

## **Relazione del progetto 1 di Metodi del Calcolo Scientifico**

**Studenti:** Daniel Scalena - 844608 - [d.scalena@campus.unimib.it](mailto:d.scalena@campus.unimib.it)

**Corso:** Metodi del calcolo scientifico

### **Contents**

<b>1 Overview</b>	<b>3</b>
1.1 Problema e progetto . . . . .	3
1.2 Menzioni tecniche informatiche . . . . .	3
1.3 Aspetti matematici del problema affrontato . . . . .	4
<b>2 Strumenti Open Source selezionati</b>	<b>5</b>
2.1 Math Kernel Library, basso livello . . . . .	5
2.2 Linguaggio di programmazione e librerie, alto livello . . . . .	5
<b>3 Implementazione</b>	<b>8</b>
3.1 Matlab . . . . .	8
3.2 Libreria Open Source (w/ Python) . . . . .	9
<b>4 Analisi dei risultati</b>	<b>11</b>
4.1 Premesse sui risultati ottenuti . . . . .	11
4.2 Risultati . . . . .	11

<b>5 Considerazioni conclusive</b>	<b>24</b>
5.1 Grafici riepilogativi . . . . .	24
5.2 Sviluppi futuri . . . . .	25
5.3 Conclusioni . . . . .	29

# 1 Overview

## 1.1 Problema e progetto

Lo scopo del progetto è quello di effettuare un confronto tra due ambienti diversi per la risoluzione di sistemi lineari. Uno di questi ambienti, MATLAB, è scelto a priori, rappresentate lo standard de facto nel calcolo scientifico per ambienti relativi alla matematica. Il secondo ambiente invece dovrà essere scelto con l'unico vincolo di essere open source, ovvero gratuito e con un codice pubblico in grado di essere letto liberamente online.

Il confronto non riguarda solamente la comparazione dei tempi di risoluzione del sistema lineare ma anche dello spazio in memoria utilizzato, del tempo di caricamento della matrice, dell'accuratezza delle soluzioni nonché dei tempi di caricamento della matrice stessa viste le grandi dimensioni che può assumere. Viene quindi assunto il medesimo ambiente hardware indipendentemente dal software utilizzato, assicurando la fedeltà dei dati riportati a meno di problemi poco controllabili dovute a questioni relative al clock massimo e alla temperatura ambientale, comunque affrontate nei capitoli successivi.

Per ultimo vengono ulteriormente messi a confronto due sistemi operativi: Windows, closed source, e Linux, open source. In questo caso si vuole valutare come questa variabile possa impattare sulle performance generali dei vari programmi.

## 1.2 Menzioni tecniche informatiche

Viene presentata in seguito una breve overview del sistema utilizzato per la risoluzione dei sistemi e la misurazione delle performance.

Tutti i test sono stati effettuati su piattaforma x86\_64 ad eccezione di dove diversamente specificato. Il processore usato è un AMD Ryzen 5700X dotato di 8 core e 16 threads con una frequenza minima di 3.4 Ghz e una frequenza boost di 4.6 Ghz. Il quantitativo di RAM a disposizione, utile per mantenere in memoria la matrice, è pari a 32 GB di tipo DDR4 a 3600 Mhz. Le matrici infine sono state memorizzate su un SSD NVME di quarta generazione, in grado di assicurare le migliori performance in lettura e scrittura. Relativamente a quest'ultimo sarà utile, durante la computazione, quando si supererà il quantitativo di memoria RAM a disposizione, dovendo archiviare i dati necessari direttamente sul disco.

Le caratteristiche software invece comprendono l'impiego di due sistemi operativi: Windows 11 aggiornato all'ultima versione dei primi di luglio 2022 e una distro linux debian based denominata PopOS, anch'essa aggiornata all'ultima versione sia dei pacchetti che del kernel disponibile nei primi di luglio 2022. Sempre riguardo il software vengono impiegate le stesse release dei pacchetti di Python relativamente alla versione 3.9 e l'ultima versione di Matlab disponibile R2022a con una licenza accademica.

