CHOLESKY ANALYSIS

METODI DEL CALCOLO SCIENTIFICO

Mario Avolio 880995

Simone Benitozzi 889407

13 giugno 2023

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

L'obiettivo dell'elaborato è principalmente quello di confrontare l'ambiente open-source con quello proprietary-software per la risoluzione di un problema comune, in maniera tale da trarne i giusti spunti e considerazioni.

Nello specifico verrà analizzata l'implementazione e il funzionamento in ambienti di programmazione e Sistemi Operativi differenti del metodo di Cholesky, per la risoluzione sistemi lineari per matrici sparse, simmetriche e definite positive.

Sono stati confrontati MATLAB, linguaggio chiuso e utilizzabile sotto licenza, e Python, open source e ricco di librerie per la risoluzione di una gran vastità di problemi. Per quanto riguarda invece i Sistemi Operativi, il confronto vedrà la contrapposizione tra esecuzioni in Windows e Linux.

Lo scopo è quello di presentare un'analisi quanto più approfondita possibile, che tenga conto dei possibili trade-off e non si limiti solamente all'aspetto tecnico-informatico, ragionando quindi in termini di performance ed efficacia, ma che abbracci anche l'ambito manageriale, considerando i costi e le ripercussioni di una scelta rispetto all'altra, simulando un contesto reale di decision making e facendo sì che lo studio possa portare ad una scelta ponderata.

Reperibilità del progetto su GitHub Nella trattazione che segue verranno descritte le scelte di progettazione effettuate e i corrispettivi codici Matlab e Python. Invitiamo il lettore a dare uno sguardo alla Repository GitHub qualora si voglia analizzare più dettagliatamente il progetto.

2 Descrizione del dominio di riferimento e obiettivi della sperimentazione

2.1 Risoluzione di Sistemi Lineari

Il problema della risoluzione di sistemi lineari attraverso una computazione efficiente è tuttora aperto e al centro di progetti e obiettivi di ricerca. Allo stato dell'arte attuale si definiscono due tipologie principali di metodologie per la loro risoluzione: i metodi diretti e i metodi iterativi.

Metodi Diretti I metodi diretti permettono di arrivare ad una soluzione in un numero finito di step e restituirebbero la soluzione esatta sotto l'assunzione di essere eseguiti su un'aritmetica con precisione infinita, cosa in realtà non possibile sui reali calcolatori. La tecnica di base per una risoluzione di questo tipo è detta *fattorizzazione LU* di una matrice A, che viene scomposta in una *Lower triangular L* e una *Upper triangular U*, con il risultato finale calcolato come segue:

$$\begin{aligned} Ax &= b \\ L(Ux) &= b; Ux = y \\ Ly &= b \end{aligned} \tag{1}$$

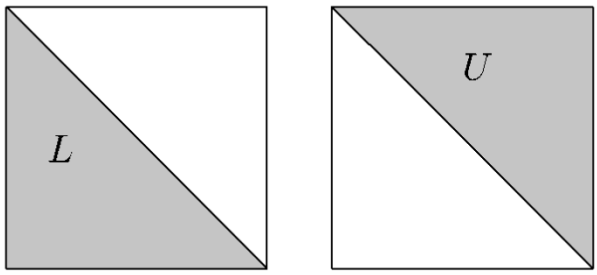


Figura 1: LU Decomposition for Direct Methods

Metodi Iterativi I metodi iterativi non assicurano di terminare in un numero finito di step: a partire da un *initial guess* formano iterazioni successive che convergono solo al limite della soluzione esatta. Per far sì che l'iterazioni termini, viene utilizzato un criterio di convergenza per specificare quando una soluzione approssima sufficientemente al risultato esatto.

Anche in caso dell'utilizzo di un'aritmetica con precisione infinita, i metodi iterativi non restituirebbero comunque, in linea teorica, una soluzione esatta, ma il loro vantaggio è quello di essere computazionalmente più efficienti dei diretti, e pertanto più utilizzati nella pratica, sebbene abbiano la necessità di essere eseguiti su matrici che rispettino precise proprietà per essere certi della convergenza. Ad esempio, 2 tra i metodi iterativi più comuni, *Jacobi* e *Gauß-Seidel*, garantiscono la convergenza solo in caso di matrici a dominanza diagonale in input.

2.1.1 Cholesky

Nel presente elaborato viene analizzata la risoluzione di sistemi lineari mediante decomposizione di Cholesky, una particolare tipologia di metodo diretto che sfrutta la proprietà secondo cui, se la matrice *A* è *SDP* (simmetrica e definita positiva), è allora possibile riorganizzare la decomposizione in maniera tale che *U* sia la trasposta di *L*, pertanto *A* possa essere riscritta come

$$Ax = LL' \tag{2}$$

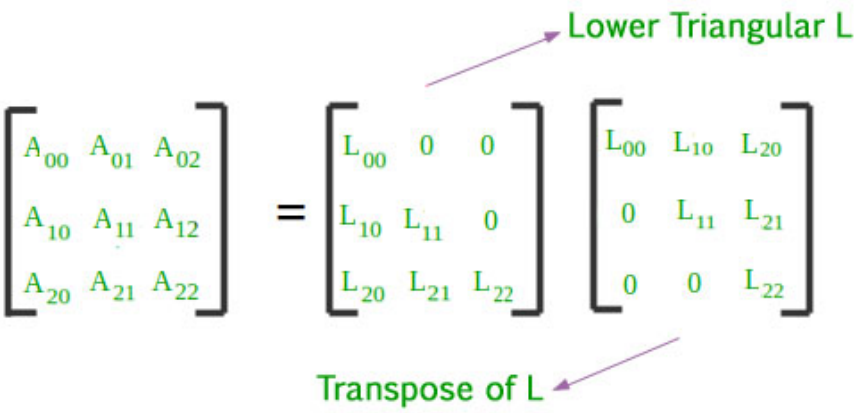


Figura 2: Basic visualization for Cholesky Decomposition

Se la matrice rispetta tale proprietà, allora è garantito che la sua decomposizione di Cholesky esista e sia unica, il che rende la risoluzione del sistema più efficiente e stabile rispetto ad una normale decomposizione *LU*.

Come già visto precedentemente, sebbene si tratti di un metodo di risoluzione diretto, che pertanto dovrebbe, in linea teorica, restituire un risultato esatto a seguito di un numero finito di step, ciò non è garantito dal fatto di utilizzare dei calcolatori non provvisti di precisione aritmetica infinita. È per questo che tra le analisi presentate in seguito, si terrà conto nella valutazione del miglior risultato, anche dell'errore relativo tra il risultato calcolato e quello atteso, auspicando comunque che esso si avvicini quanto più possibile all'epsilon di macchina, pari a

$$2.220446049250313 * 10^{-16}$$

2.2 Matrici Sparse

Avendo a che fare con dati di dimensioni molto grandi, nelle analisi che seguono sono state utilizzate le matrici sparse: si tratta di matrici con la maggior parte dei valori pari a 0, che vengono memorizzate in maniera compatta, tenendo conto dei soli valori positivi.

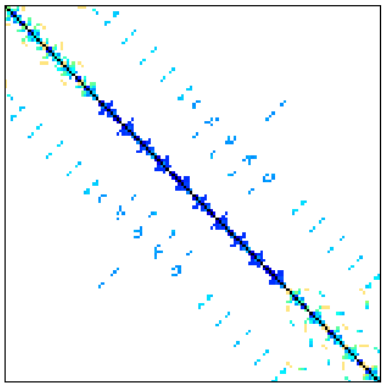


Figura 3: Sparse matrix example

Questo modo di operare permette, oltre al fatto di risparmiare spazio in memoria a seguito del cricamento della matrice stessa, anche una risoluzione più efficiente della decomposizione di Cholesky stessa.

La semplice decomposizione di Cholesky, applicata ad una matrice sparsa, ne causerebbe il *fill-in* [1], a seguito della generazione di elementi diversi da 0. Ciò non accade in una particolare tipologia di matrici sparse, le matrici *tridiagonali*, nelle quali gli elementi diversi da zero sono solo sulla diagonale principale e sulle due sottodiagonali.

Per trattare matrici sparse generali viene quindi effettuata una permutazione preliminare di righe e colonne in modo che l'algoritmo di Cholesky generi il minor numero possibile di elementi diversi da zero, per una risoluzione più efficiente e performante.

[1] Il fill-in di una matrice è costituito da quei valori che da zero diventano diversi da zero durante l'esecuzione di un algoritmo.

3 Descrizione dei Dati

Di seguito verranno trattate le diverse matrici utilizzate durante la sperimentazione. In particolar modo si vuole sottolineare che esse fanno parte della SuiteSparse Matrix Collection che colleziona matrici sparse derivanti da applicazioni di problemi reali come: ingegneria strutturale, fluidodinamica, elettromagnetismo, termodinamica, computer graphics/vision, network e grafi. [2]

3.1 Matrici

Tabella 1: Flan-1565 Information

Name	Flan_1565
Group	Janna
Matrix ID	2544
Num Rows	1564794
Num Cols	1564794
Nonzeros	114165372
Pattern Entries	117406044
Kind	Structural Problem
Symmetric	Yes
Date	2011
Author	C. Janna, M. Ferronato
Editor	T. Davis
Structural Rank	1564794
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	3240672
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

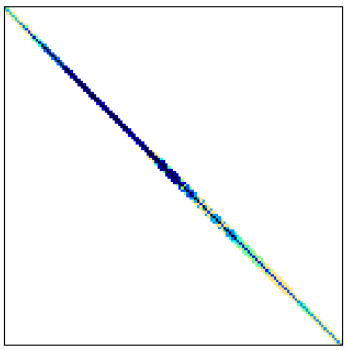


Figura 4: Flan-1565

[2]Nei dati riportati ricorrerà spesso la sigla CFDP, acronimo di "Computational Fluid Dynamics Problem".

Tabella 2: StocF-1465 Information

Name	StocF-1465
Group	Janna
Matrix ID	2547
Num Rows	1465137
Num Cols	1465137
Nonzeros	21005389
Pattern Entries	21005389
Kind	CFDP
Symmetric	Yes
Date	2011
Author	C. Janna, M. Ferronato
Editor	T. Davis
Structural Rank	1465137
Structural Rank Full	true
Num Dmperm Blocks	29105
Strongly Connect Components	29105
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

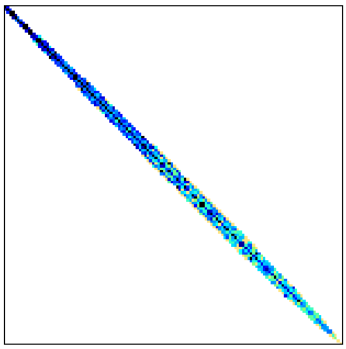


Figura 5: StocF-1465

Tabella 3: cfd2 Information

Name	cfd2
Group	Rothberg
Matrix ID	805
Num Rows	123440
Num Cols	123440
Nonzeros	3085406
Pattern Entries	3087898
Kind	CFDP
Symmetric	Yes
Date	1997
Author	E. Rothberg
Editor	T. Davis
Structural Rank	123440
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	2492
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

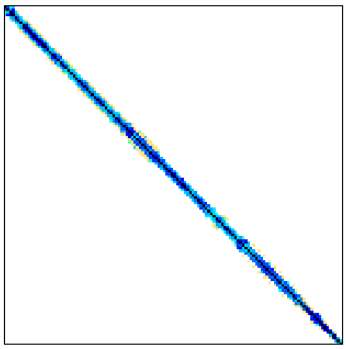


Figura 6: cfd2

Tabella 4: cdf1 Information

Name	cfd1
Group	Rothberg
Matrix ID	804
Num Rows	70656
Num Cols	70656
Nonzeros	1825580
Pattern Entries	1828364
Kind	CFDP
Symmetric	Yes
Date	1997
Author	E. Rothberg
Editor	T. Davis
Structural Rank	70656
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	2784
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

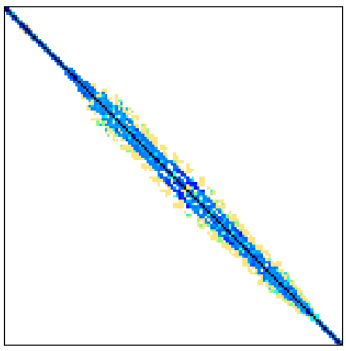


Figura 7: cdf1

Tabella 5: G3-circuit Information

Name	G3_circuit
Group	AMD
Matrix ID	1421
Num Rows	1585478
Num Cols	1585478
Nonzeros	7660826
Pattern Entries	7660826
Kind	Circuit Simulation Prob
Symmetric	Yes
Date	2006
Author	U. Okuyucu
Editor	T. Davis
Structural Rank	1585478
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

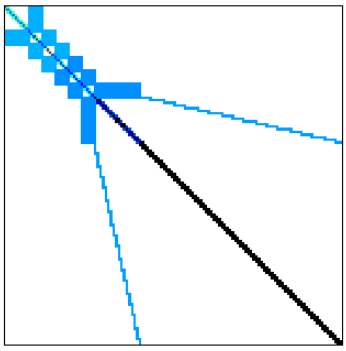


Figura 8: G3-circuit

Tabella 6: Parabolic Fem Information

Name	parabolic_fem
Group	Wissgott
Matrix ID	1853
Num Rows	525825
Num Cols	525825
Nonzeros	3674625
Pattern Entries	3674625
Kind	CFDP
Symmetric	Yes
Date	2007
Author	P. Wissgott
Editor	T. Davis
Structural Rank	525825
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

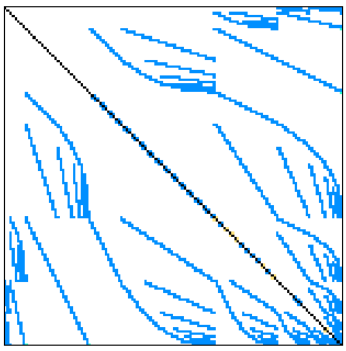


Figura 9: Parabolic Fem

Tabella 7: apache2 Information

Name	apache2
Group	GHS_psdef
Matrix ID	1423
Num Rows	715176
Num Cols	715176
Nonzeros	4817870
Pattern Entries	4817870
Kind	Structural Problem
Symmetric	Yes
Date	2006
Author	NaN
Editor	N. Gould, Y. Hu, J. Scot
Structural Rank	715176
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

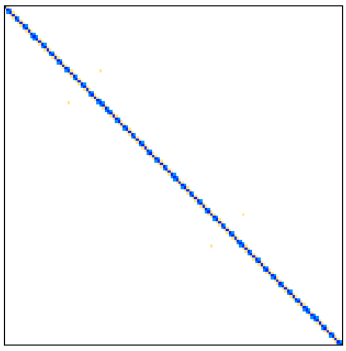


Figura 10: apache2

Tabella 8: shallow water1 Information

Name	shallow_water1
Group	MaxPlanck
Matrix ID	2261
Num Rows	81920
Num Cols	81920
Nonzeros	327680
Pattern Entries	327680
Kind	CFDP
Symmetric	Yes
Date	2009
Author	K. Leppkes, U. Nauman
Editor	T. Davis
Structural Rank	81920
Structural Rank Full	true
Num Dmperm Blocks	1
Strongly Connect Components	1
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

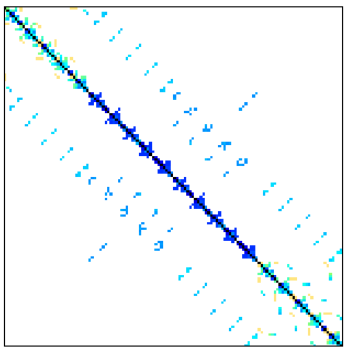


Figura 11: shallow water1

Tabella 9: ex15 Information

Name	ex15
Group	FIDAP
Matrix ID	413
Num Rows	6867
Num Cols	6867
Nonzeros	98671
Pattern Entries	98671
Kind	CFDP
Symmetric	Yes
Date	1994
Author	A. Baggag, Y. Saad
Editor	A. Baggag, Y. Saad
Structural Rank	6867
Structural Rank Full	true
Num Dmperm Blocks	2
Strongly Connect Components	2
Num Explicit Zeros	0
Pattern Symmetry	100%
Numeric Symmetry	100%
Cholesky Candidate	yes
Positive Definite	yes
Type	real

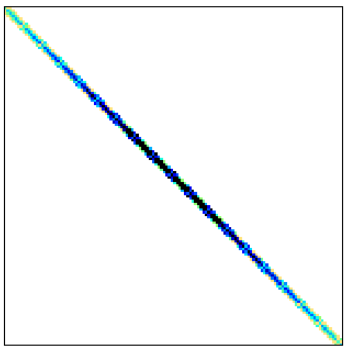


Figura 12: ex15

4 Scelte di Progettazione

Nella seguente sezione verranno trattati le principali scelte di progettazione atte all'analisi dei dati delle matrici e ai corrispettivi dati riscontrati dall'analisi. La figura 13 mostra i passi fondamentali eseguiti durante tutta la durata della sperimentazione:

1. Esecuzione del metodo di Cholesky sulle singole Matrici al variare dell'OS e del linguaggio.
2. Salvataggio dei dati riscontrati.
3. Analisi dei dati.

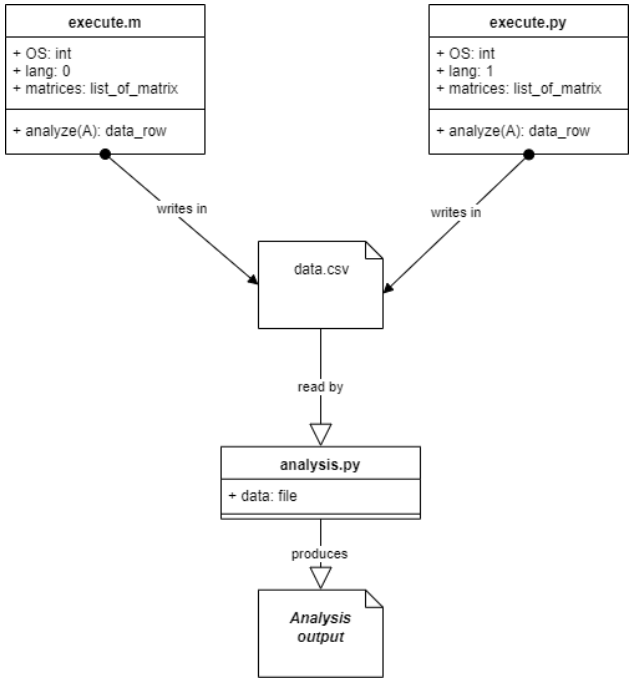


Figura 13: Diagramma UML di progettazione

La seguente trattazione non ha il compito di definire in modo particolarmente tecnico tutta la fase di progettazione, bensì si vuole soffermare su quegli aspetti ritenuti fondamentali per comprendere al meglio l'operato. Dalla figura 13 si possono notare anche molteplici altri aspetti:

1. **Reperimento Dati:** sono stati utilizzati due programmi che contengono alcuni particolare script (*execute.m* e *execute.py*) per eseguire il metodo di Cholesky sulle varie matrici e per il reperimento dei corrispettivi dati. E' doveroso sottolineare che essi non hanno il compito di analizzare i dati da essi stessi prodotti. L'utilizzo di questa metodologia ha permesso una migliore divisione tra le componenti adibite al reperimento dei dati e quelle per la corrispettiva analisi. Un approccio diverso sarebbe stato estramamente lento e poco preciso per una corretta e omogenea analisi.
2. **Salvataggio Dati:** il salvataggio dei dati, riscontrati al punti precedente, avviene mediante un file *data.csv* che verrà analizzato come se fosse un **dataset**.
3. **Analisi dei Dati:** l'analisi dei dati è stata effettuata a posteriori rispetto al reperimento dati, in particolar modo dopo aver eseguito i due script sui sistemi operativi di interesse. Questa procedura ha permesso di analizzare i dati in maniera indipendente rispetto agli script atti al reperimento dei dati.

Nelle prossime sezioni verranno trattati più in dettaglio i procedimenti sfruttati e le librerie utilizzate per lo scopo. Oltre a ciò verrà anche definita l'architettura sfruttata per l'esecuzione dei diversi Sistemi Operativi.

