativo o privato sarà necessario acquistarla.

- **Installazione semplice**: l'installazione dell'ambiente MATLAB risulta intuitiva e guidata, soprattutto per ambienti Windows. Accedendo con il proprio account è stato sufficiente scaricare i file necessari e seguire pochi passaggi per installare ed avviare il software.

- **Ampia scelta di funzioni built-in**: durante l'utilizzo di MATLAB per implementare il progetto abbiamo trovato molto utile l'insieme di funzioni built-in, implementate nativamente dall'ambiente che possono essere utilizzate senza la necessità di importare librerie esterne.

- **Documentazione ben fatta**: consultando il sito ufficiale di MATLAB (https://it.mathworks.com/help/matlab/) è possibile notare come la documentazione sia ben strutturata, ricca di informazioni, spiegazioni ed esempi. Questa

caratteristica si è dimostrata molto utile durante la realizzazione del progetto: nelle fasi iniziali questo ci ha permesso di apprendere le funzioni basi di MATLAB rapidamente.

- **Mantenimento frequente**: MATLAB è soggetto a regolari aggiornamenti, garantendo così la correzione di bug e l'introduzione di nuove funzionalità.

Tuttavia, come già anticipato, è importante sottolineare che l'utilizzo di MATLAB richiede una licenza d'uso del software, e la sua comunità, sebbene attiva, potrebbe essere meno numerosa rispetto a quella di alcuni software `open-source`.

4.6.2 Ambiente R

L'utilizzo di R all'interno del progetto è stato agevole:

- **Open-Source e facile installazione**: R è un software `open-source` e può essere utilizzato senza acquistare nessuna licenza. L'installazione di R e dell'ambiente di sviluppo RStudio risulta è semplice sia per sistemi operativi Windows che Linux.
- **Documentazione**: l'organizzazione della documentazione consultata non risulta essere d'immediata comprensione, nonostante sia completa per raggiungere gli scopi previsti dal progetto.
- **Sintassi intuitiva**: La sintassi di R è anch'essa intuitiva grazie al suo tipaggio debole e alle sue funzioni built-in.
- **Manutenzione regolare**: R è soggetto a manutenzione regolare, garantendo la stabilità e l'aggiornamento delle librerie.

4.6.3 Ambiente Java con libreria EJML

L'uso di Java in combinazione con la libreria EJML ha permesso di trarre le seguenti conclusioni:

- **Installazione di Java**: per sviluppare ed eseguire qualsiasi tipo di applicazione scritta in Java sulla propria macchina è obbligatorio installare correttamente il `Java Development Kit` disponibile al seguente link: https://www.oracle.com/it/java/technologies/downloads/. Esso è un insieme di software `open-source` scaricabile gratuitamente.

- **Importazione della libreria EJML tramite Maven**: L'utilizzo della libreria EJML all'interno del progetto Java è stato agevolato grazie all'utilizzo di Maven. La presenza della libreria EJML all'interno del `Maven Central Repository` permette l'aggiunta della libreria in modo rapido nel proprio progetto, gestendone l'importazione.
- **Usabilità della libreria**: la libreria EJML offre un'ampia gamma di funzionalità per il calcolo scientifico e la manipolazione delle matrici sparse facilmente implementabili per chi ha dimestichezza in ambiente Java .
- **Comprensione della documentazione**: durante la realizzazione del progetto, pur essendo disponibile il sito internet della libreria con il rispettivo manuale e la JavaDoc, inizialmente non è stato molto facile quali metodi della libreria utilizzare per raggiungere lo scopo del progetto. Sebbene tutte le classi e i vari metodi siano spiegati singolarmente, non risulta molto chiaro, per chi si approccia per la prima volta, capire quali parti della libreria siano necessarie per raggiungere uno specifico obbiettivo.

4.6.4 Ambiente Python con SciPy e SciKit-Sparse

L'utilizzo di Python con le librerie SciPy e SciKit-Sparse ha presentato alcune sfide relative all'installazione:

- **Installazione complessa su Windows**: L'installazione di Anaconda, necessaria per utilizzare `SciKit-Sparse`, può risultare complessa su sistemi Windows, in quanto è richiesta una configurazione di un ambiente Python specifico. Sebbene sia necessaria una quantità aggiuntiva di tempo per il corretto funzionamento dell'ambiente, rispetto agli altri linguaggi testati, seguendo le istruzioni riportate al seguente link: https://github.com/scikit-sparse/scikit-sparse permettono di eseguire un'installazione completa di tutto il necessario.
- **Risorse ricche per il calcolo scientifico**: Python, una volta configurato, offre risorse estremamente ricche per il calcolo scientifico, tra cui numerose librerie specializzate: `Scipy`, `Numpy` e la sopra citata `SciKit-Sparse` permettono di eseguire calcoli complessi anche con matrici di tipo sparso il tutto supportato dall'utilizzo di una sintassi agevole per il programmatore.