Viene inoltre precisato come per il sistema operativo di Microsoft non viene usato un ambiente WSL visto che in questo caso le chiamate di sistema sarebbero fatte a un kernel linux (come fosse una macchina virtuale ma senza una grande perdita di prestazioni).

### 1.3 Aspetti matematici del problema affrontato

Le matrici considerate saranno a valori reali, simmetriche e definite positive, dovendo affrontare una tipologia di decomposizione LU definita come decomposizione di Cholesky. La decomposizione in questione restituisce due matrici una trasposta dell'altra, la triangolare superiore e inferiore, riuscendo a fornire un vantaggio in termini di efficienza nella risoluzione del sistema. Inoltre, vista il vasto impiego di matrici sparse (ovvero con molti zeri) questo tipo di decomposizione risulta essere estremamente vantaggiosa, se comparata ad esempio con il metodo di eliminazione di Gauss, vista la capacità di ridurre il fenomeno del *fill-in* durante la risoluzione del sistema  $Ax = b$  dove  $A$  è la matrice considerata,  $x$  è il vettore delle incognite e  $b$  sono i termini noti.

Nella pratica l'algoritmo avrà questa struttura basilare:  $A$  viene inizialmente scomposta in  $LU$ , viene risolto  $Ly = b$  con  $y = Ux$  e infine viene risolto  $Ux = y$ , trovando la soluzione in  $x$ .

## 2 Strumenti Open Source selezionati

Viene di seguito dettagliata la scelta e le diverse configurazioni adottate per quanto riguarda l'impiego della libreria open source selezionata. Non viene approfondito invece l'ambiente di MATLAB vista la sua natura chiusa e segreta di progettazione.

### 2.1 Math Kernel Library, basso livello

Partendo dal basso, si è deciso di sperimentare per il progetto una libreria Kernel prodotta da Intel e ottimizzata per le operazioni matematiche o in generale scientifiche, ingegneristiche e finanziarie. Dal 2020 la libreria kernel è disponibile senza lock-in per quanto riguarda la CPU, essendo quindi non più un esclusiva sui processori Intel Core. Il suo funzionamento comprende soprattutto la distribuzione parallela dei calcoli su tutti i core della CPU, ottimizzando al massimo le chiamate di sistema al kernel, più di quanto le librerie ad alto livello quali NumPy e SciPy (prendendo Python ad esempio) possano fare. Online possono essere inoltre trovati indizi su come MATLAB faccia uso di questa libreria open source per ottimizzare la velocità di calcolo su CPU consumer e prosumer.

Attualmente questa libreria risulta essere preinstallata ma non vengono fatte ottimizzazioni manuali lasciando una configurazione standard considerata stabile e ottimale per la maggior parte degli impieghi. Nel corso del progetto si è deciso però di sfruttare al massimo le potenzialità offerte dalla libreria, immaginando che in produzione si utilizzi una macchina destinata puramente al calcolo scientifico, non dovendo richiedere eccessive risorse per quanto riguarda il multitasking tra diverse applicazioni. Per questo la variabile *MKL\_CPU\_Type* è stata impostata con il numero massimo di core disponibili sulla CPU in uso.

### 2.2 Linguaggio di programmazione e librerie, alto livello

La scelta più classica in ambito scientifico e open source in termini di linguaggio di programmazione è sicuramente Python, ormai standard per la sua semplicità e versatilità di utilizzo. Tra le diverse librerie, tutte open source, che python mette a disposizione per la risoluzione di sistemi lineari vi è sicuramente SciKit-Sparse e NumPy, dove sono implementati i metodi Cholesky usati per la risoluzione di sistemi lineari con matrici sparse che godono delle proprietà SPD.