4.1 L'utilizzo della Virtualizzazione

Al fine di garantire una più pulita analisi dei dati si è optato per l'utilizzo della virtualizzazione dei sistemi Linux e Microsoft Windows, su cui si sono svolti i dovuti test. In particolare la scelta in ambiente linux si è rivolta sulla distribuzione **Linux Lubuntu**, mentre in ambiente Microfost Windows la scelta è ricaduta su **Windows 10**.

HyperVisor Per favorire la miglior virtualizzazione possibile è stato necessario l'impiego dell'hypervisor (Wikipedia (2021)) che è il componente centrale e più importante di un sistema basato sulle macchine virtuali. Un computer sul quale venga eseguito un hypervisor che a sua volta controlla una o più macchine virtuali è detto macchina host, e ogni macchina virtuale è detta macchina guest. Il compito di un hypervisor è quello di presentare all'utente i sistemi operativi delle macchine guest e di gestire la loro esecuzione. Grazie ad un hypervisor, su una macchina host possono essere in esecuzione contemporaneamente diverse macchine guest, su ognuna delle quali può girare un sistema operativo diverso che ha il controllo sulle risorse hardware virtualizzate rese disponibili dall'hypervisor. La ragione per cui è stato scelto è inerente al fatto che questo tipo di virtualizzazione è diversa dalla virtualizzazione a livello di sistema operativo, dove tutte le istanze (dette anche container) devono essere eseguite in un unico kernel. In particolar modo è opportuno sottolineare che per tale scopo è stato sfruttato **Microsoft Hyper-V**, ovvero un **HyperVisor di tipo 1** (*native hypervisor*). Questa tipologia di virtualizzazione offre numerosi benefici rispetto agli **HyperVisor di tipo 2** (*hosted hypervisor*). Sebbene in questa esposizione non si voglia entrare nel merito della questione, nella figura 14 sono schematizzate le principali differenze tra i due sistemi di virtualizzazione. E' doveroso notare che la virtualizzazione è stata eseguita sfruttando lo stesso *hardware* per entrambi i sistemi operativi testati, in particolare ad ognuno di essi sono stati dedicati 6 core sulla CPU (*Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz 1.61 GHz*) e 8GB di RAM.

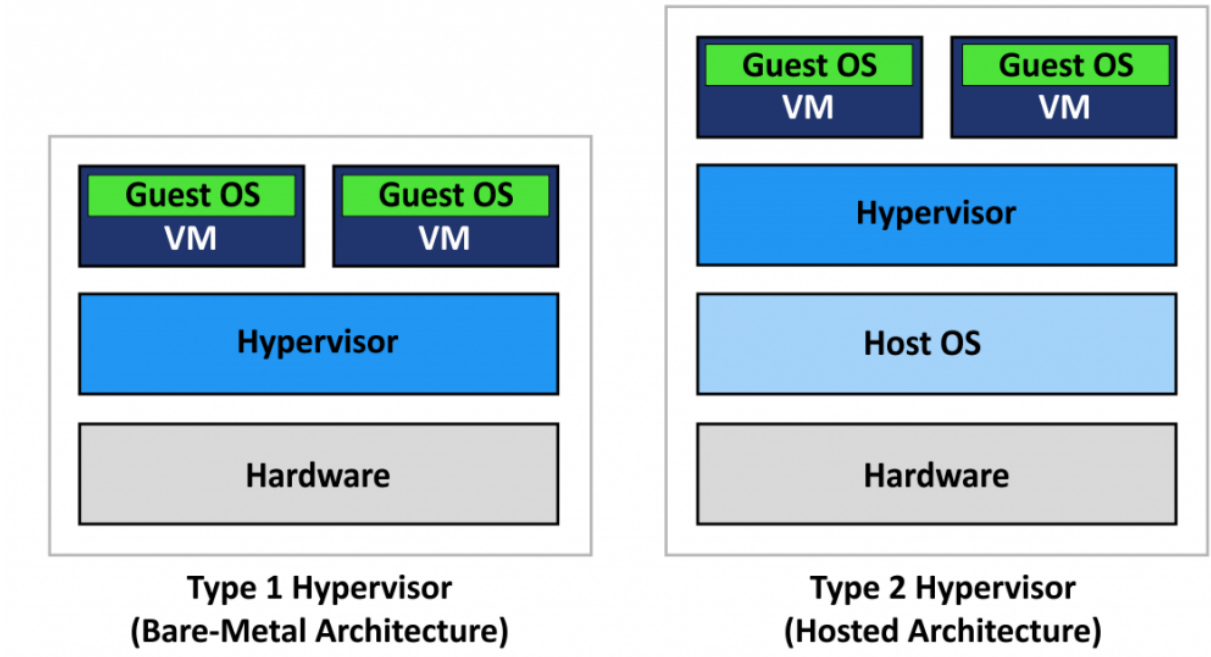


Figura 14: HyperVisor Types from www.nakivo.com

4.2 Reperimento Dati

Di seguito verranno introdotte più in dettaglio le strutture di progettazione.

4.2.1 MATLAB

Lo script execute.m Lo script *execute.m* ha il compito di prendere in considerazione ogni matrice per poi analizzarla singolarmente mediante lo script *analyze.m*.

Lo stesso viene fatto da *execute_backslash.m*, con la differenza che viene chiamata la funzione *analyze_backslash.m*, nella quale il sistema lineare viene risolto in maniera alternativa.

La figura 15 ne mostra il funzionamento.

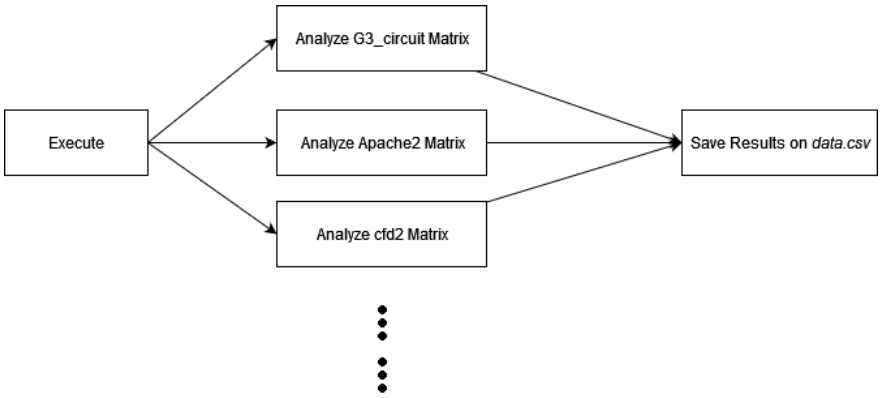


Figura 15: Funzionamento dello script *execute*

In MATLAB la risoluzione del sistema lineare è stata eseguita attraverso 2 diverse metodologie, di cui sono stati analizzati i risultati e confrontati tra loro

La risoluzione con chol() Il primo metodo è la funzione *chol()*, specifico per la decomposizione di Cholesky, che fattorizza la matrice in input *A*, restituendo una triangolare superiore *R* tale che

$$A = R' * R \tag{3}$$

Lavorando su matrici sparse è stato inoltre possibile utilizzare la variante ottimizzata dell'algoritmo, che restituisce un'addizionale matrice di permutazione *P* tale che

$$R' * R = P' * S * P \tag{4}$$

Di conseguenza il risultato finale *x* è stato calcolato, tenendo conto di tutti i dati a disposizione, attraverso la seguente formula:

$$x = P * (R \backslash (R' \backslash (P' * b)) \tag{5}$$

Risoluzione con backslah Il secondo procedimento è stato implementato mediante l'utilizzo del metodo built-in in MATLAB per la risoluzione dei sistemi lineari: il "backslash" (\).
Tale funzione, se verifica che la matrice da scomporre è *SDP*, utilizza anch'essa la decomposizione di Cholesky, attraverso un flusso di operazioni "nascosto", e probabilmente maggiormente ottimizzato rispetto alla soluzione precedente.
Ne risulta che il sistema viene semplicemente risolto come segue

$$x = A \backslash b \tag{6}$$

I risultati dei due metodi verranno confrontati tra loro, sebbene ci si aspetta che sia il *backslash* ad avere la meglio su gran parte delle metriche analizzate.
In quanto al confronto con la libreria open source, verrà tenuto conto dei risultati restituiti dal metodo *chol()*, il cui funzionamento è più simile al metodo utilizzato dalla controparte, e pertanto permette un'analisi più equa

Lo script matrix_properties Nelle analisi che seguono vengono tenuti in considerazione, oltre alle classiche metriche, quali errore, tempo e memoria, anche proprietà strutturali delle matrice, che contribuiscono ad un'interpretazione più accurata dei risultati.
Nello specifico questo script calcola numero di righe, numero di valori diversi da 0, da cui estrarre densità e sparsità, e il coefficiente di condizionamento della matrice, necessario per interpretare gli errori relativi nel calcolo del sistema lineare.
Il calcolo del condizionamento è stato fatto mediante la funzione *condest()* di MATLAB, che fa ausilio della norma 1, in quanto ottimizzata per l'esecuzione su matrici sparse. La funzione più classica *cond()*, che utilizza invece norma 2, richiedeva la conversione in matrice densa dell'input, in quanto non supporta matrici sparse.
La raccolta di queste metriche è stata effettuata in uno script separato vista la quantità di lavoro richiesta per calcolare il condizionamento delle matrici più grandi, che andrebbe a sommarsi al carico di lavoro già non indifferente per la risoluzione del sistema.

```
function [error, mem, time] = analyze(A)

% Analyzes the Cholesky decomposition on the sparse Matrix given
% in input
%
% Inputs:
%   A: the Matrix to be analyzed
%
% Outputs:
%   error: the relative error between the expected result and the computed
%       result
%   mem: the difference in memory (expressed in MB) used by MATLAB between
%       right after the matrix is loaded and after the linear system
%       solution is computed
%   time: the number of seconds required to compute the solution

    try
        % checks if the input matrix A is sparse
        if(~issparse(A))
            err = MException('analyze:NoSparse', ...
                    'Invalid Input. The matrix given in input is not Sparse');
                throw(err);
        end
    catch exception
        fprintf("Error: %s\n", exception.identifier)
        error = NaN; mem = NaN; time = NaN;
        return
    end

    % memory usage before execution (after loading matrix)
    try
        before_mem = memory;
    catch
        [~, pid] = system('pgrep MATLAB');
        [~, mem_usage] = system(['cat /proc/' strtrim(pid) '/status | grep VmSize'
    ]);
        before_mem = str2double(strtrim(extractAfter(extractBefore(mem_usage, ' kB
    '), ':'))) / 1000;
    end

```

```
39      % problem parameters
40      xe = ones(size(A, 1), 1);
41      b = A*xe;
42
43      % --- system solving
44      try
45          tic %starts timing
46
47          % Cholesky decomposition (the input matrix A needs to be sparse to apply
        the amd Permutation)
48          [R, flag, P] = chol(A);
49
50          % flag checks if A is a Definite Positive Matrix
51          if(flag ~= 0)
52                  err = MException('analyze:NoSPD', ...
53                      'Invalid Input. The matrix given in input is not a Positive
        Definite');
54                  throw(err);
55          end
56
57          % --- solution computing
58
59          % x = R\(R'\b); % Solution without using the permutation matrix (
        inefficient)
60          x = P*(R\(R'\(P'*b)));
61
62          time = toc; % ends timing
63      catch exception
64          fprintf("Error: %s\n", exception.identifier)
65          error = NaN; mem = NaN; time = NaN;
66          return
67      end
68
69      % --- memory usage estimation
70      try
71          after_mem = memory;
72          % difference and MB conversion
73          mem = (after_mem.MemUsedMATLAB - before_mem.MemUsedMATLAB) * 1e-6;
74      catch
75          [~, pid] = system('pgrep MATLAB');
76          [~, mem_usage] = system(['cat /proc/' strtrim(pid) '/status | grep VmSize'
        ]);
77          after_mem = str2double(strtrim(extractAfter(extractBefore(mem_usage, ' kB'
        ), ':'))) / 1000;
78
79          mem = after_mem - before_mem;
80      end
81
82      % --- error estimation
83      error = norm(x - xe, 2) / norm(xe, 2);
84
85  end
```

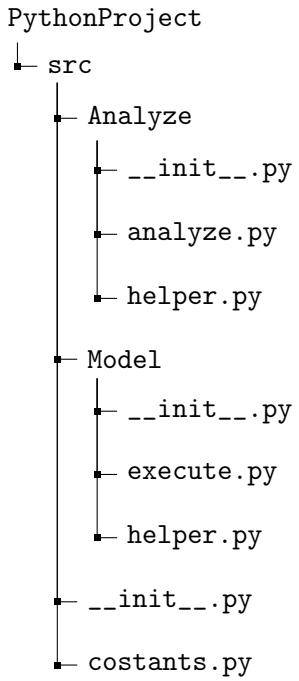
Listing 1: Analyze Class in MATLAB

Problemi Riscontrati e limitazioni L'unica limitazione riscontrata in MATLAB è stata relativa al calcolo della memoria utilizzata dall'operazione di decomposizione e risoluzione del sistema.
Per Windows è stata infatti utilizzata la classica funzione *memory()*, che restituisce la quantità di memoria utilizzata effettivamente dal solo MATLAB. Tale funzione non è eseguibile in ambiente Linux, in cui è stato inveece necessario estrarre la quantità di memoria utilizzata da MATLAB analizzando direttamente il processo in esecuzione in memoria.

I risultati restituiti nei due casi si sono rilevati comunque essere coerenti tra loro, il che lascia pensare che sebbene ci siano differenze nell'estrazione dei dati, l'esito è molto accurato in entrambi i casi.

4.2.2 Python

Per gestire al meglio la giusta separazione tra gli elementi del progetto si è deciso di sfruttare un particolare pattern strutturale definito dallo schema sottostante. Il modello, cattura il comportamento dell'applicazione in termini di dominio del problema e gestisce direttamente i dati, la logica e le regole del progetto.
Nella seguente trattazione, al fine di evitare panoramiche fuori focus, si vogliono sottolineare solo gli aspetti fondamentali utilizzati per il reperimento dei dati tramite l'esecuzione del metodo di Choleksy.



Lo script execute.py Lo script *execute.py* svolge il ruolo primario dell'intero ecosistema python. Esso ha il compito di prendere in considerazione ogni matrice per poi analizzarla singolarmente mediante lo script *analyze.py*.

Lo script analyze.py Il package **Analyze** si occupa principalmente di andare ad applicare ad una spefica matrice il metodo di **Cholesky**. Il codice 2 riporta l'implementazione della classe fondamentale situata nel file **analyze.py**.

```
1  class Analyze:
2      def __init__(self):
3          self.__name = None
4          self.__error = None
5          self.__memoryUsed = None
6          self.__timeTotal = None
7          writeCSV('Name', 'Error', 'Memory', 'Time',
8                   language='Language', operatingSystem='OS')  # write header
9
10     def __analyze(self, path):
11
12         matrix = readMatrix(path)  # read matrix
13         b = getB(matrix)  # get b = A*xe
14
```

```
15          # start time and memory track
16          start_time = time.time()
17          startTrackMemory()
18          ############################
19
20          # CHOLESKY
21          x = scikit_sparse_cholesky(matrix, b)
22          ############################
23
24          # stop time and memory track
25          self.__memoryUsed = convert_size(endTrackMemory())
26          self.__timeTotal = time.time() - start_time
27          ############################
28
29          # compute distance
30          self.__error = relativeError(x)
31
32      def __setName(self, name):
33          self.__name = name.split(".mtx")[0]
34
35      def startAnalyze(self, path, name):
36          self.__setName(name)
37
38          # Analysis
39          self.__analyze(path)
40
41          # write data
42          writeCSV(self.__name, self.__error,
43                   self.__memoryUsed, self.__timeTotal, language=1, operatingSystem=
    getOperatingSystem())
44
45          print(f"Name: {self.__name}")
46          print(f"Memory: {self.__memoryUsed}")
47          print(f"Seconds: {self.__timeTotal} \n")
48          print(f"Error: {self.__error}")
```

Listing 2: Analyze Class in Python

Come si può notare essa racchiude le parti fondamentali dell'operazione effettuata, in particolar modo si vuole sottolineare l'importanza della funzione **__analyze(self, path)**. Essa prevede in input il *path* dov'è situata la matrice da leggere. La funzione esegue i seguenti step:

1. Lettura della Matrice
2. Inizio dell'analisi sulla memoria occupata e sul tempo impiegato.
3. Esecuzione del metodo di Cholesky
4. Fine dell'analisi sulla memoria occupata e sul tempo impiegato.
5. Calcolo dell'errore relativo rispetto alla soluzione fornita di default.

Come si può facilmente notare dal codice 2, si è predisposto il tutto per scrivere i risultati ottenuti in tabelle **.csv*, al fine di facilitarne l'analisi futura.

L'utilizzo di *Scikit-Sparse* Python fornisce librerie di default per trattare matrici con il metodo di Cholesky, tra esse è doveroso nominare **Numpy** (Harris et al. (2020)) e **SciPy** (Virtanen et al. (2020)). Sfortunatamente queste librerie non foniscono un metodo diretto per analizzare **matrici sparse**, per cui la scelta è ricaduta su una libreria **open-source** chiamata **Scikit-Sparse**. Essa si basa su *SciPy.Sparse* ma, al contrario di quest'ultima, offre funzioni veloci ed efficienti per trattare matrici sparse con Cholesky. Il codice 3 mostra l'implementazione effettuata tramite l'utilizzo della libreria sopra indicata. Si noti che tale funzione viene richiamata dal metodo __analyze(self, path) a riga *21* nel codice 2.

```python
def scikit_sparse_cholesky(A: csc_matrix, b):
    """

    If A is a sparse, symmetric, positive-definite matrix, and b is a matrix
    or vector (either sparse or dense), then the following code solves the
    equation Ax=b

    """
    try:
        factor = cholesky(A)
        x = factor(b)
        return x
    except Exception as e:
        raise Exception(f"Failed to execute cholesky\n Error Type: {e}")
```

Listing 3: scikit_sparse_cholesky function

L'utilizzo di Tracemalloc Per il montitoraggio della memoria utilizzata si è deciso di sfruttare la libreria tracemalloc. Essa offre un modo semplice e preciso per tenere traccia dei blocchi di memoria allocati durante l'analisi di cholesky. Un'altra libreria che è presa in considerazione durante la sperimentazione è psutil, i cui valori non sono stati ritenuti troppo realistici.

Problemi Riscontrati e limitazioni La principale problematica, riscontrata durante la progettazione dell'ecosistema Python, ricade principalmente nella struttura della libreria utilizzata. Sfortunatamente la compilazione di *Scikit-Sparse* non fornisce esito positivo in ambienti **Microsoft Windows**. I test sono stati effettuati mediante l'ausilio del package manager Pip e Anaconda, ma nessuno ha dato buon fine. Per questo motivo si è ritenuto opportuno l'utilizzo della **WSL** (*Windows Subsystem for Linux*) al fine di risolvere la problematica.

4.3 Salvataggio Dati

La tabella 10 rappresenta il dataset in formato *csv* generato dalla procedura di salvataggio dati. Di seguito verranno descritte tutte le label di interesse:

- **Name**: indica il nome della matrice analizzata
- **Rows**: indica il numero di righe della matrice
- **Columns**: indica il numero di colonne della matrice
- **Error**: indica l'errore relativo prodotto in relazione alla soluzione proposta.
- **Memory**: indica l'occupazione di memoria registrata durante l'esecuzione del metodo di cholesky.
- **Time**: indica la quantità di tempo registrata durante l'esecuzione del metodo di cholesky.
- **Language**: indica il linguaggio utilizzato per l'esecuzione del metodo di cholesky dulla matrice.
  - **Language 0**: Matlab
  - **Language 1**: Python
- **OS**: indica il sistema operativo su cui è stato eseguito il tutto.
  - **OS 0**: Windows
  - **OS 1**: Linux
- **Nnz**: numero di elementi diversi da zero all'interno della matrice.
- **Cond**: numero di condizionamento della matrice.