In conclusione, mentre Matlab, R e Java (con EJML) hanno dimostrato una facilità d'uso significativa e un'installazione relativamente semplice, l'utilizzo di Python con

SciKit-Sparse potrebbe comportare una maggiore complessità in fase di configurazione, soprattutto su sistemi Windows. Tuttavia, la potenza e la flessibilità delle librerie Python per il calcolo scientifico possono risultare un vantaggio significativo per il raggiungimento degli obiettivi fissati dal progetto, permettendo di porre in secondo piano le difficoltà riscontrate durante l'installazione.

5 Conclusioni

In questo progetto, abbiamo esaminato le prestazioni offerte da quattro linguaggi di programmazione, MATLAB, Java, Python e R, nell'ambito della risoluzione di sistemi lineari mediante la decomposizione di Cholesky, utilizzando due diversi sistemi operativi, Windows e Linux, come ambiente di esecuzione. Questa analisi ha fornito importanti indicazioni su quale linguaggio e sistema operativo possano essere più adatti per affrontare gli obiettivi prefissati dal progetto, tenendo conto delle prestazioni e dell'efficienza nell'utilizzo delle risorse.

5.1 Prestazioni nei diversi linguaggi

Riepiloghiamo quanto ottenuto in termini di metriche prestazionali dai linguaggi di programmazioni analizzati all'interno del progetto.

Utilizzo di memoria

Iniziamo esaminando l'utilizzo di memoria. Abbiamo scoperto che, in generale, Python e R richiedono meno memoria rispetto a MATLAB e Java. Questa differenza diventa particolarmente evidente quando si lavora con matrici di grandi dimensioni. Python, in particolare, si è dimostrato il più efficiente in termini di `MegaByte` utilizzati, soprattutto in ambiente Linux.

Tempo di esecuzione

Passando al tempo di esecuzione, abbiamo osservato che MATLAB, in ambiente Windows, risulta essere il più veloce per matrici di grandi dimensioni. Tuttavia, utilizzando l'ambiente Linux, Python si dimostra essere il linguaggio di programmazione più ottimizzato in termini di secondi necessari a risolvere il sistema. R, sebbene leggermente più lento di Python, è comunque competitivo permettendo di elaborare le stesse matrici del linguaggio Python. Java, pur mostrando prestazioni accettabili, è quello che impiega più tempo per elaborare le matrici.

Errore relativo

L'analisi dell'errore relativo ha mostrato che tutti i linguaggi hanno ottenuto risultati simili sulla matrice altamente mal condizionata `ex15.mat`. Tuttavia, per le altre matrici, Python è emerso come il linguaggio che ha prodotto l'errore relativo più basso, dimostrando una buona stabilità numerica.

5.2 Scelta del linguaggio e del sistema operativo

La scelta tra questi linguaggi e sistemi operativi dipende dai requisiti specifici che si desiderano rispettare e dalle risorse disponibili. Ecco alcune considerazioni chiave:

- **MATLAB vs. linguaggi Open-Source**: nel caso in cui il progetto richiede prestazioni eccezionali su sistemi lineari di grandi dimensioni con la possibilità di investire nelle licenze, MATLAB può essere una scelta valida. Tuttavia, se si è alla ricerca di una soluzione `open-source` altamente efficiente, Python si dimostra una scelta eccellente in termini di prestazioni e utilizzo di memoria.
- **Python vs. R**: Python sembra essere il linguaggio preferito, avendo dimostrato che entrambi i linguaggi riescano a risolvere il maggior numero di matrici, e nel caso in cui la stabilità numerica sia un requisito critico del proprio progetto, con corrispondente errore relativo minimo. Al contrario, R si dimostra una buona alternativa, soprattutto se il contesto richiede analisi statistiche più avanzate.
- **Sistema operativo**: La scelta tra Windows e Linux ricade in gran parte sulla familiarità e sulle esigenze specifiche. Se è necessario ottenere prestazioni elevate e un controllo più fine sul sistema, Linux, soprattutto in combinazione con Python, risulta essere la scelta migliore. É necessario inoltre considerare

che per l'utilizzo di Windows è necessaria un'apposita licenza, a differenza di Linux che è `open-source`.

In conclusione, non esiste una risposta universale alla domanda se sia meglio affidarsi a MATLAB mediante licenza o se sia valido avventurarsi nel mondo open source: dipende dai requisiti del progetto e dalle risorse disponibili. Tuttavia, basandoci sulle prestazioni osservate in questo studio, l'uso di linguaggi `open-source` come Python e R in combinazione con Linux, sembra essere una scelta altamente competitiva in termini di efficienza e flessibilità per una vasta gamma di applicazioni scientifiche e di calcolo numerico.