Vengono effettuati dei test per quanto riguarda la scelta tra la libreria NumPy e SciKit-Sparse, cercando quale tra le due risulta essere più efficiente ed adeguata per un confronto con la controparte proprietaria Matlab. In figura 1 vengono visualizzati i tempi di esecuzione per 5 diverse matrici sparse generate randomicamente, ripetendo il test 10 volte. Come evidente i risultati sono sovrapponibili osservando anche la media e la varianza riportate nella stessa figura. Viene ulteriormente analizzato l'andamento all'aumentare della dimensione delle matrici tra le due librerie e, come evidente in figura 2, questo rimane simile per le librerie indipendente dal parametro  $n$  scelto. Viene fatto notare come, sempre nel grafico in figura 2 SciKit-Sparse risulta essere lievemente più lento ma è un risultato dovuto all'ordine di esecuzione dell'algoritmo. Eseguendo prima la computazione con NumPy, questa riscalda la CPU, degradandone leggermente le performance quando sarà compito di SciKit-sparse risolvere i sistemi; se si inverte l'ordine di esecuzione si osserva, come previsto, il risultato inverso a quanto rappresentato in figura dove SciKit è lievemente più veloce di NumPy.

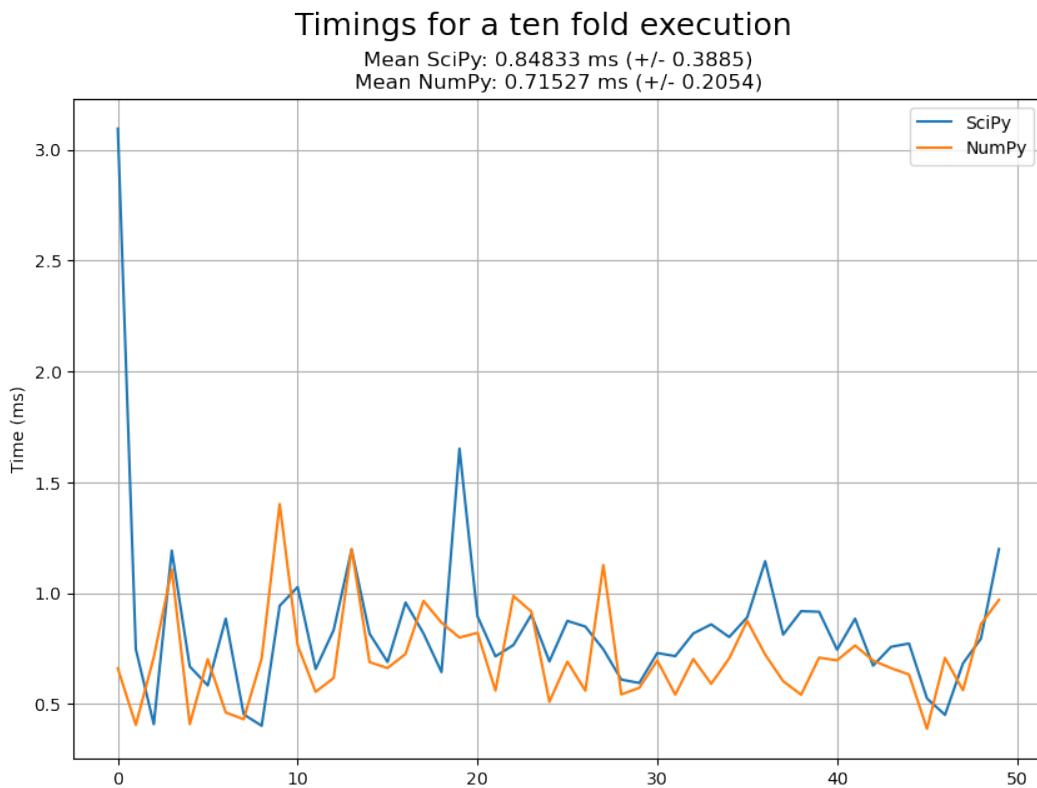


Figure 1: Timings on a random generated matrix for SciKit-sparse and Numpy

Timings for a ten fold execution for each dimension

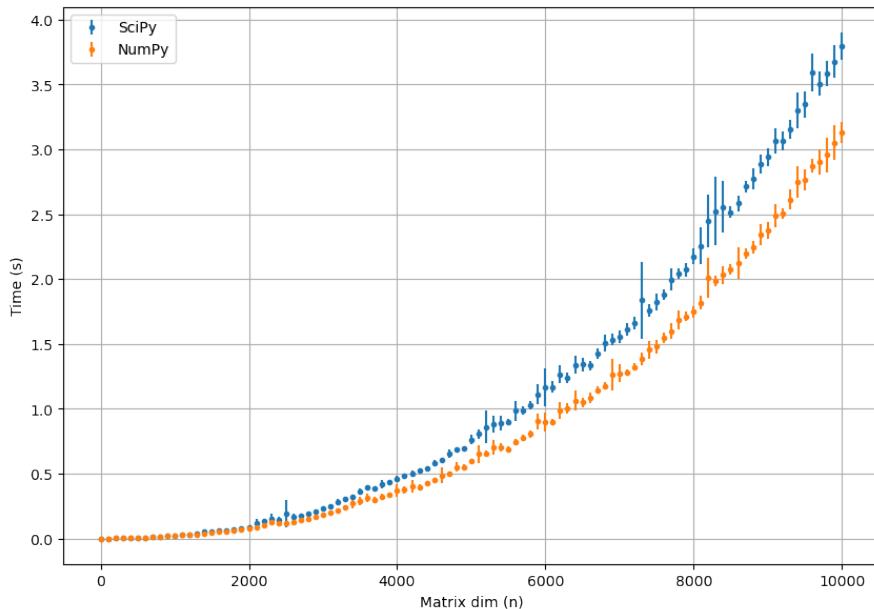


Figure 2: Osservazione dell’andamento delle due librerie all’aumentare delle dimensioni della matrice sparsa. Nota: si è eseguito prima NumPy e successivamente SciKit-sparse ma, invertendo la computazione, si ottiene il risultato opposto dove SciKit-sparse è lievemente più veloce di Numpy