Name	Rows	Columns	Error	Memory	Time	Language	OS	Nnz	Cond
apache2	715176	715176	2.235941e-08	0.004471	66.232192	1	0	4817870	5.316861e+06
apache2	715176	715176	4.388900e-11	2877.018100	13.430700	0	0	4817870	5.316861e+06
apache2	715176	715176	2.235941e-08	131.386811	83.372801	1	1	4817870	5.316861e+06
apache2	715176	715176	4.388900e-11	2769.992000	15.145300	0	1	4817870	5.316861e+06
cfd1	70656	70656	6.628463e-12	0.003326	4.461982	1	0	1825580	1.335081e+06
cfd1	70656	70656	1.135900e-13	577.585200	2.557600	0	0	1825580	1.335081e+06
cfd1	70656	70656	6.628463e-12	49.466571	5.382657	1	1	1825580	1.335081e+06
cfd1	70656	70656	1.135900e-13	560.592000	2.723700	0	1	1825580	1.335081e+06
cfd2	123440	123440	1.334161e-10	0.003326	10.917099	1	0	3085406	3.728473e+06
cfd2	123440	123440	3.348600e-13	1187.708900	6.083400	0	0	3085406	3.728473e+06
cfd2	123440	123440	1.334161e-10	0.003326	13.483177	1	1	3085406	3.728473e+06
cfd2	123440	123440	3.348600e-13	1145.468000	6.241800	0	1	3085406	3.728473e+06
ex15	6867	6867	5.281170e-05	0.003326	0.015078	1	0	98671	1.432642e+13
ex15	6867	6867	6.348200e-07	3.616800	0.030822	0	0	98671	1.432642e+13
ex15	6867	6867	5.281170e-05	2.664623	0.022156	1	1	98671	1.432642e+13
ex15	6867	6867	6.348200e-07	0.000000	0.024699	0	1	98671	1.432642e+13
G3_circuit	1585478	1585478	3.452568e-09	210.620559	27.939265	1	0	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.576600e-12	3262.529500	19.503700	0	0	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.452568e-09	0.003326	33.810687	1	1	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.576600e-12	3203.660000	18.883200	0	1	7660826	2.238425e+07
parabolic_fem	525825	525825	8.836779e-10	0.003326	6.439602	1	0	3674625	2.110820e+05
parabolic_fem	525825	525825	1.050000e-12	607.600600	3.033300	0	0	3674625	2.110820e+05
parabolic_fem	525825	525825	8.836779e-10	0.003326	7.920758	1	1	3674625	2.110820e+05
parabolic_fem	525825	525825	1.050000e-12	538.452000	3.286900	0	1	3674625	2.110820e+05
shallow_water1	81920	81920	6.923992e-14	9.067514	0.615064	1	0	327680	3.628000e+00
shallow_water1	81920	81920	2.672800e-16	41.566200	0.254370	0	0	327680	3.628000e+00
shallow_water1	81920	81920	6.923992e-14	0.003326	0.745701	1	1	327680	3.628000e+00
shallow_water1	81920	81920	2.672800e-16	0.000000	0.263050	0	1	327680	3.628000e+00

Tabella 10: Global Dataset

4.4 Analisi Dei Dati

Nella seguente sottosezione verranno trattate le procedure di progettazione utilizzate durante l'analisi dei dati prodotti, in particolare le librerie utilizzate. Nello specifico i dati analizzati saranno quelli provenienti dai file **.csv*. Per far ciò è stato sfruttato *JupyterBook* Perez et al. (2011) mediante il linguaggio Python. Di seguito viene proposta la struttura dell'intero progetto di analisi.

```
PythonAnalysis
└─ src
   ├─ model
   │  ├─ __init__.py
   │  ├─ additional_analysis.ipynb
   │  ├─ backslash_comparison.ipynb
   │  ├─ LanguageCompare.ipynb
   │  └─ OSCompare.ipynb
   └─ Resources
      ├─ data.csv
      └─ dataBackslash.csv
```

Si noti che l'analisi è stata volutamente strutturata in diverse sotto-analisi:

- Analisi dei Linauggi
- Analisi degli OS
- Confronto metodo *Chol* con *Backslash* in Matlab
- Analisi aggiuntive e di interesse

4.4.1 Librerie utilizzate

Le librerie sfruttate allo scopo sono molteplici, ma si vuole sottolineare l'importanza di *matplotlib* Hunter (2007) e *pandas* pandas development team (2020) Wes McKinney (2010). Allo scopo sono state realizzate delle funzioni per la creazione **omogenea** di tutti i grafici utilizzati durante l'analisi. Il codice 4 ne fornisce un esempio implementativo.

```
def plot(labels, data_py, data_matlab, x_name, y_name, title, legend1, legend2,
    path, **keywords):
    figure(figsize=(20, 6), dpi=80)
    title = title + " " + "(Line Plot)"

    plt.plot(labels, data_py, label=legend1, linestyle="-")
    plt.plot(labels, data_matlab, label=legend2, linestyle="--")
    show(x_name, y_name, title, path, **keywords)


def bar_plot(labels, data_py, data_matlab, x_name, y_name, title, legend1, legend2
    , path, **keywords):
    figure(figsize=(20, 6), dpi=80)
    title = title + " " + "(Bar Plot)"

    x_axis = np.arange(len(labels))

    # Multi bar Chart

    plt.bar(x_axis - 0.2, data_py, width=0.4, label=legend1)
    plt.bar(x_axis + 0.2, data_matlab, width=0.4, label=legend2)

    # Xticks

    plt.xticks(x_axis, labels)
    show(x_name, y_name, title, path, **keywords)


def show(x_name, y_name, title, path, **keywords):
    x_label, y_label = "", ""
    if X_LABEL in keywords.keys():
        x_label = f"({keywords[X_LABEL]})"
    if Y_LABEL in keywords.keys():
        y_label = f"({keywords[Y_LABEL]})"
    if LOG_SCALE in keywords.keys():
        plt.yscale('log')  # logarithmic scale

    plt.xlabel(f"{x_name} {x_label}", labelpad=15, fontsize=12, color="#333533")
    plt.ylabel(f"{y_name} {y_label}", labelpad=15, fontsize=12, color="#333533")
    plt.title(title, fontsize=18, color="#333533", pad=35)

    # removing axes from the figure
    plt.gca().spines['right'].set_visible(False)
    plt.gca().spines['top'].set_visible(False)
    plt.legend(loc='upper center', ncol=2, frameon=False)

    plt.savefig(path)
    plt.show()
```

Listing 4: Plot functions

5 Risulti Sperimentali

In questa sezione saranno principalmente analizzati i risultati ottenuti dalle esecuzioni in MATLAB, attraverso il metodo *chol()*, e in Python con la libreria *Scikit-Sparse*, su entrambi i Sitemi Operativi.
Il confronto sarà effettuato per mezzo di 3 metriche: il tempo di esecuzione per la risoluzione del sistema lineare, la quantità di memoria utilizzata e l'errore relativo derivante dalla risoluzione del sistema. Quest'ultimo viene calcolato calcolato come segue:

$$\frac{\|x - x_e\|_2}{\|x\|_2}$$

Tutti i paragoni sono graficamente espressi attraverso 2 tipi di plot, che usati insieme forniscono una migliore interpretazione dei risultati: i line plot esprimono meglio la forma della distribuzione dei dati, mentre i bar plot rendono un'idea migliore sulla differenza dei valori confrontati.

In aggiunta, saranno esposte altre analisi che cercano di dare un'interpretazione più accurata di alcuni risultati ottenuti, tra cui il confronto tra le implementazioni in MATLAB dell'algoritmo *chol()* e il *backslash* built-in. Da sottolineare che i confronti tra Python e MATLAB sono stati eseguiti mediante la prima dell due funzioni, come verrà esposto in seguito più nel dettaglio.

È importante anticipare che, delle 9 matrici analizzate, sono disponibili i risultati delle prime 7, in quanto la *Flan-1565* e la *StocF-1465* si sono rilevate essere di dimensioni troppo elevate per essere risolte, sia in Linux che in Windows e per entrambi i linguaggi.

5.1 Confronto OS: Windows e Linux

Per confrontare i due sistemi operativi è stata eseguita l'analisi al variare dei linguaggi di programmazione. Dal dataset precedentemente descritto si può notare la colonna OS, indicante la tipologia di sistema operativo su cui è stata eseguita l'analisi della corrispettiva riga. In questa trattazione ci concentreremo sui risultati dati fissando *Language=1* (Python) e *Language=0* (Matlab) e facendo variare la label *OS* inerente al sistema operativo in analisi.

5.1.1 Risultati sperimentali in ambiente MATLAB

Name	Rows	Columns	Error	Memory	Time	Language	OS	Nnz	Cond
ex15	6867	6867	6.348200e-07	3.6168	0.030822	0	0	98671	1.432642e+13
ex15	6867	6867	6.348200e-07	0.0000	0.024699	0	1	98671	1.432642e+13
cfd1	70656	70656	1.135900e-13	577.5852	2.557600	0	0	1825580	1.335081e+06
cfd1	70656	70656	1.135900e-13	560.5920	2.723700	0	1	1825580	1.335081e+06
shallow_water1	81920	81920	2.672800e-16	41.5662	0.254370	0	0	327680	3.628000e+00
shallow_water1	81920	81920	2.672800e-16	0.0000	0.263050	0	1	327680	3.628000e+00
cfd2	123440	123440	3.348600e-13	1187.7089	6.083400	0	0	3085406	3.728473e+06
cfd2	123440	123440	3.348600e-13	1145.4680	6.241800	0	1	3085406	3.728473e+06
parabolic_fem	525825	525825	1.050000e-12	607.6006	3.033300	0	0	3674625	2.110820e+05
parabolic_fem	525825	525825	1.050000e-12	538.4520	3.286900	0	1	3674625	2.110820e+05
apache2	715176	715176	4.388900e-11	2769.9920	15.145300	0	1	4817870	5.316861e+06
apache2	715176	715176	4.388900e-11	2877.0181	13.430700	0	0	4817870	5.316861e+06
G3_circuit	1585478	1585478	3.576600e-12	3262.5295	19.503700	0	0	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.576600e-12	3203.6600	18.883200	0	1	7660826	2.238425e+07

Tabella 11: MATLAB Dataset

Analisi Errore Come ovvio che sia, non c'è alcuna differenza negli errori relativi nelle esecuzioni sui diversi sistemi operativi dell'algoritmo *chol()* di MATLAB: questo perchè l'implementazione è esattamente la stessa e non possono esserci differenze nel calcolo del sistema lineare.

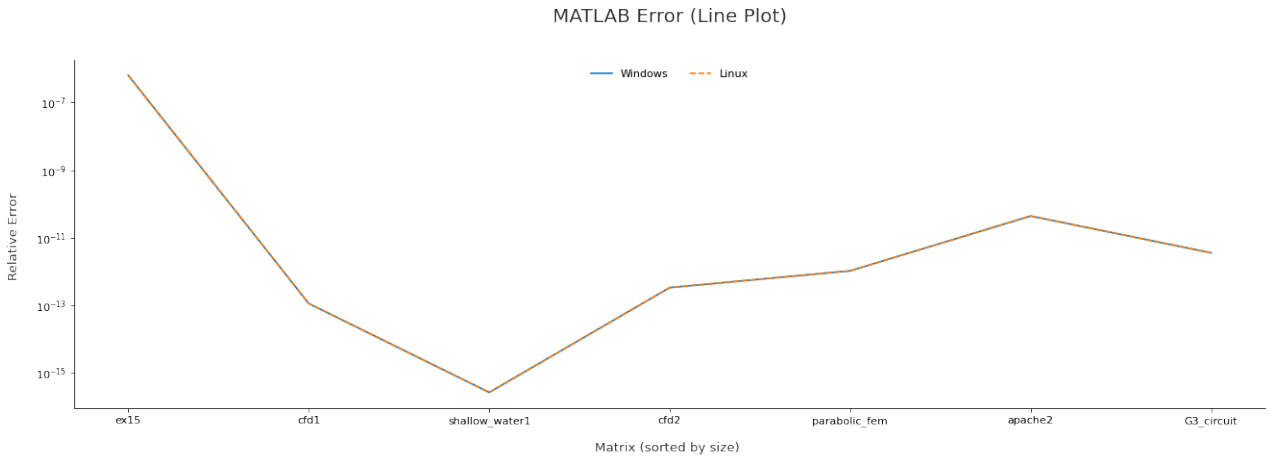


Figura 16: Line Plot errore dei sistemi operativi in ambiente MATLAB

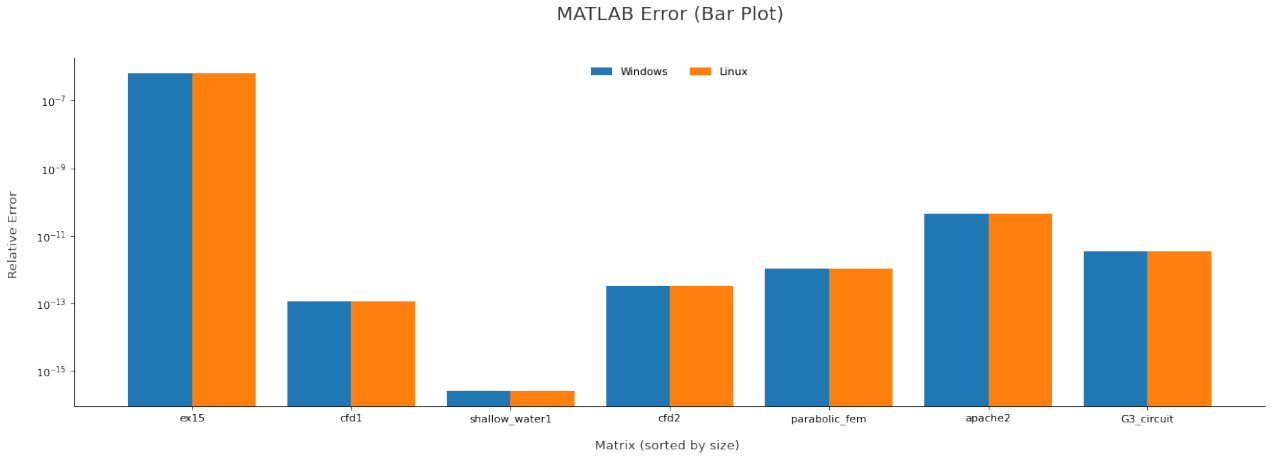


Figura 17: Line Plot errore dei sistemi operativi in ambiente MATLAB

Analisi Memoria In quanto alla memoria, eseguendo l'algoritmo di MATLAB su Linux viene utilizzata leggermente meno memoria, con una media di 48MB in meno rispetto all'esecuzione in Windows.
Risultato poco determinante nel confronto, e che potrebbe essere influenzato dalla differenza nel calcolo della memoria nei due sistemi operativi

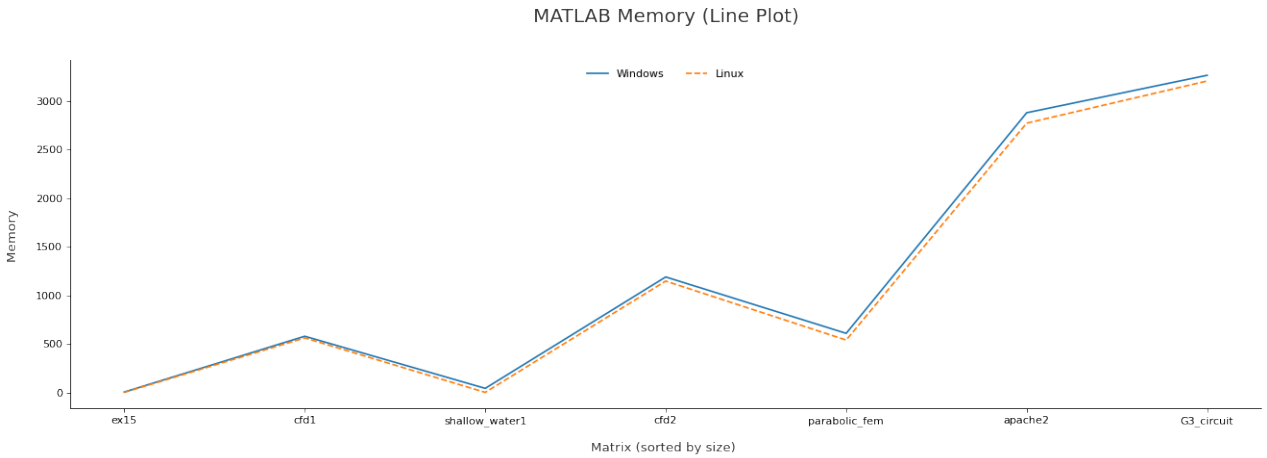


Figura 18: Line Plot memoria dei sistemi operativi in ambiente MATLAB

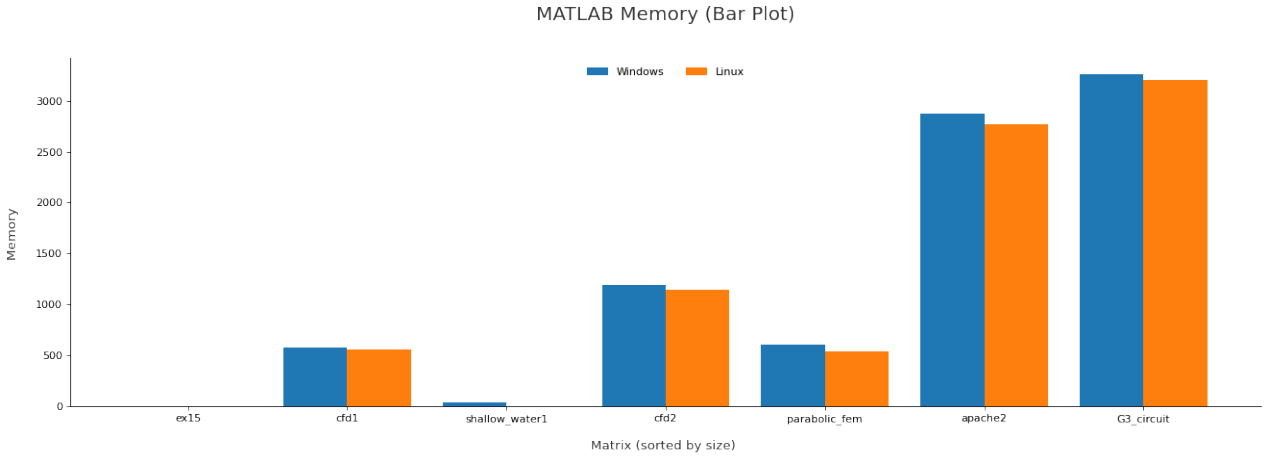


Figura 19: Line Plot memoria dei sistemi operativi in ambiente MATLAB

Analisi Tempo Anche il tempo di esecuzione è molto simile nei due sitemi operativi: per le prime 5 matrici ci sono differenze nell'ordine dei centesimi di secondo.
Per la risoluzione di *apache2* Linux impiega solo 1.7 secondi in più, mentre per *G3_circuit* è Windows ad impiegare 0.6 secondi in più. Complessivamente, Windows richiede in media 0.24 secondi in meno per la computazione, anche qui risultato poco determinante in termini di paragone tra i due sistemi operativi.

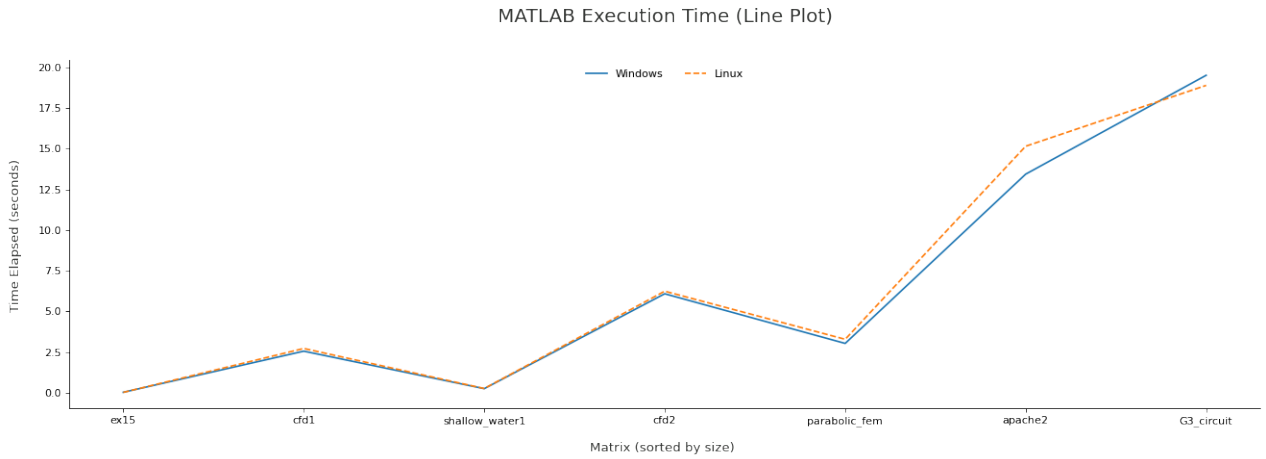


Figura 20: Line Plot tempo dei sistemi operativi in ambiente MATLAB

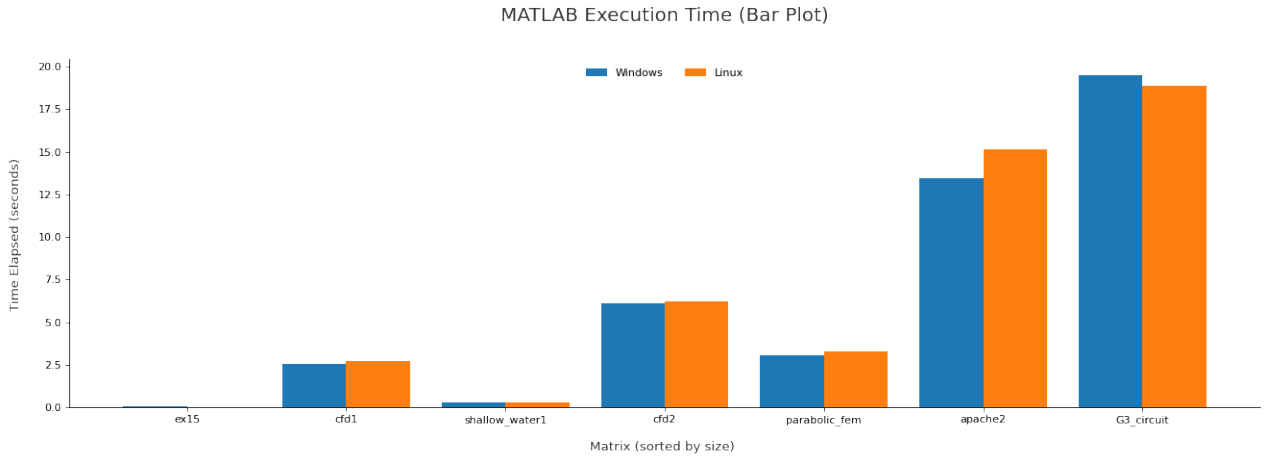


Figura 21: Line Plot tempo dei sistemi operativi in ambiente MATLAB

5.1.2 Risultati sperimentali in ambiente Python

Name	Rows	Columns	Error	Memory	Time	Language	OS	Nnz	Cond
ex15	6867	6867	5.281170e-05	2.664623	0.022156	1	1	98671	1.432642e+13
ex15	6867	6867	5.281170e-05	0.003326	0.015078	1	0	98671	1.432642e+13
cfd1	70656	70656	6.628463e-12	0.003326	4.461982	1	0	1825580	1.335081e+06
cfd1	70656	70656	6.628463e-12	49.466571	5.382657	1	1	1825580	1.335081e+06
shallow_water1	81920	81920	6.923992e-14	9.067514	0.615064	1	0	327680	3.628000e+00
shallow_water1	81920	81920	6.923992e-14	0.003326	0.745701	1	1	327680	3.628000e+00
cfd2	123440	123440	1.334161e-10	0.003326	13.483177	1	1	3085406	3.728473e+06
cfd2	123440	123440	1.334161e-10	0.003326	10.917099	1	0	3085406	3.728473e+06
parabolic_fem	525825	525825	8.836779e-10	0.003326	6.439602	1	0	3674625	2.110820e+05
parabolic_fem	525825	525825	8.836779e-10	0.003326	7.920758	1	1	3674625	2.110820e+05
apache2	715176	715176	2.235941e-08	131.386811	83.372801	1	1	4817870	5.316861e+06
apache2	715176	715176	2.235941e-08	0.004471	66.232192	1	0	4817870	5.316861e+06
G3_circuit	1585478	1585478	3.452568e-09	210.620559	27.939265	1	0	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.452568e-09	0.003326	33.810687	1	1	7660826	2.238425e+07

Tabella 12: Python Dataset

Analisi Errore Così come per MATLAB, gli errori relativi sono esattamente gli stessi per lo stesso motivo di implementazione.

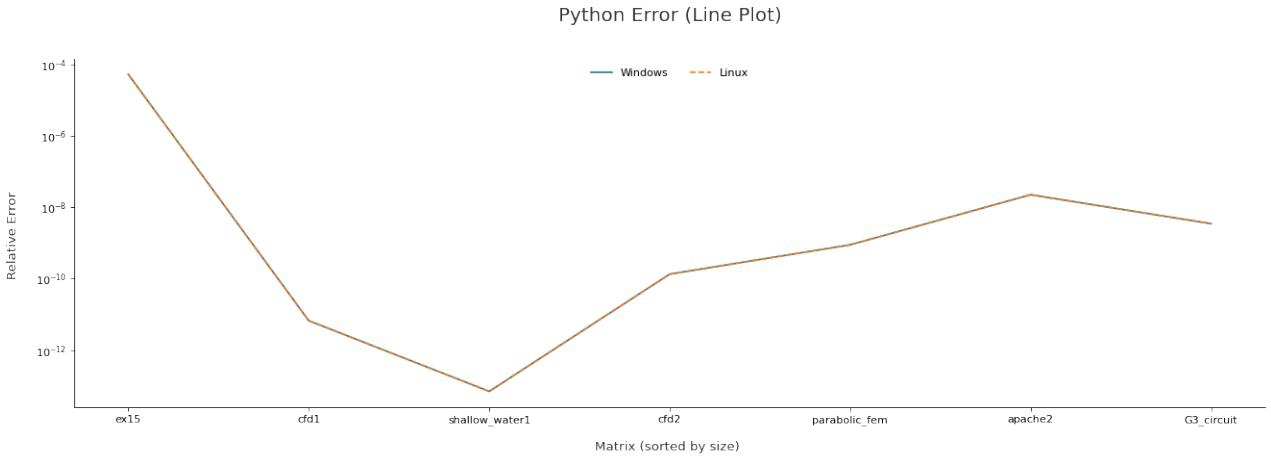


Figura 22: Line Plot errore dei sistemi operativi in ambiente Python

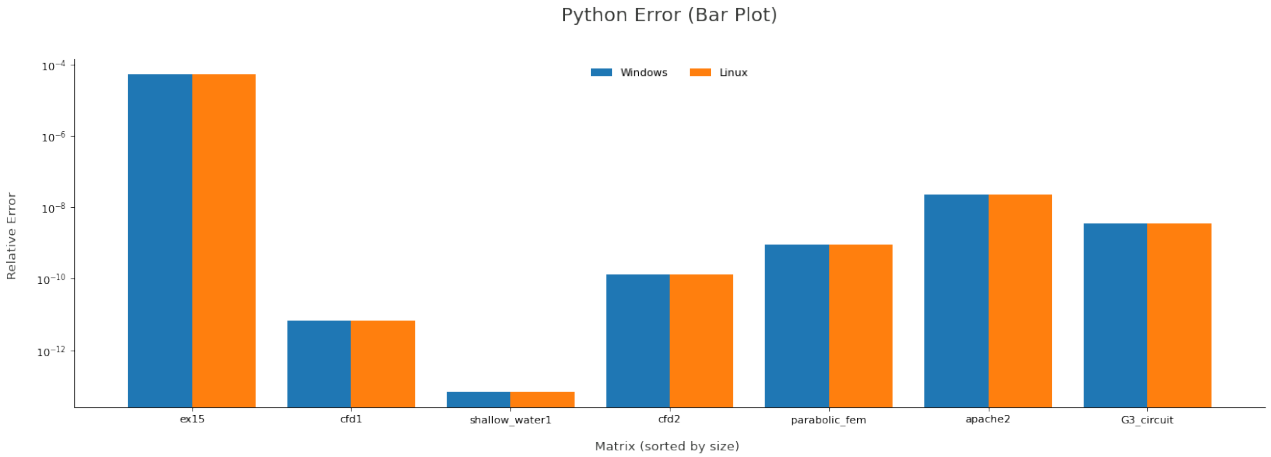


Figura 23: Bar Plot errore dei sistemi operativi in ambiente Python

Analisi Memoria L'utilizzo della memoria nell'esecuzione in Python è quella che ha restituito risultati più variabili, e forse meno affidabili nel caso delle ultime due matrici.

Come si vede nelle figure 24 e 25 per le matrici più piccole il consumo della memoria è molto ridotto, tendente allo 0.
Per *apache2* e *G3_circuit* si ottengono invece risultati discordanti, in quanto sulla prima Windows performa decisamente meglio, mentre per la seconda c'è il trend diametralmente opposto.

Tali risultati possono probabilmente essere attribuiti al metodo di calcolo della memoria utilizzata.

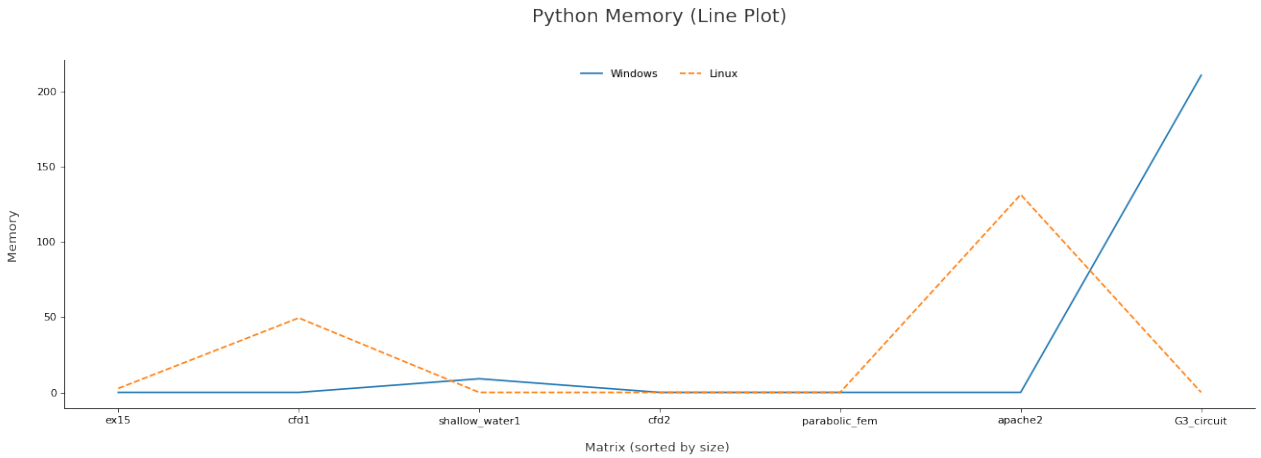


Figura 24: Line Plot memoria dei sistemi operativi in ambiente Python

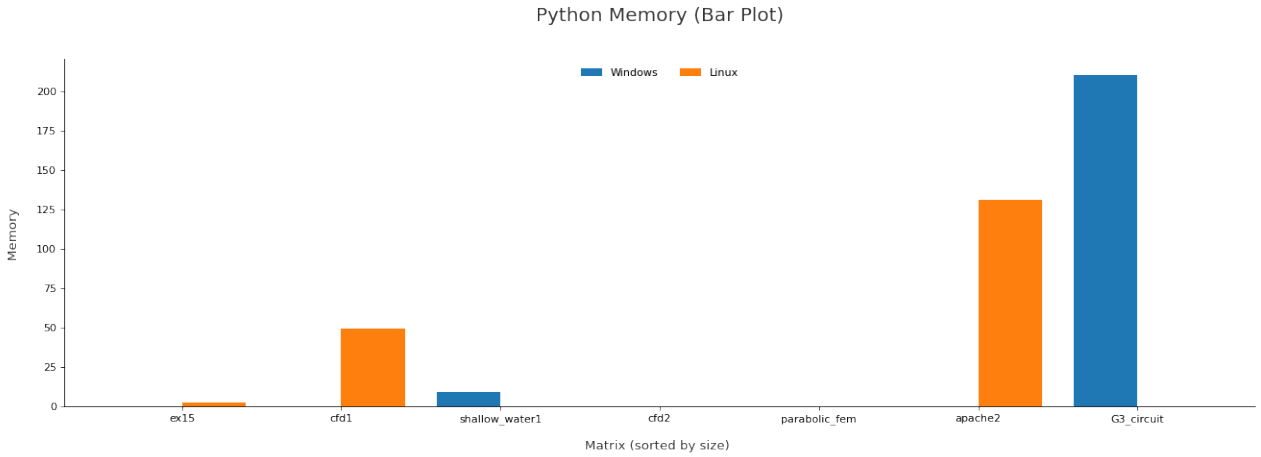


Figura 25: Bar Plot memoria dei sistemi operativi in ambiente Python

Analisi Tempo Il tempo di esecuzione in Python mostra risultati migliori per Windows: al di là di 2 delle matrici più piccole, *ex15* e *shallow_water1*, per cui la differenza è espressa in decimi di secondo, per tutte le altre c'è almeno 1 secondo di differenza, con una differenza più marcata per *G3_circuit*, con 6 secondi di differenza, e *apache2*, in cui con Linux si impiegano ben 17 secondi in più.
In media l'esecuzione su Windows richiede 4 secondi in meno, pertanto risulta più performante per le esecuzioni in ambiente Python.

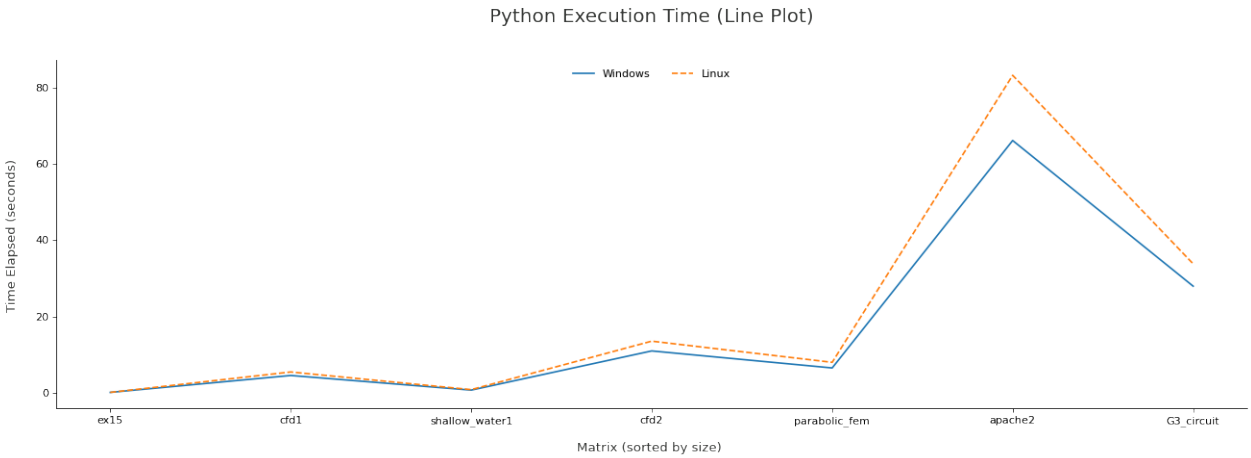


Figura 26: Line Plot tempo dei sistemi operativi in ambiente Python

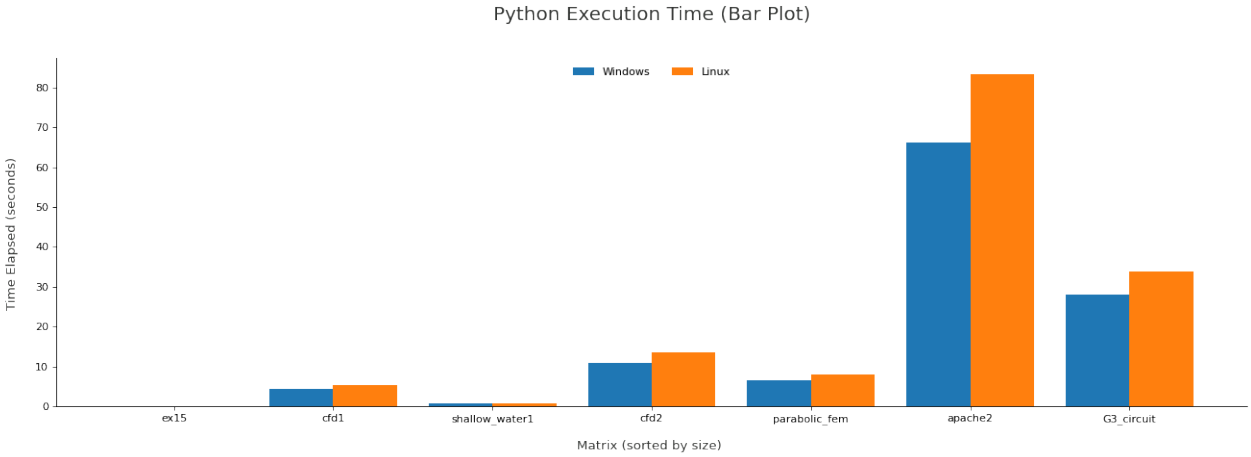


Figura 27: Bar Plot tempo dei sistemi operativi in ambiente Python

5.1.3 Resoconto e Considerazioni Finali

Errore Dalle analisi è emerso che la variabile inerente all'**errore** tra le soluzioni è chiaramente indipendente dal sistema operativo su cui vengono eseguite le analisi.

Memoria Non volendo sbilanciarsi si può anche ipotizzare che, in un ambiente primo di processi interagenti, Linux potrebbe generalmente comportarsi meglio sul costo della **memoria**. Sebbene si vuole precisare che quest'ultima analisi verrà discussa in seguito più nel dettaglio poichè potrebbe dipendere molto dal linguaggio su cui è stato effeattuato un test, in particolar modo su come le corrispettive librerie calcolano la memoria occupata da un determinato processo.

Tempo Tutti i dati analizzati hanno anche mostrato che in ambiente Windows il **tempo** si esecuzione sia generalmente minore indipendentemente dal linguaggio sfruttato per l'analisi.

5.2 Confronto Linguaggi: Python e Matlab

Per confrontare i due linguaggi si è deciso di eseguire l'analisi al variare dei sistemi operativi. Dal dataset precedentemente descritto si può notare la colonna OS, indicante la tipologia di sistema operativo su cui è stata eseguita l'analisi della corrispettiva riga. In questa trattazione ci concentreremo sui risultati dati fissando

$OS = 1$ (Linux) o $OS = 0$ (Windows). Si noti che, sebbene il valore di OS sia costante per ogni riga, la variabile inerente a $Language$ cambi continuamente in relazione al linguaggio analizzato: $Language = 1$ (Python), $Language = 0$ (Matlab).

5.2.1 Risultati sperimentali in ambiente Windows

Name	Rows	Columns	Error	Memory	Time	Language	OS	Nnz	Cond
apache2	715176	715176	2.235941e-08	0.004471	66.232192	1	0	4817870	5.316861e+06
apache2	715176	715176	4.388900e-11	2877.018100	13.430700	0	0	4817870	5.316861e+06
cfd1	70656	70656	6.628463e-12	0.003326	4.461982	1	0	1825580	1.335081e+06
cfd1	70656	70656	1.135900e-13	577.585200	2.557600	0	0	1825580	1.335081e+06
cfd2	123440	123440	1.334161e-10	0.003326	10.917099	1	0	3085406	3.728473e+06
cfd2	123440	123440	3.348600e-13	1187.708900	6.083400	0	0	3085406	3.728473e+06
ex15	6867	6867	5.281170e-05	0.003326	0.015078	1	0	98671	1.432642e+13
ex15	6867	6867	6.348200e-07	3.616800	0.030822	0	0	98671	1.432642e+13
G3_circuit	1585478	1585478	3.452568e-09	210.620559	27.939265	1	0	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.576600e-12	3262.529500	19.503700	0	0	7660826	2.238425e+07
parabolic_fem	525825	525825	8.836779e-10	0.003326	6.439602	1	0	3674625	2.110820e+05
parabolic_fem	525825	525825	1.050000e-12	607.600600	3.033300	0	0	3674625	2.110820e+05
shallow_water1	81920	81920	6.923992e-14	9.067514	0.615064	1	0	327680	3.628000e+00
shallow_water1	81920	81920	2.672800e-16	41.566200	0.254370	0	0	327680	3.628000e+00

Tabella 13: Windows Dataset

Analisi Errore A sostegno dello studio effettutato per analizzare l'errore delle corrispettive soluzione nei vari linguaggi, si è calcolato l'errore relativo esposto in precedenza. Si forniscono i grafici 29 e 28 da cui si può facilmente notare che l'indagine tramite Python fornisce risultati di errore generalmente superiori rispetto al concorrente. Dalle figure si può notare che non esiste una forma particolare di correlazione tra l'errore calcolato e le dimensioni delle matrici, questo perchè si dovrebbe tenere in considerazione la sparsità e il coefficiente di condizionamento delle matrici. Nello specifico la matrice che ha fornito un valore di errore maggiore è stata $ex15$, di cui si parlerà in seguito.