In seguito a questi test si è pertanto deciso di proseguire con SciKit-Sparse, essendo una libreria più specifica rispetto a NumPy, nonostante le prestazioni del tutto simili e soprattutto alla forte condivisione dello stesso codice sorgente tra le due rispettive repository online.

Per la gestione del software vengono impiegate ulteriori librerie per la gestione numerica e statistica (NumPy, SciKit), *matplotlib* per la visualizzazione dei risultati, *time* e *trace malloc* per il monitoraggio dei tempi e della memoria utilizzata e *scikit-learn* per la verifica delle proprietà iniziali delle matrici (simmetria). Queste librerie in ogni caso non influenzano in alcun modo i dati riportati in seguito.

## 3 Implementazione

Di seguito vengono riassunti brevemente i codici relativi all'implementazione delle diverse funzioni per risolvere i sistemi lineari e monitorarne le performance di spazio, di soluzione e tempo sia di lettura che di risoluzione.

È necessario precisare che, per quanto riguarda il monitoraggio della memoria, **Matlab e Python utilizzano metodi differenti che potrebbero riportare risultati non confrontabili con precisione**. Non esistono soluzioni terze per il monitoraggio della memoria vista la presenza di sistemi di sicurezza nei sistemi operativi che isolano l'esecuzione di un programma, non permettendo quindi di leggere ciò che un altro programma conserva nella propria memoria e non potendo quindi distinguere tra memoria allocata ed effettiva memoria realmente in uso.

Fornita questa premessa, da qui in avanti si considerano equivalenti le due soluzioni adottate.