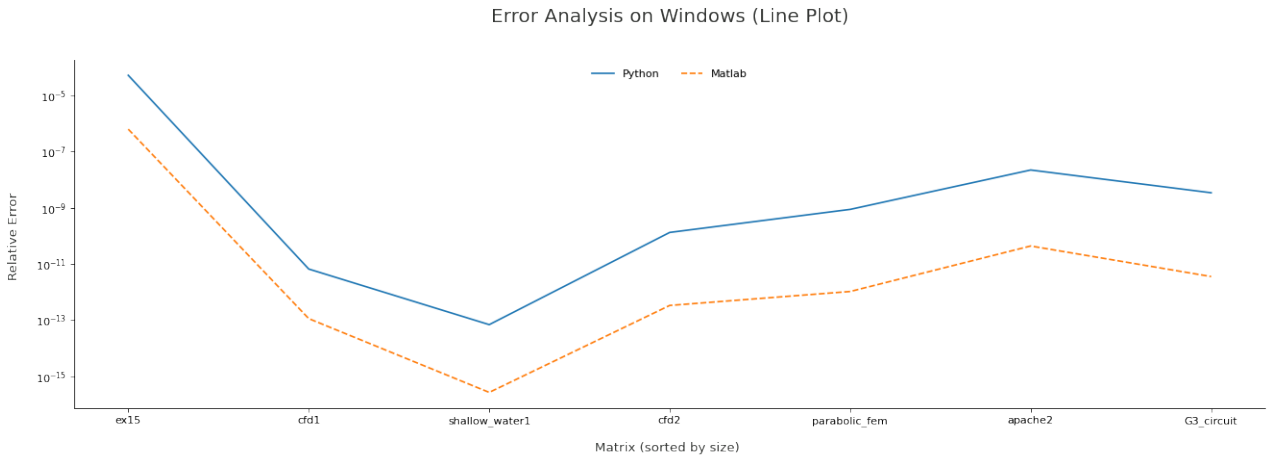


Figura 28: Line Plot errore dei linguaggi in ambiente Windows

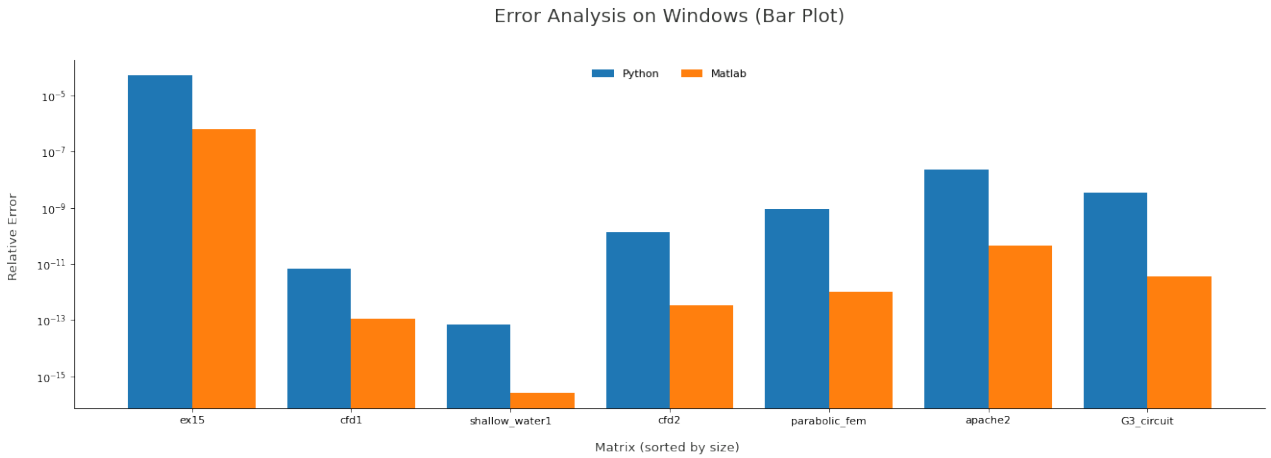


Figura 29: Bar plot errore dei linguaggi in ambiente Windows

Analisi Memoria Come specificato precedentemente, lo studio della memoria su linguaggio Python è stato eseguito mediante la libreria tracemalloc. Dalle figure 31 e 30 si può notare come Matlab occupi uno spazio estremamente maggiore, rispetto a Python, durante l'esecuzione. Questo risultato può essere giustificato da motivi tecnici dovuti alla differenza con cui la quantità di memoria utilizzata viene calcolata. Difatti l'algoritmo per il calcolo della memoria allocata è basato sulla differenza tra la memoria prima e dopo l'esecuzione del metodo di Cholesky.In conclusione è doveroso annotare che questo studio dovrebbe essere approfondito in futuro, magari utilizzando modi più accurati per tracciare l'andamento dell'allocazione di blocchi di memoria in Matlab.

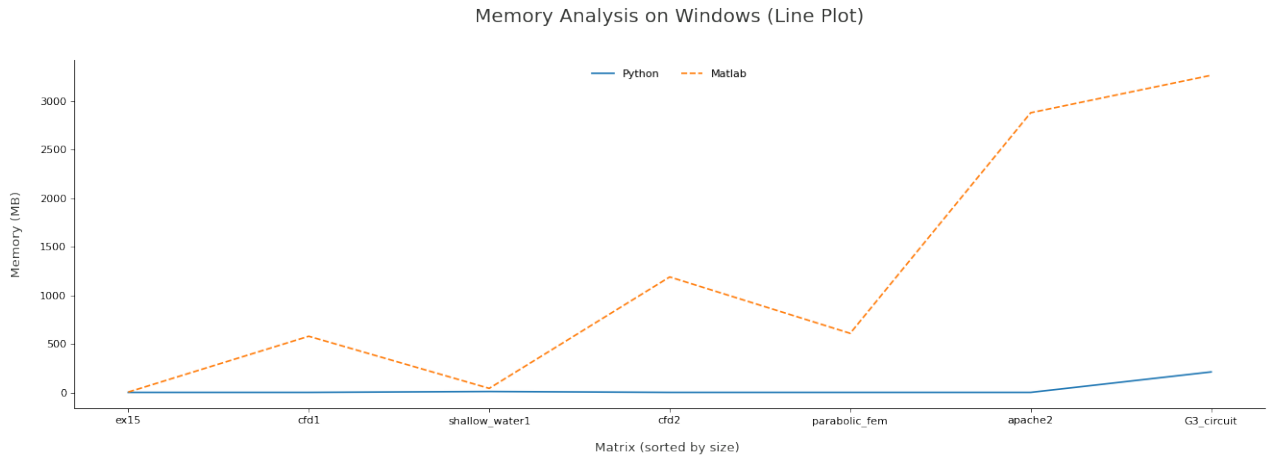


Figura 30: Line Plot memoria dei linguaggi in ambiente Windows

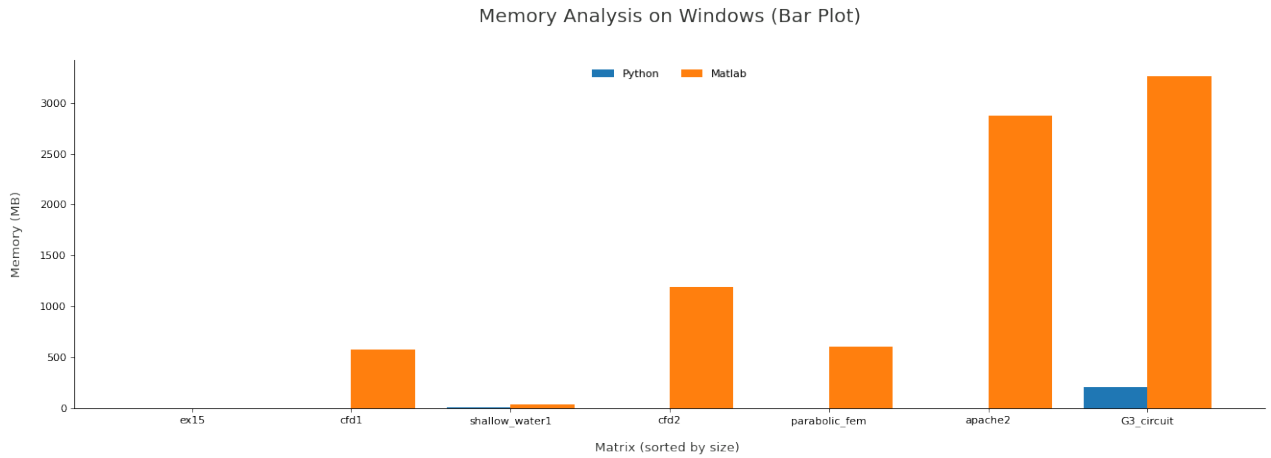


Figura 31: Bar plot memoria dei linguaggi in ambiente Windows

Analisi Tempo L'analisi del tempo ha fornito elementi abbastanza interessanti. Come si può notare dalle figure 33 e 32 generalmente Matlab processa i dati in tempo sicuramente minore rispetto a Python. Si può pensare che tempo e dimensioni delle matrici siano correlate direttamente tra loro, questo non è generalmente vero in questa situazione: la matrice *Apache2* sembrerebbe la più difficile da processare sebbene sia più piccola di *G3_circuit*, in questo caso Python impiega circa 6 volte il tempo che impiega Matlab.

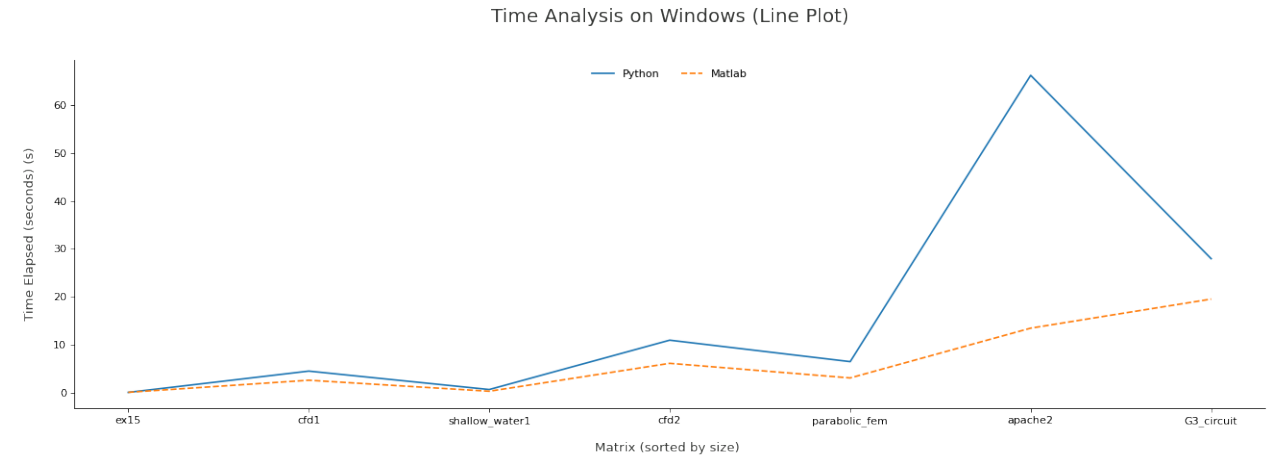


Figura 32: Line Plot tempo dei linguaggi in ambiente Windows

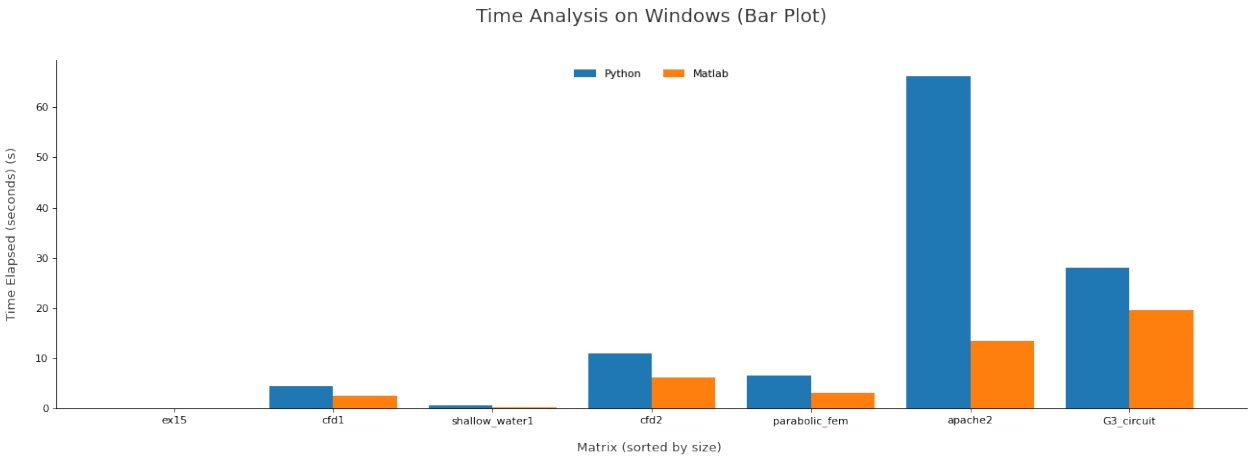


Figura 33: Bar plot tempo dei linguaggi in ambiente Windows

5.2.2 Risultati sperimentali in ambiente Linux

Proprio come in ambiente Windows, la tabella 14 mostra il dataset prodotto selezioanando le righe $OS = 1$.

Name	Rows	Columns	Error	Memory	Time	Language	OS	Nnz	Cond
apache2	715176	715176	2.235941e-08	131.386811	83.372801	1	1	4817870	5.316861e+06
apache2	715176	715176	4.388900e-11	2769.992000	15.145300	0	1	4817870	5.316861e+06
cfd1	70656	70656	6.628463e-12	49.466571	5.382657	1	1	1825580	1.335081e+06
cfd1	70656	70656	1.135900e-13	560.592000	2.723700	0	1	1825580	1.335081e+06
cfd2	123440	123440	1.334161e-10	0.003326	13.483177	1	1	3085406	3.728473e+06
cfd2	123440	123440	3.348600e-13	1145.468000	6.241800	0	1	3085406	3.728473e+06
ex15	6867	6867	5.281170e-05	2.664623	0.022156	1	1	98671	1.432642e+13
ex15	6867	6867	6.348200e-07	0.000000	0.024699	0	1	98671	1.432642e+13
G3_circuit	1585478	1585478	3.452568e-09	0.003326	33.810687	1	1	7660826	2.238425e+07
G3_circuit	1585478	1585478	3.576600e-12	3203.660000	18.883200	0	1	7660826	2.238425e+07
parabolic_fem	525825	525825	8.836779e-10	0.003326	7.920758	1	1	3674625	2.110820e+05
parabolic_fem	525825	525825	1.050000e-12	538.452000	3.286900	0	1	3674625	2.110820e+05
shallow_water1	81920	81920	6.923992e-14	0.003326	0.745701	1	1	327680	3.628000e+00
shallow_water1	81920	81920	2.672800e-16	0.000000	0.263050	0	1	327680	3.628000e+00

Tabella 14: Linux Dataset

Analisi Errore Dalle figure 35 e 34 si può notare che, come succedeva in ambiente Windows, anche qui Python fornisce un errore generalmente superiore rispetto a Matlab. Sebbene non sia questo lo scopo di questa sezione, è opportuno comunque osservare, rispetto alle analisi della sezione precedente, che non si è riscontrata una differenza marcata tra i due sistemi operativi.

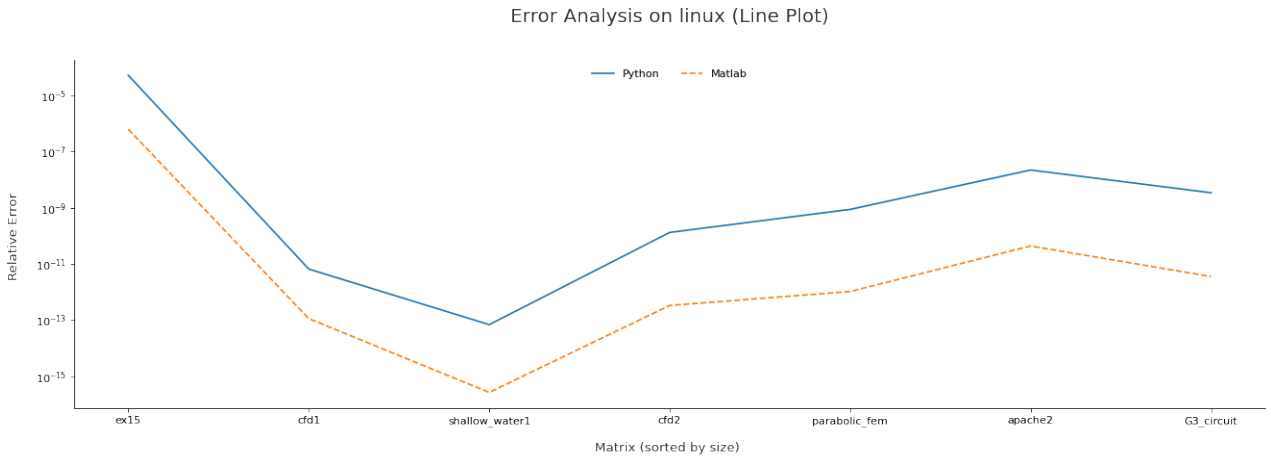


Figura 34: Line Plot errore dei linguaggi in ambiente linux

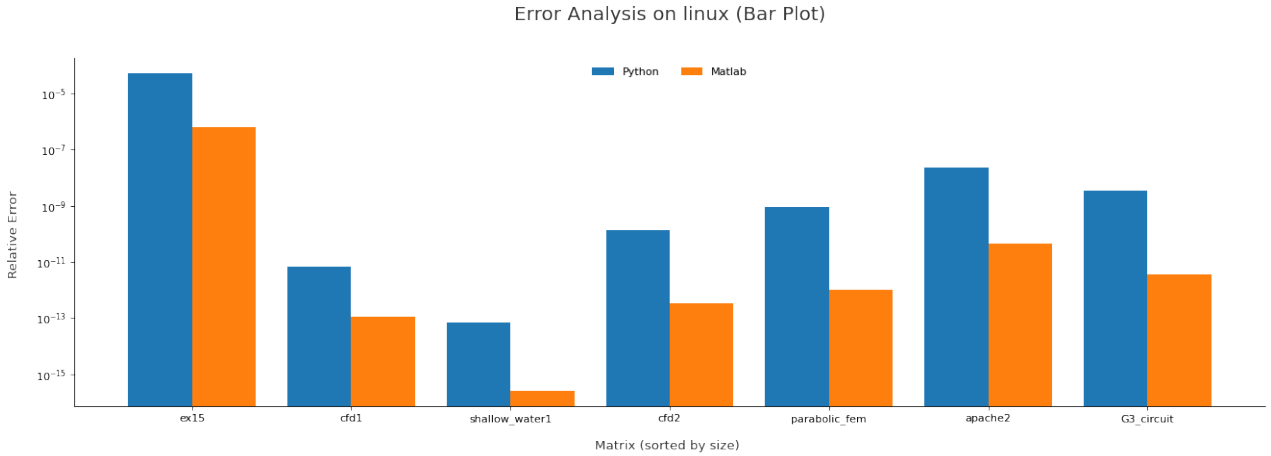


Figura 35: Bar plot errore dei linguaggi in ambiente linux

Analisi Memoria Dalle figure 37 e 36 si può notare che Matlab sfrutta molta più memoria rispetto al concorrente. Come nel caso dell'analisi in ambiente Windows, le matrici più sovradimensionate arrivano ad occupare fino a 3GB di RAM. Al contrario del caso precedente, Python occupa necessita di più memoria per la matrice *apache2* rispetto che a *G3_circuit*. Si vuole comunque sottolineare che non sussiste una differenza notevole rispetto all'analisi fornita in ambiente Windows.

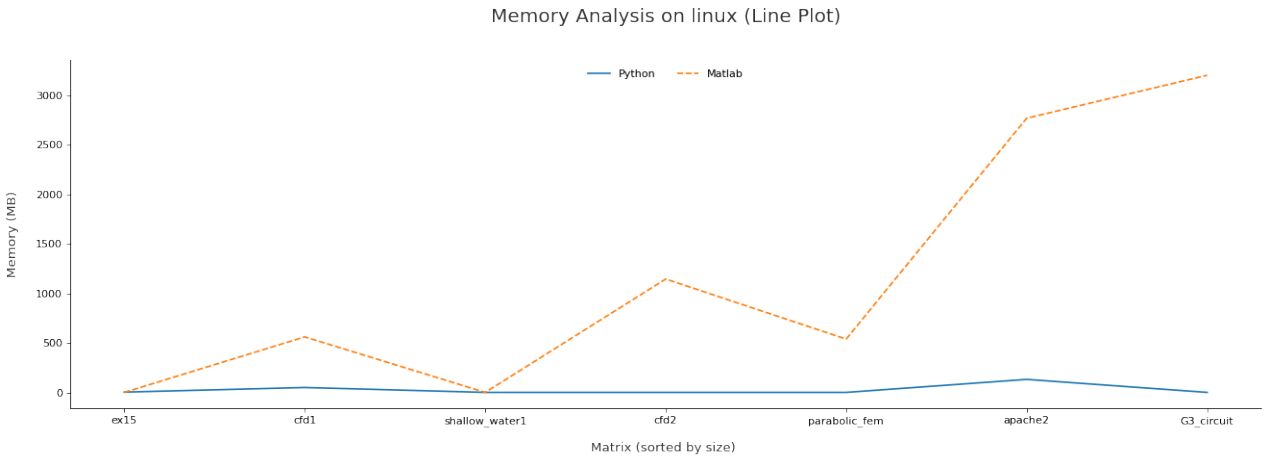


Figura 36: Line Plot memoria dei linguaggi in ambiente linux

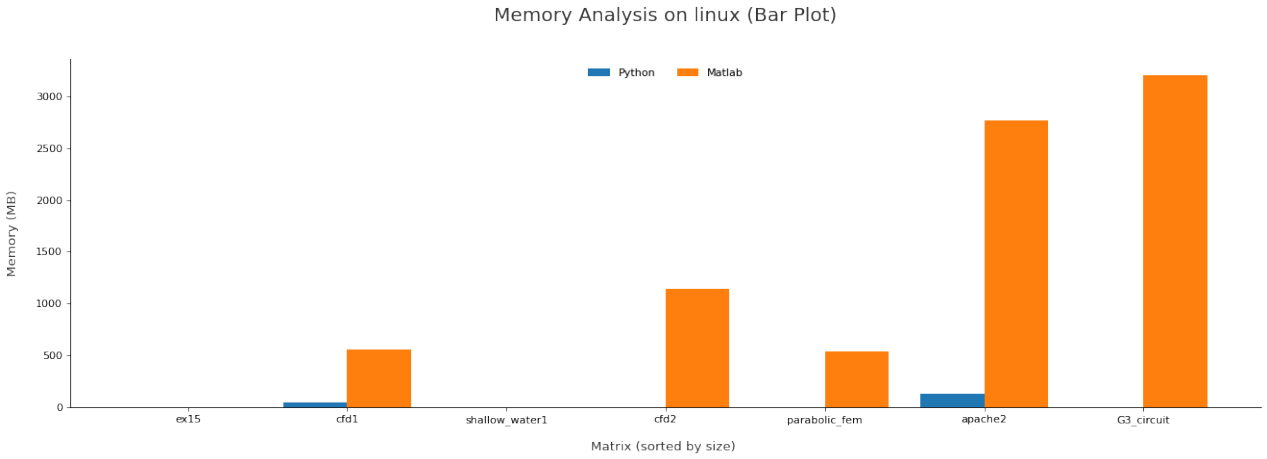


Figura 37: Bar plot memoria dei linguaggi in ambiente linux

Analisi Tempo Dalle figure 39 e 38 si può notare anche questa volta che Python generalmente impiega più tempo d'esecuzione rispetto a Matlab.

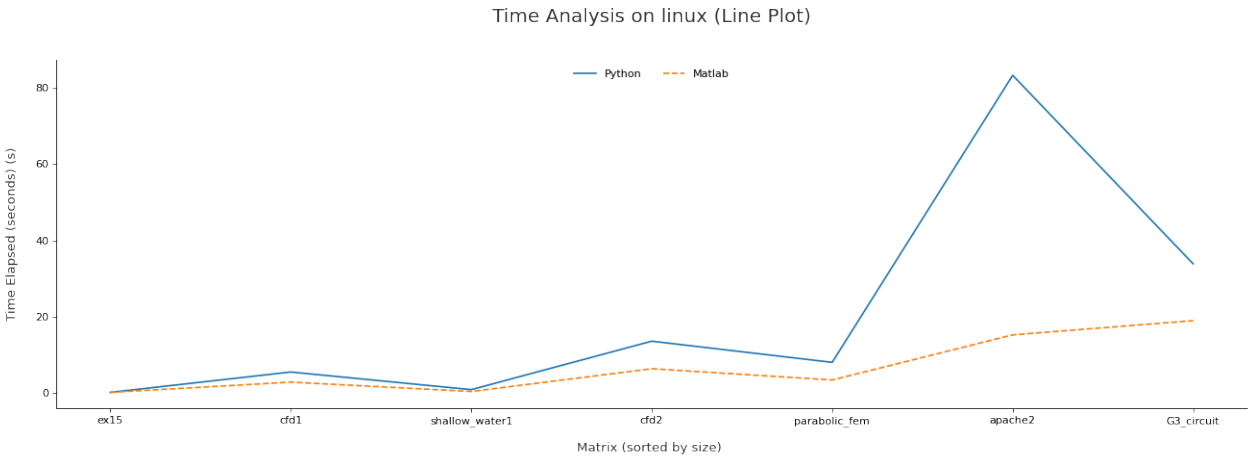


Figura 38: Line Plot tempo dei linguaggi in ambiente linux

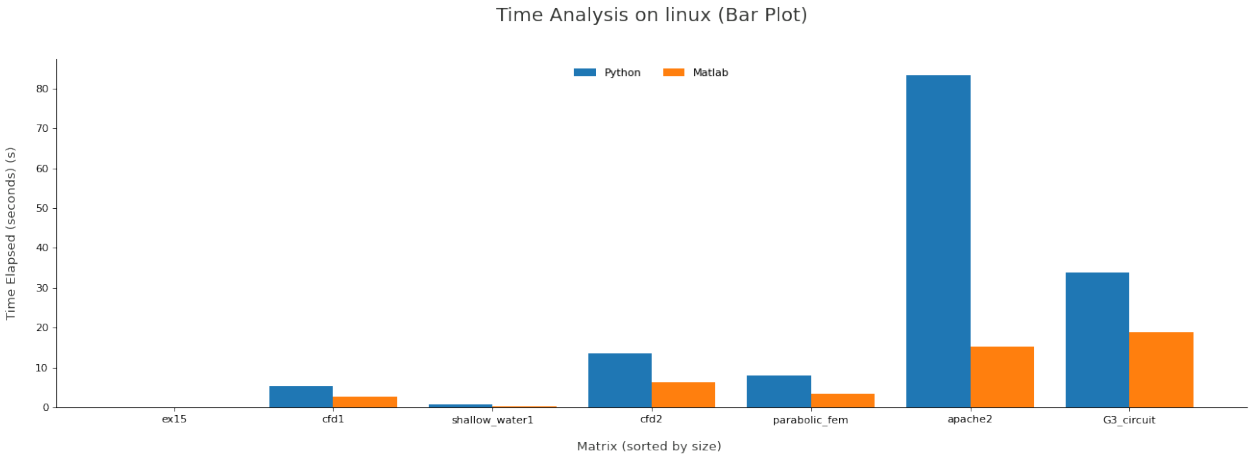


Figura 39: Bar plot tempo dei linguaggi in ambiente linux

5.2.3 Resoconto e Considerazioni Finali

Errore Dalle analisi precedentemente mostrate si può notare come Matlab offra un errore generalmente minore. Si può anche notare che questo è indipendente dal sistema operativo su cui vengono eseguite le operazioni bensì dal linguaggio stesso.

Memoria Bisogna sottolineare che le analisi sulla memoria occupata possono essere migliorate. Sfortunatamente Matlab non possiede un metodo diretto ed efficiente per tenere traccia dei blocchi di memoria occupata dalle singole variabili e dalle singole funzioni sfruttate. Allo stesso tempo non si ritiene corretto effettuare questa tipologia di analisi mediante linguaggi diversi che operano in condizioni differenti e che sfruttano funzioni e metodi che analizzano blocchi di memoria in modo eterogeneo. Dai grafici precedenti si può facilmente notare come Python generalmente occupi molta meno memoria rispetto a Matlab. Al contrario delle analisi di memoria della sezione precedente, si vuole sottolineare che in questo caso si è notata un'indipendenza rispetto al sistema operativo su cui vengono effettuati i test.

Tempo Dalle analisi fornite si può facilmente notare come che tendenzialmente Matlab offre tempi più ragionevoli rispetto a Python.

5.3 Confronto Soluzioni in MATLAB: chol() e backslash

5.3.1 Risultati sperimentali in ambiente Windows

Name	Rows	Nnz	Error	Memory	Time	Language	OS
G3_circuit	1585478	7660826	3.576900e-12	133.42310	8.253900	0	0
apache2	715176	4817870	4.388800e-11	23.16700	5.512300	0	0
cfd1	70656	1825580	1.042100e-13	2.84670	0.890600	0	0
cfd2	123440	3085406	3.687300e-13	4.41960	2.025700	0	0
ex15	6867	98671	6.450600e-07	0.65536	0.012636	0	0
parabolic_fem	525825	3674625	1.049900e-12	13.37750	1.626700	0	0
shallow_water1	81920	327680	2.377800e-16	1.05270	0.158500	0	0

Tabella 15: Backslash on Windows Dataset

Analisi Errore L'errore relativo ottenuto nelle due soluzioni è quasi lo stesso, con un resto impercettibile graficamente: le differenze sono infatti minime, ma sorprendentemente è la soluzione di *chol()* a restituire il risultato migliore, restituendo in media 1.46^{-9} in meno di errore.
Seppur infima, tale differenza rappresenta un risultato interessante che lascia pensare che la decomposizione di Cholesky nel *backslah* di MATLAB potrebbe essere implementata in maniera leggermente diversa.

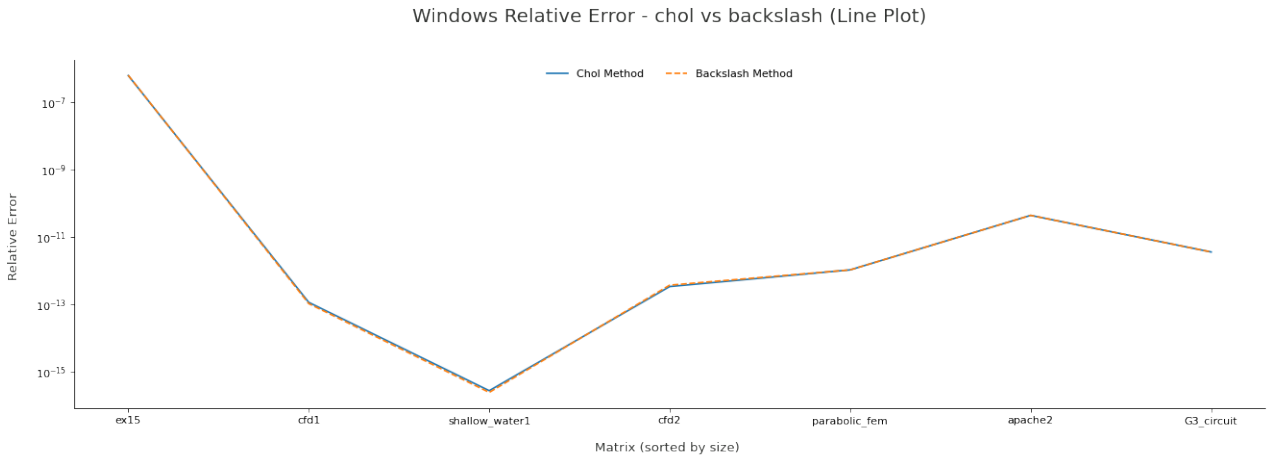


Figura 40: Line Plot errore delle due soluzioni in ambiente windows

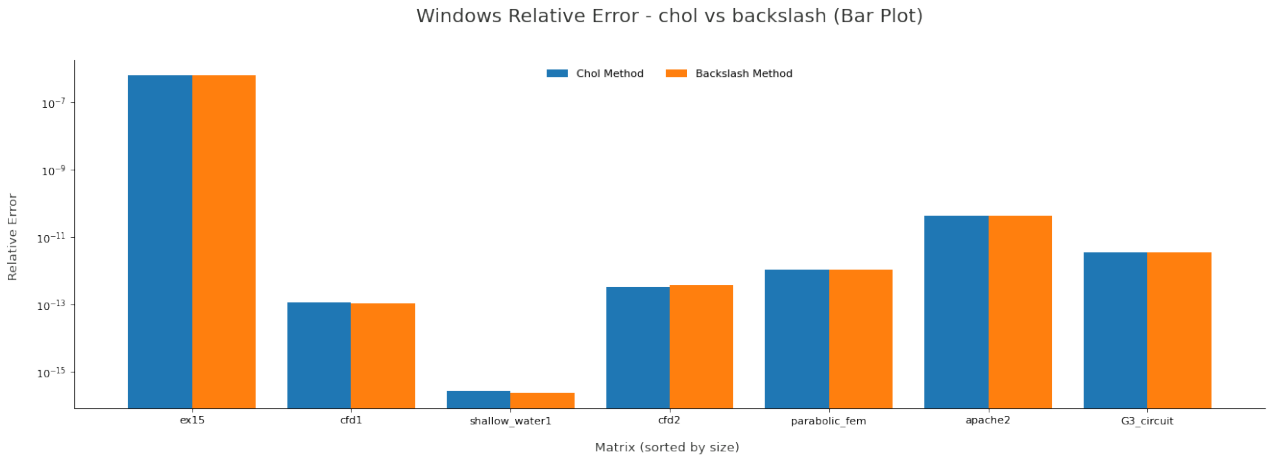


Figura 41: Bar Plot errore delle due soluzioni in ambiente windows

Analisi Memoria Sull'utilizzo della memoria ci sono differenze molto rilevanti, con risultati uguali solo per le matrici *ex15* e *shallow_water1*, mentre per le altre la soluzione con *backslash* richiede molta meno memoria, con una media di 1196 MB in meno.
Questa differenza può essere attribuita ad una maggiore ottimizzazione con eventuali politiche di recycle bin attuate da MATLAB nel calcolo della soluzione in *backslash*, che evitano di mantenere dati non necessari in memoria. Nella soluzione con *chol()*, infatti, vengono mantenute in memoria le matrici di decomposizione e permutazione, il che va a gravare molto sul calcolo della differenza utilizzata prima e dopo la computazione della soluzione.

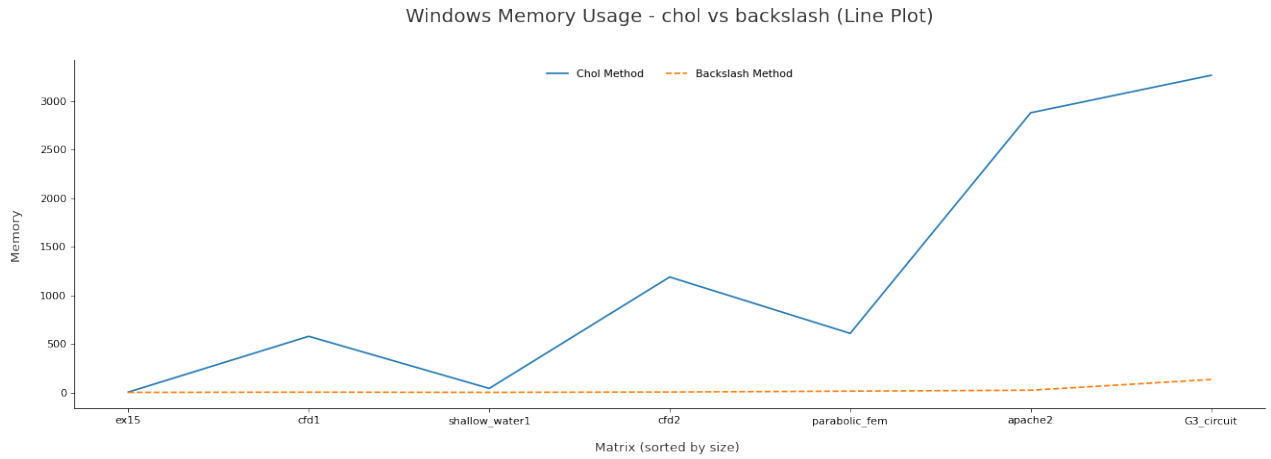


Figura 42: Line Plot memoria delle due soluzioni in ambiente windows

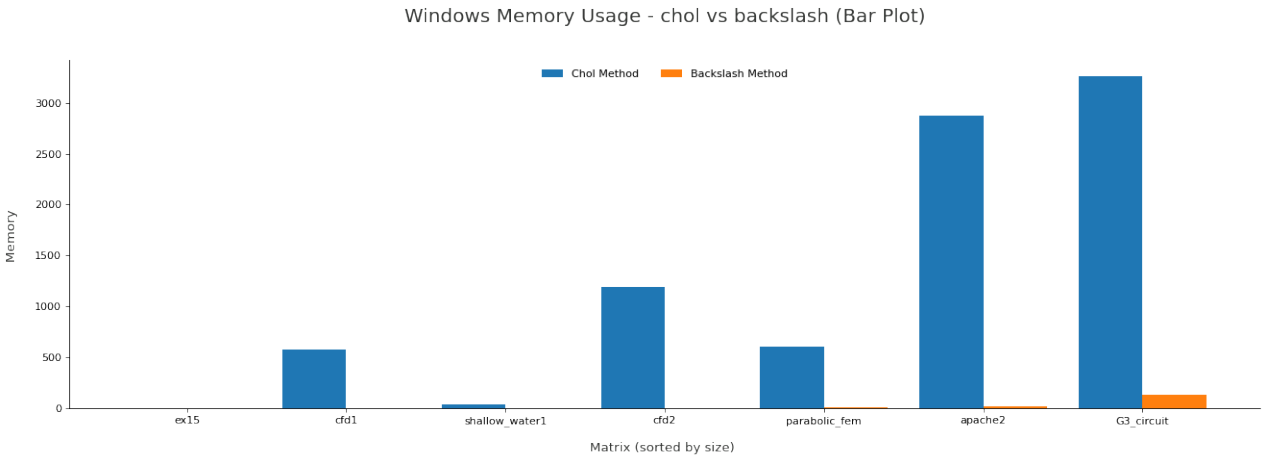


Figura 43: Bar Plot memoria delle due soluzioni in ambiente windows

Analisi Tempo Anche nel tempo di esecuzione la soluzione con *backslash* esce vincente dal paragone, con una media di 3.7 secondi in meno rispetto a *chol()*, che già era di per sè migliore della soluzione in Python. Le differenze maggiori si rilevano nelle ultime due matrici, per cui *backslash* impiega rispettivamente 8 e 11 secondi in meno.

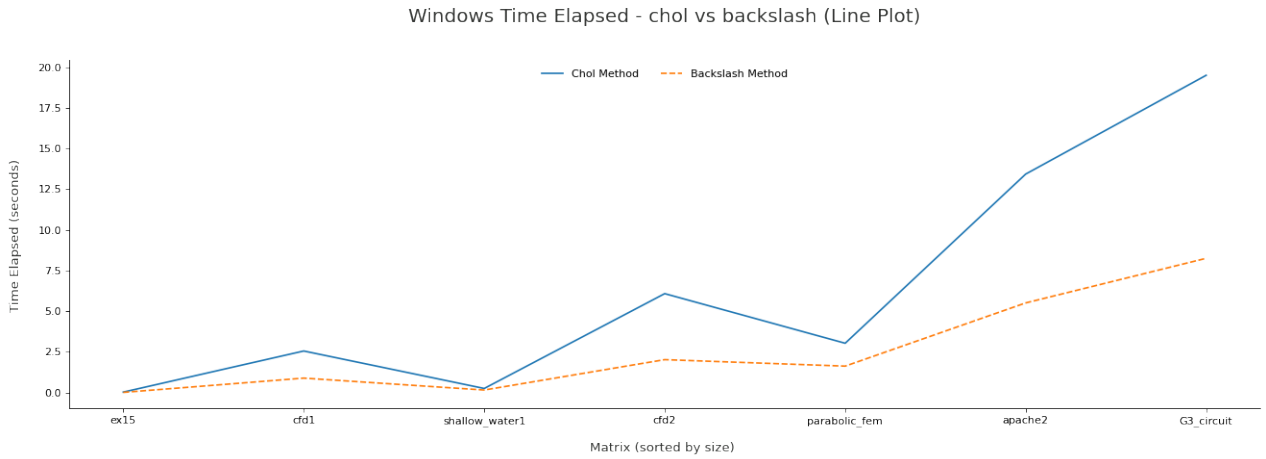


Figura 44: Line Plot tempo delle due soluzioni in ambiente windows

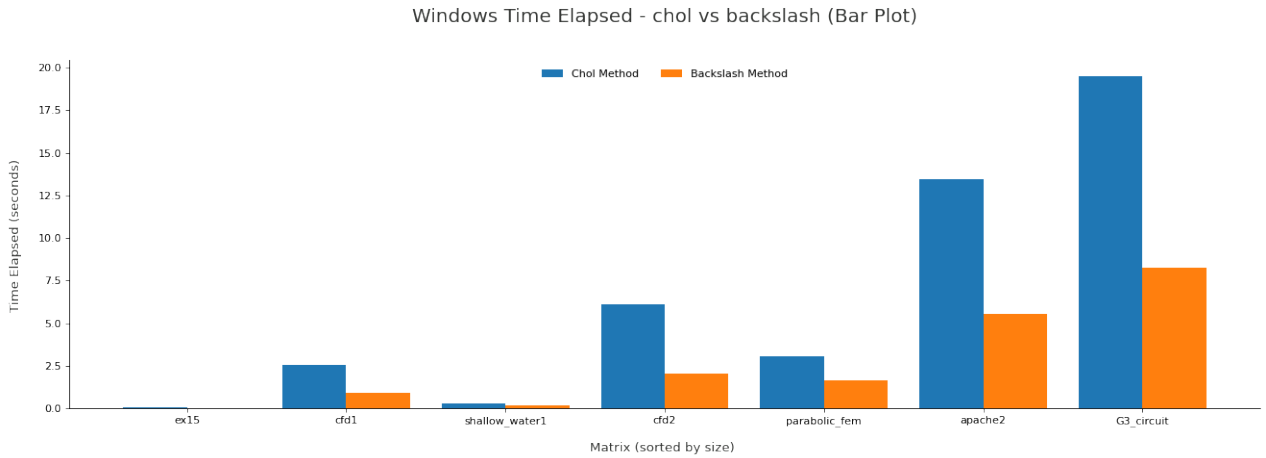


Figura 45: Bar Plot tempo delle due soluzioni in ambiente windows

5.3.2 Risultati sperimentali in ambiente Linux

Name	Rows	Nnz	Error	Memory	Time	Language	OS
G3_circuit	1585478	7660826	3.576900e-12	289.980	8.799500	0	1
apache2	715176	4817870	4.388800e-11	0.000	5.913800	0	1
cfd1	70656	1825580	1.042100e-13	29.064	0.947520	0	1
cfd2	123440	3085406	3.687300e-13	19.140	2.200200	0	1
ex15	6867	98671	6.450600e-07	0.000	0.013287	0	1
parabolic_fem	525825	3674625	1.049900e-12	-48.212	1.766000	0	1
shallow_water1	81920	327680	2.377800e-16	0.000	0.155340	0	1

Tabella 16: Backslash on Linux Dataset

Analisi Errore Anche in questo caso il sistema operativo non causa differenze sull'errore relativo risultante dalla risoluzione del sistema, c'è quindi un leggero vantaggio in favore dell'implementazione che fa uso di *chol()*, uguale a quello in Windows.

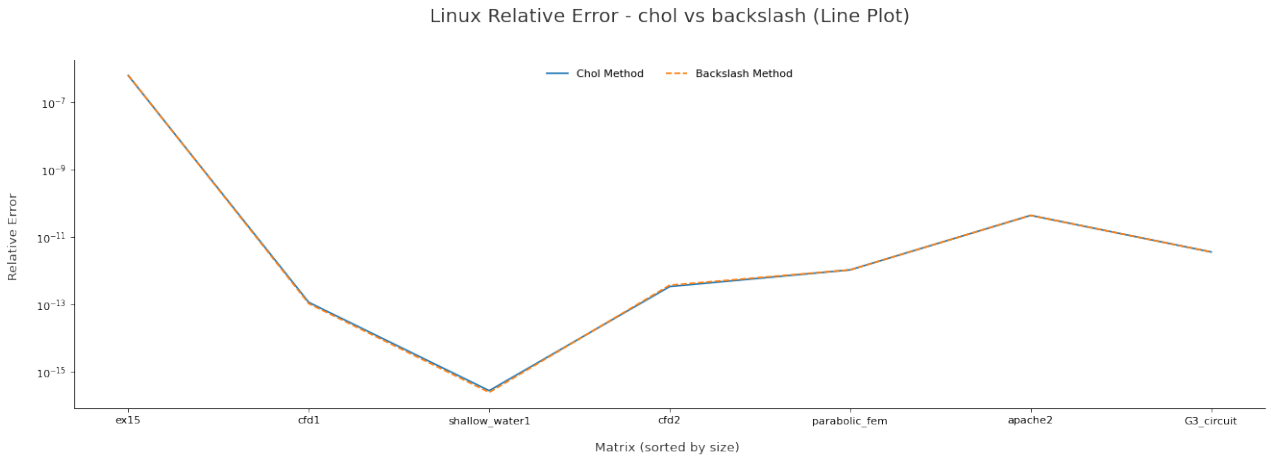


Figura 46: Line Plot errore delle due soluzioni in ambiente linux

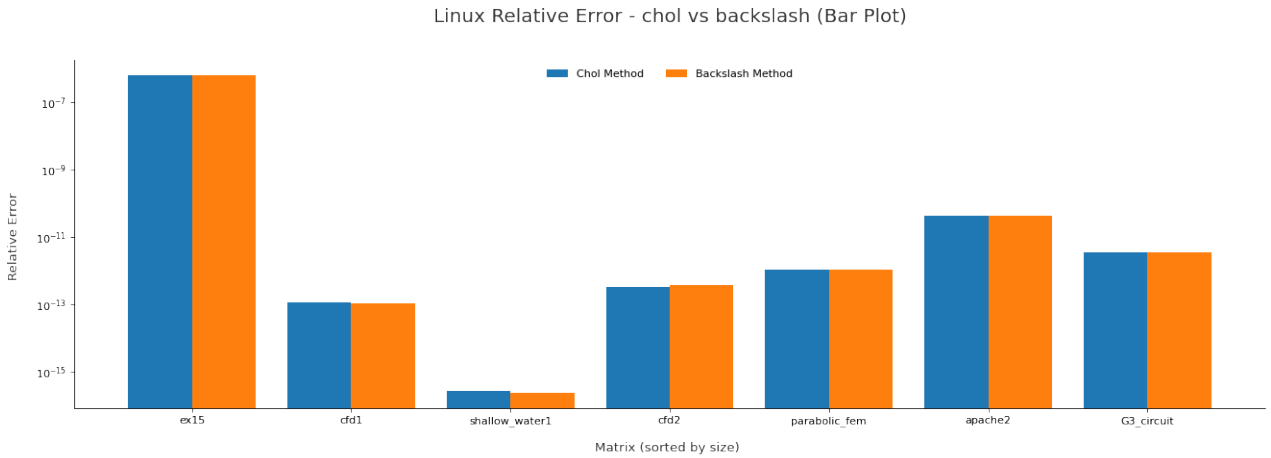


Figura 47: Bar Plot errore delle due soluzioni in ambiente linux

Analisi Memoria Anche in Linux la soluzione di *backslash* occupa decisamente meno memoria, con una media in questo caso di 1132 MB, poco minore di quella in Windows (1196), ma comunque rilevante.

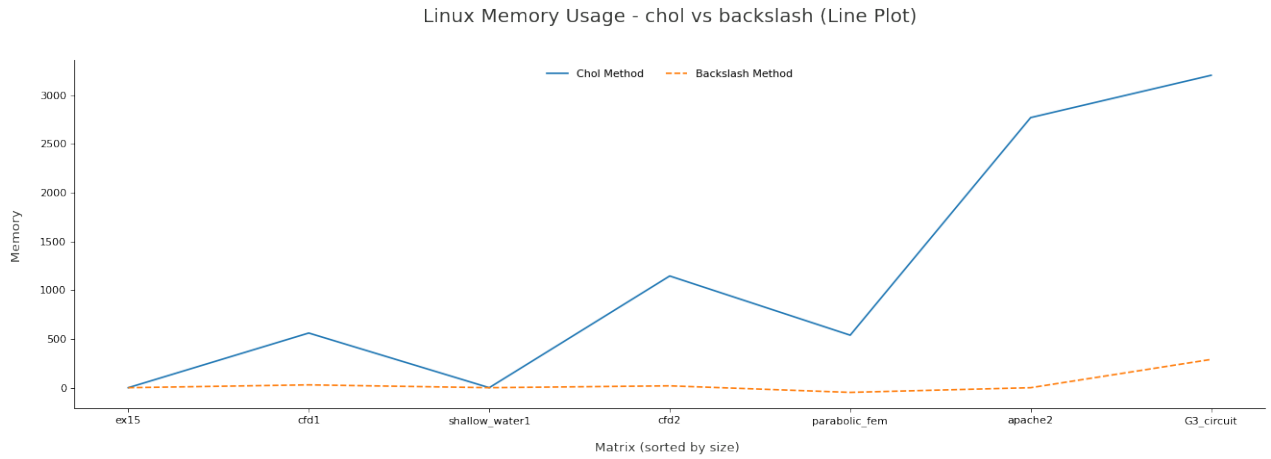


Figura 48: Line Plot memoria delle due soluzioni in ambiente linux

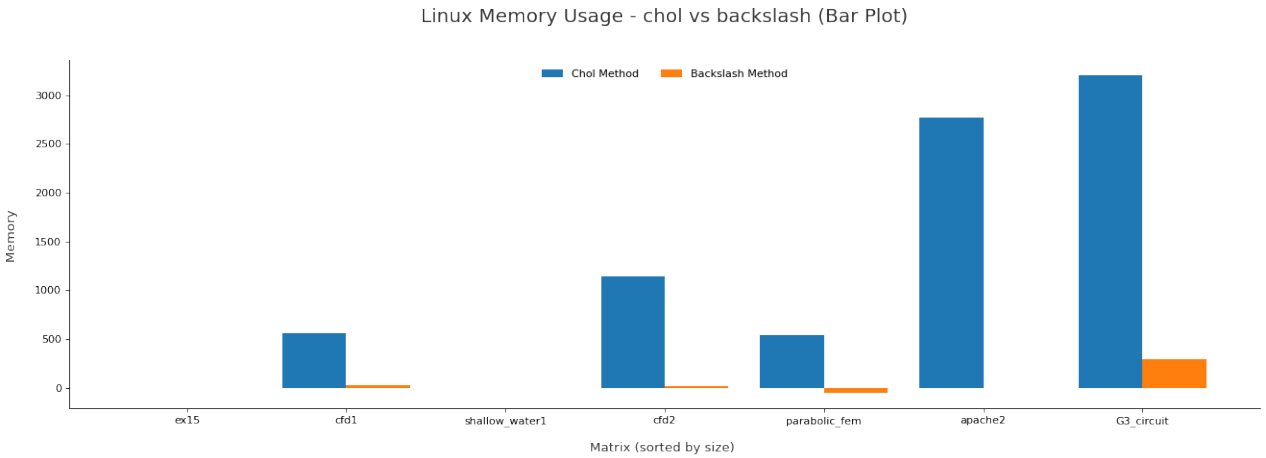


Figura 49: Bar Plot memoria delle due soluzioni in ambiente linux

Analisi Tempo Anche in Linux il tempo di esecuzione è minore con *backslash*, con una media di 3.8 secondi in meno rispetto a *chol()*.
Le differenze maggiori si rilevano nelle ultime due matrici, per cui *backslash* impiega rispettivamente 9 e 10 secondi in meno.

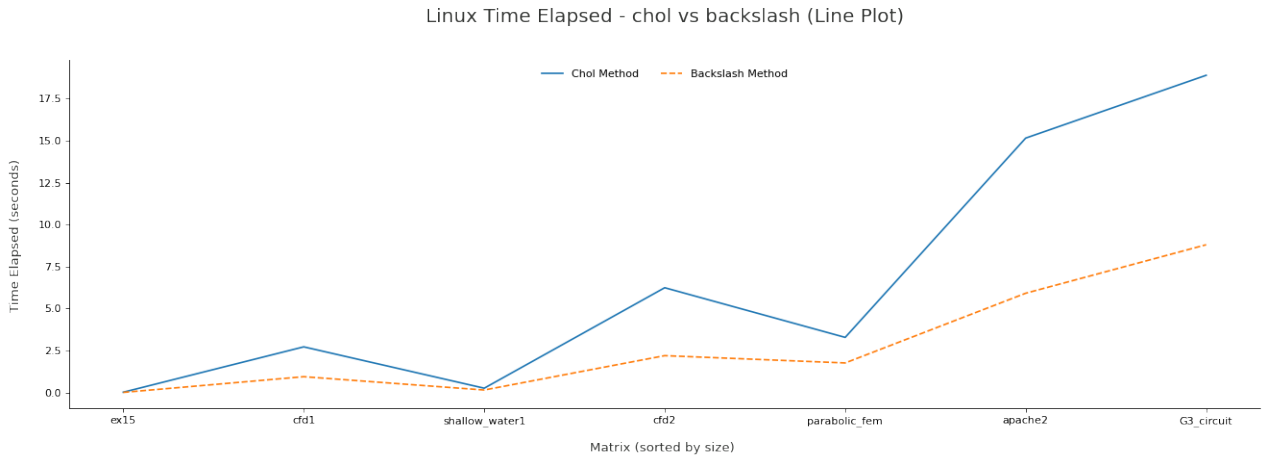


Figura 50: Line Plot tempo delle due soluzioni in ambiente linux

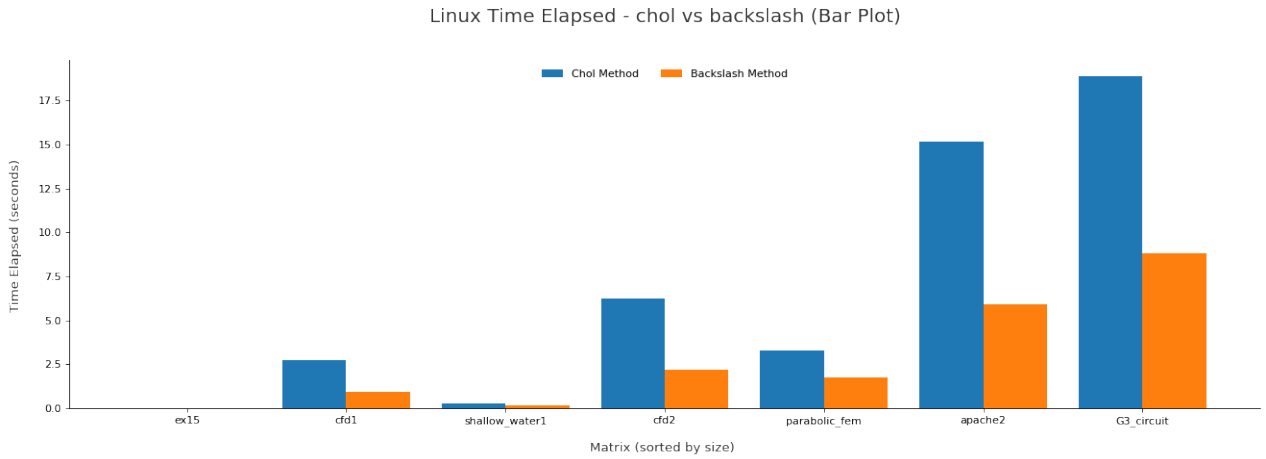


Figura 51: Bar Plot tempo delle due soluzioni in ambiente linux

5.3.3 Resoconto e Considerazioni Finali

Errore Dalle analisi fornite precedentemente si può constatare che il metodo *chol()* offra risultati leggermente migliori rispetto al *BackSlash*. Non è stata riscontrata alcuna differenza al variare del sistema operativo utilizzato.

Memoria Indipendentemente dal sistema operativo utilizzato si è potuto notare come l'esecuzione del *BackSlash* sia estremamente più performante rispetto al concorrente. Questo risultato può essere giustificato, come descritto precedentemente, dalle politiche di *recycle bin* attuate da MATLAB.

Tempo Anche in questo caso si è potuto notare dall'analisi come la soluzione *BackSlash* offrisse soluzioni più immediate rispetto a *chol()*, indipendentemente dal sistema operativo utilizzato.

Le notevoli differenze tra le due metodologie possono portare a pensare che la struttura delle relative funzioni sia estremamente differente.

5.4 Altri Risultati Sperimentali

Aggregazione dei dati A seguito delle analisi, con le metriche risultanti, è stata fatta un'aggregazione dei dati precedentemente raccolti, ottenendo il dataset in tabella 17:

Name	Rows	errorMat	errorPy	memoryMat	memoryPy	timeMat	timePy	Nnz	Cond
apache2	715176	4.388900e-11	2.235941e-08	2823.50505	65.695641	14.288000	74.802496	4817870	5.316861e+06
cfd1	70656	1.135900e-13	6.628463e-12	569.08860	24.734949	2.640650	4.922319	1825580	1.335081e+06
cfd2	123440	3.348600e-13	1.334161e-10	1166.58845	0.003326	6.162600	12.200138	3085406	3.728473e+06
ex15	6867	6.348200e-07	5.281170e-05	1.80840	1.333975	0.027761	0.018617	98671	1.432642e+13
G3_circuit	1585478	3.576600e-12	3.452568e-09	3233.09475	105.311943	19.193450	30.874976	7660826	2.238425e+07
parabolic_fem	525825	1.050000e-12	8.836779e-10	573.02630	0.003326	3.160100	7.180180	3674625	2.110820e+05
shallow_water1	81920	2.672800e-16	6.923992e-14	20.78310	4.535420	0.258710	0.680383	327680	3.628000e+00

Tabella 17: Aggregation Dataset

Ogni entry del dataset rappresenta una delle 7 matrici, i cui dati, sia strutturali che di esecuzione sono aggregati ed espressi dalle colonne.
A causa delle differenze riscontrate nelle analisi tra i due linguaggi di programmazione, per ogni matrice le 3 metriche analizzate sono espresse da 6 attributi: rispettivamente 3 per MATLAB e 3 per Python, ottenute dalla media aritmetica delle metrice ottenute da quel linguaggio sui due sitemi operativi (valori in ogni caso molto simili tra loro).

Correlazione tra Errore relativo e Condizionamento delle Matrici Un'analisi fondamentale è quella che riguarda l'interpretazione dell'errore relativo ottenuto nella risoluzione dei sitemi lineari. Dalle precedenti analisi si nota come l'errore non cresca direttamente al crescere della dimensione delle matrici, pertanto si è deciso di considerare il coefficiente di condizionamento della matrice che misura la sensibilità della soluzione alla perturbazione dei dati in input e in relazione alle approssimazioni effettuate durante la risoluzione a causa dell'errore di macchina.

Il grafico in figura 52, sulle cui ascisse sono poste le matrici ordinate in base al condizionamento in ordine crescente, mostra già come la crescita dell'errore è direttamente proporzionale alla crescita del condizionamento della matrice: per questo *ex15*, pur essendo la matrice più piccola, è quella con errore relativo più grande. L'errore considerato qui è quello di MATLAB, dal momento che il coefficiente di condizionamento è stato calcolato in esso, ma anche per Python il grafico assume un comportamento simile.

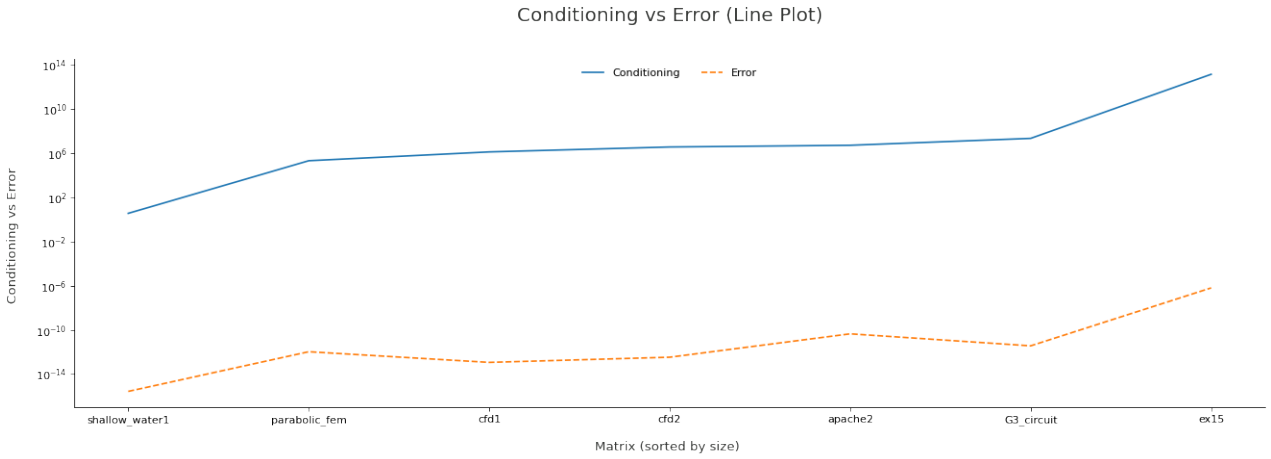


Figura 52: Line plot della correlazione tra Condizionamento ed Errore Relativo

Per un'analisi più precisa e dettagliata sulla correlazione è stato poi calcolato il coefficiente di correlazione, integrato con una regressione lineare per mostrare l'andamento dei dati sul grafico. Dalle analisi precedenti si vede come variare i sistemi operativi non causa alcuna differenza nel calcolo degli errori, pertanto i due grafici che seguono rappresentano il coefficiente per MATLAB e Python. Dalle figrue 53 e 54 si nota come le due metriche siano completamente correlate, con il cofficiente di correlazione che tende a 1 sia in MATLAB che Python.

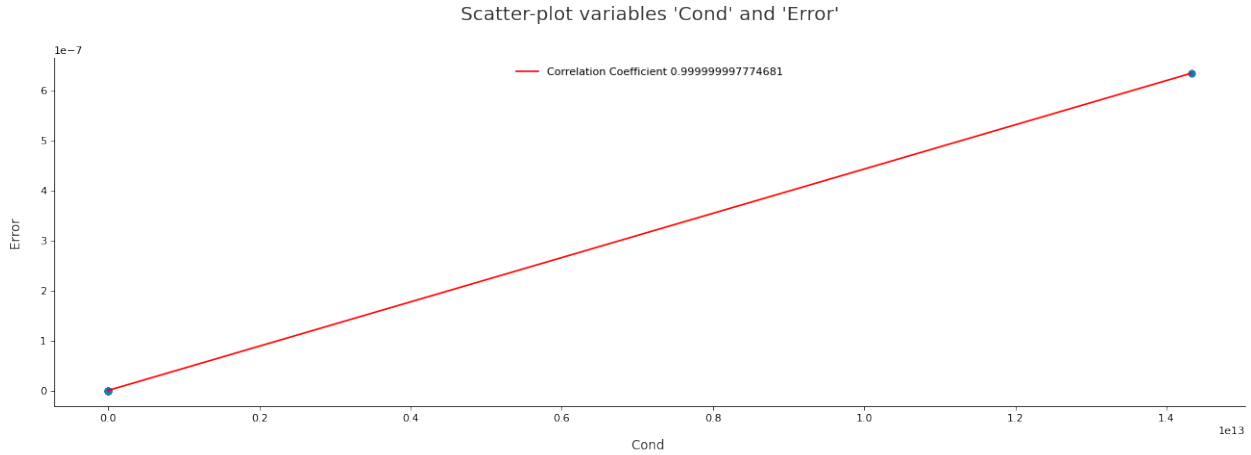


Figura 53: Coefficiente di correlazione tra Condizionamento ed Errore Relativo in MATLAB, con regressione lineare

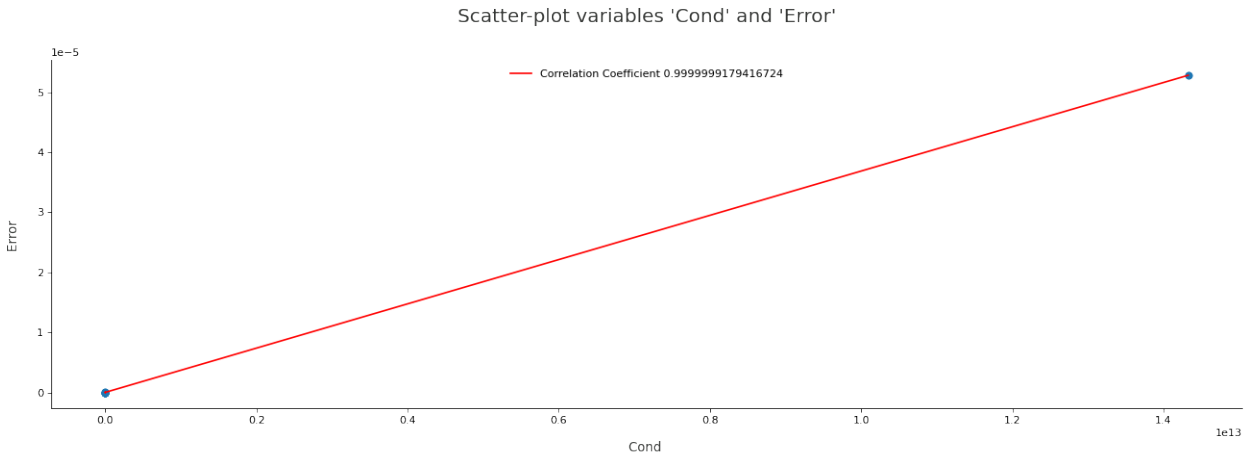


Figura 54: Coefficiente di correlazione tra Condizionamento ed Errore Relativo in Python, con regressione lineare

Questa analisi risulta determinante per l'interpretazione della metrica dell'errore relativo ottenuta della risoluzione dei sistemi lineari.

Correlazione tra Densità delle Matrici e Tempo di esecuzione È stato inoltre analizzato quanto la densità di una matrice, data dal rapporto tra non-zeri ed elementi totali, potesse influire sul tempo di calcolo del sistema associato. Come si nota dai line plot delle figure 55 e 57, e successivamente anche dal cofficiente di correlazione delle figure 56 e 58, che esprime una correlazione negativa, non c'è un legame tra le due misure. Si può notare difatti che la matrice più densa (*shallow_water1*) è anche quella che impiega meno tempo.

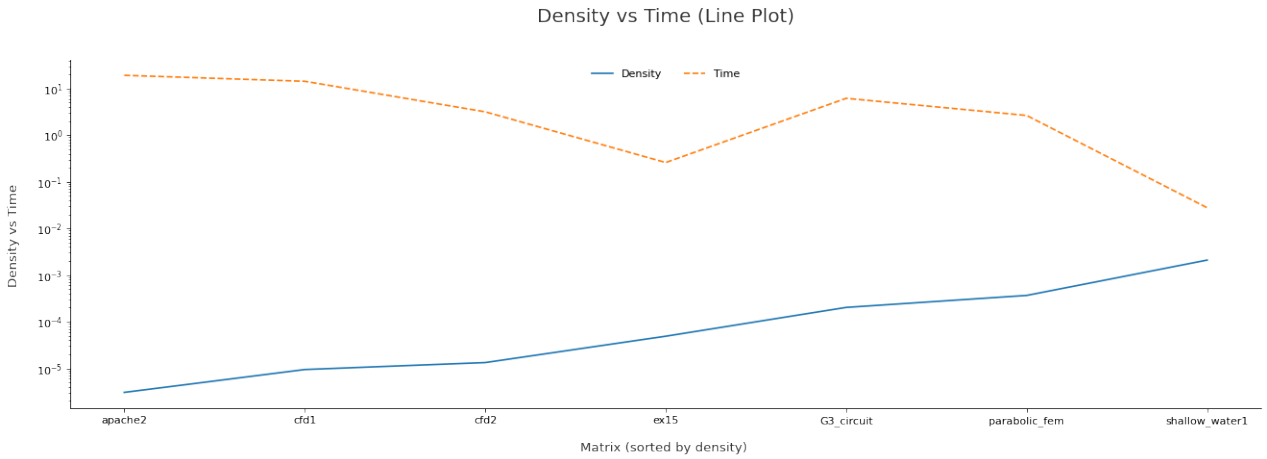


Figura 55: Line plot della correlazione tra Densità della Matrice e Tempo di Esecuzione in MATLAB

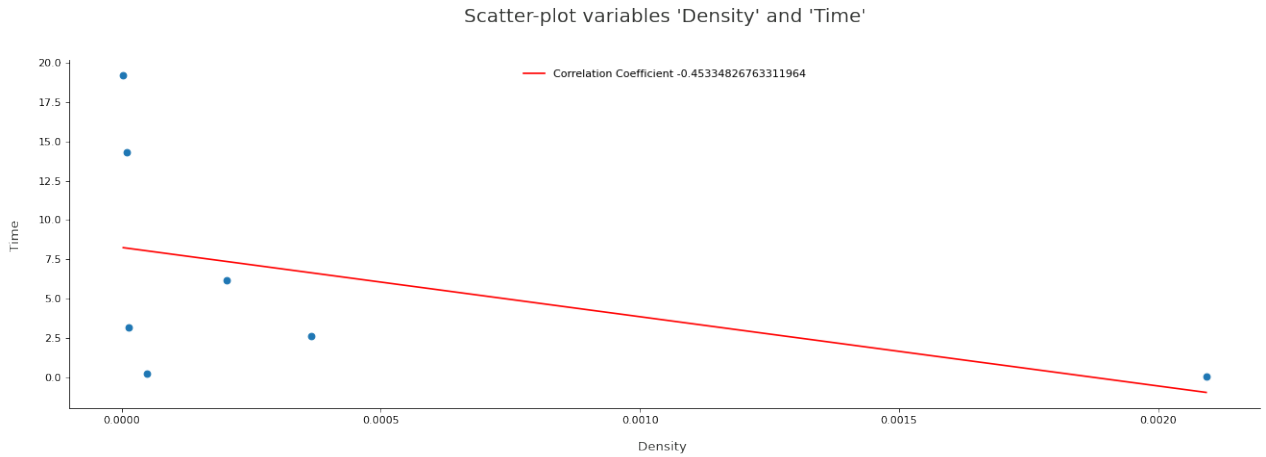


Figura 56: Cofficiente di correlazione tra Densità della Matrice e Tempo di Esecuzione in MATLAB

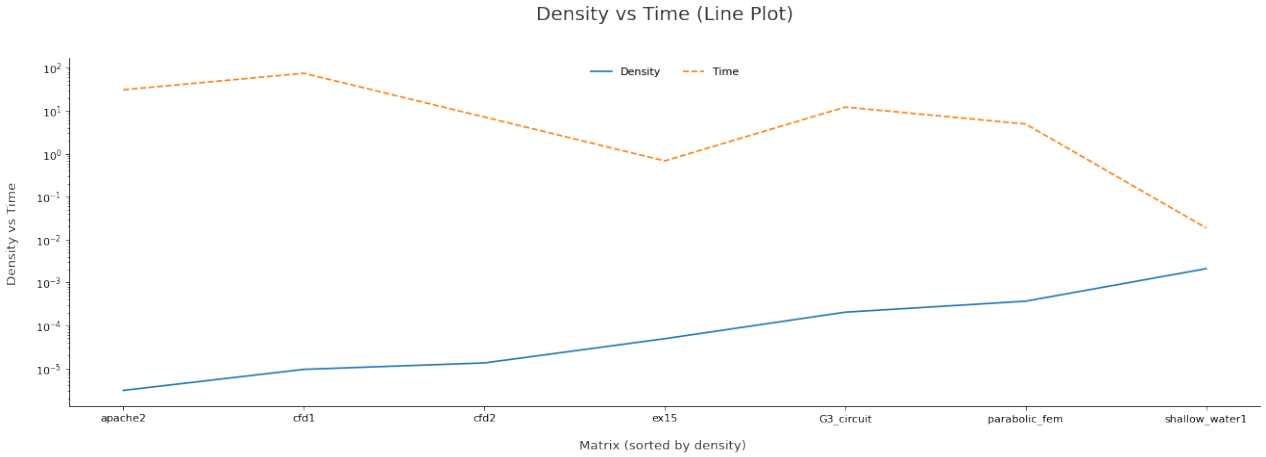


Figura 57: Line plot della correlazione tra Densità della Matrice e Tempo di Esecuzione in Python

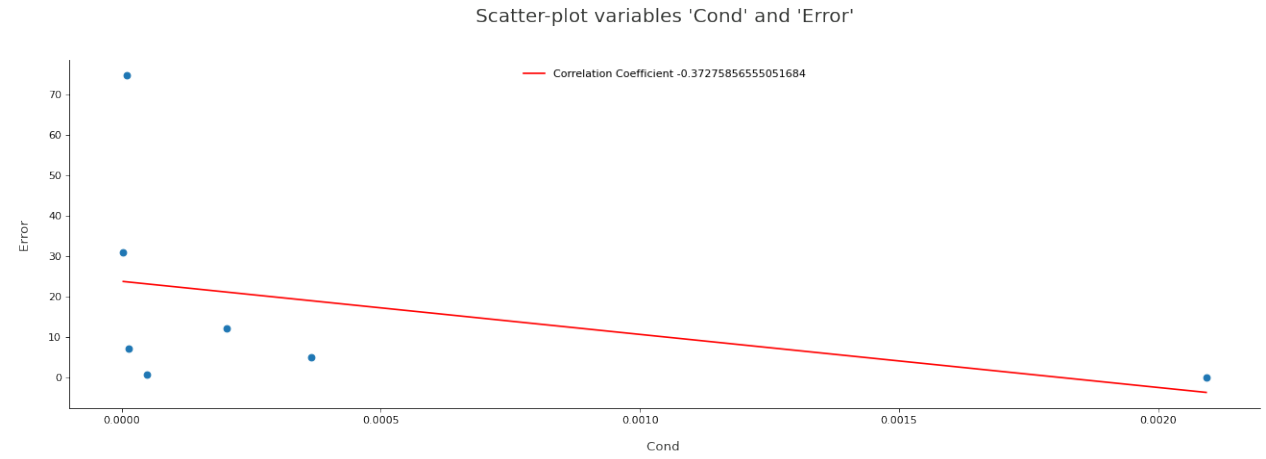


Figura 58: Cofficiente di correlazione tra Densità della Matrice e Tempo di Esecuzione in Python

Resoconto e Considerazioni Finali Dall'aggregazione dei dati su ogni matrice ne sono emerse alcune analisi interessanti e doverose. In particolar modo si sono notate alcune peculiarità:

1. Il calcolo dell'errore di una soluzione è direttamente proporzionale al condizionamento della matrice stessa.
2. Non esiste un vero legame tra il tempo di esecuzione e la densità di una matrice.

6 Conclusioni

Questa analisi ha avuto l'obiettivo di analizzare il comportamento del *Metodo di Cholesky* su diversi sistemi operativi e tramite l'ausilio di diversi linguaggi di programmazione.

Sistemi Operativi Dalle analisi svolte è emersa una grande omogeneità tra i sistemi operativi analizzati. Più nello specifico si vuole sottolineare che i dati riportano un leggero decremento dei tempi in ambiente Windows rispetto a Linux, sebbene questa discrepanza potrebbe essere dovuta all'utilizzo della virtualizzazione tramite Windows Hypervisor. Sebbene Windows sia risultato leggermente più performante in queste analisi è bene ricordare che si sono registrate non poche avversità nell'andare a compilare alcune librerie come *scikit-sparse* o nell'andare ad installare alcuni programmi. Per questo motivo si vuole anche precisare che i dati a disposizione, per eseguire una scelta di preferenza su un Sistema Operativo, siano attualmente pochi. Si vogliono quindi invitare le ricerche future a considerare anche altri parametri per eseguire un'analisi più precisa e dettagliata in questo contesto, ad esempio si potrebbe considerare anche il supporto di un sistema operativo all'utilizzo di una determinata libreria o ad un determinato linguaggio.

Linguaggi In termini di linguaggi è necessario affermare che generalmente Matlab offra prestazioni decisamente migliori per via del calcolo dell'errore molto più preciso e dei tempi di esecuzione decisamente ridotti rispetto al concorrente. Dal resoconto finale si è voluto spontanemente evitare di considerare l'indice di memoria occupata, in cui il divario tra i due linguaggi è estremamente marcato a favore di Python, poichè dalle analisi riportate sono emerse problematiche inerenti il corrispettivo monitoraggio. Dagli ulteriori studi che sono stati forniti è anche emerso che MATLAB fornisce effettivamente metodi differenti per andare a risolvere un sistema lineare, più in particolare mediante il *Metodo di Cholesky*. E' stato analizzato il metodo *chol()*, che fornisce un modo specifico per il calcolo del solo metodo, e il metodo tramite *BackSlash* che è molto più generale e flessibile. Dalle analisi è emerso che quest'ultimo metodo, non solo è adatto a più tipologie di risoluzione di sistemi lineari, ma è anche molto più efficiente di *chol()*. Per questa ragione si può afffermare che MATLAB sia estremamente più efficiente rispetto a Python in questo ambito.
Nonostante ciò si vogliono comunque fornire degli spunti di riflessione al lettore, soprattutto inerenti al linguaggio da poter scegliere per questa tipologia di analisi:

- MATLAB nasce come una **piattaforma di programmazione e calcolo numerico** ideata soprattutto per l'analisi di dati, lo sviluppo di algoritmi e la creazione di modelli.
- Python è un **linguaggio di programmazione** di "alto livello", orientato a oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.

Seguendo le definizioni sopra riportate è essenziale ricordare che le funzionalità di Python non sono primariamente ricorrenti in questa tipologia di operazioni, ed è per questo motivo che non si vuole screditare Python in questa analisi. Osservando attentamente i dati riportati si può comunque constatare che le performance di quest'ultimo non siano affatto scadenti. Per questo motivo si potrebbe comunque consigliare Python come alternativa Free e Open-Source per analisi di piccole-media entità o qualora non si abbiano le corrispettive risorse per sfruttare una licenza Matlab. Per concludere si vogliono invitare i futuri studi a comprendere meglio le implicazioni dei risultati ottenuti magari concentrarsi meglio sulle limitazioni citate precedentemente.

Riferimenti bibliografici

Wikipedia. Hypervisor — wikipedia, l'enciclopedia libera, 2021. URL http://it.wikipedia.org/w/index.php?title=Hypervisor&oldid=120906467. [Online; in data 15-aprile-2022].

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi:10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

Fernando Perez, Brian E Granger, and John D Hunter. Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21, 2011.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.

The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi:10.25080/Majora-92bf1922-00a.