### 3.1 Matlab

Per Matlab, altre a quelle listate, vengono usate ulteriori funzioni per il salvataggio automatico delle informazioni.

```
1 % inizio timer
2 read_t = tic;
3 % carico matrice
4 A = load("./matrix/" + matrix_name);
5 % lap timer
6 read_t = toc(read_t);
7 solve_t = tic;
8 % verifiche e soluzioni
9 A = A.Problem.A;
10 xe = ones(size(A, 1), 1);
11 b = A * xe;
12 % risolvo il sistema
13 x = A \ b;
14 % stop timer
15 solve_t = toc(solve_t);
```

Listing 1: Calcolo del tempo di lettura della matrice e risoluzione del sistema

```
1 A = load("./matrix/" + matrix_name);
2 % accedo al contenuto
3 A = A.Problem.A;
```

```

4     xe = ones(size(A, 1), 1);
5     b = A * xe;
6     % risolvo
7     x = A \ b;
8     % errore basato sul risultato ottenuto
9     relative_error = norm(x - xe, 1) / norm(xe, 1);

```

Listing 2: Calcolo dell'errore relativo

## 3.2 Libreria Open Source (w/ Python)

Nel codice allegato a questo report è presente anche il resto del codice per la gestione dei dati e la creazione dei grafici riportati in seguito.

```

1   try:
2       # Starting timer
3       read_time = time.time()
4       # Reading
5       A = mmread(path + matrix_name)
6       # Lapping timer
7       read_time = time.time() - read_time
8
9       # solving
10      check_symmetric(A, raise_exception = True)
11      xe = np.ones((A.shape[0], 1))
12      b = A * xe
13
14      # stopping timer
15      solve_time = time.time()
16      factor = cholesky(A)
17      x = factor(b)
18      solve_time = time.time() - solve_time
19  except ValueError:
20      print(ValueError)

```

Listing 3: Codice python per la registrazione del tempo di lettura della matrice e di risoluzione del sistema

```

1   try:
2       # Start tracking memory
3       start_track_memory()
4       # Leggo la matrice
5       A = mmread(path + matrix_name)
6       # Check symmetry of the matrix

```

```

7     check_symmetric(A, raise_exception = True)
8
9     # a solution
10    xe = np.ones((A.shape[0], 1))
11    b = A * xe
12
13    # cholesky
14    factor = cholesky(A)
15    x = factor(b)
16
17    # saving memory peak
18    memory_peak = convert_size(end_track_memory())
19    relative_error = np.linalg.norm(x - xe) / np.linalg.norm(
xe)
20    except ValueError:
21        # there is a problem
22        print(ValueError)

```

Listing 4: Codice Python per il monitoraggio della memoria e il calcolo dell'errore relativo

## 4 Analisi dei risultati

### 4.1 Premesse sui risultati ottenuti

Vengono presentati in seguito i risultati ottenuti che, come specificato, comprendono i test su due sistemi operativi diversi (Windows e Linux) e due ambienti diversi (Matlab e Python), rispettivamente per entrambi closed e open source.

Per quanto riguarda i tempi si è deciso di ripetere l'esperimento più di una volta in modo da eliminare per quanto possibile fattori esterni non attinenti ai benchmark effettuati, ottenendo degli intervalli di confidenza da poter considerare in fase comparativa. Nel dettaglio, in quasi tutti i processi di risoluzione del sistema lineare si è generalmente osservato un aumento della temperatura sul core della CPU, cosa che ha portato al verificarsi, seppur in maniera lieve, di *thermal throttling*, ovvero una lieve diminuzione del clock massimo raggiungibile per non alzare eccessivamente le temperature interne. Infine, per evitare che un test precedente influenzasse quello successivo si è aspettato un periodo di 30 secondi tra una matrice e la successiva per permettere il raffreddamento della CPU.

L'ordine dei risultati presentati infine rispetta quanto effettuato sui test, cercando di partire dalla matrice più leggera fino a raggiungere quelle più grandi e difficili da risolvere in un sistema lineare.

### 4.2 Risultati

**ex15** La prima matrice sparsa presentata è denominata *ex15* ed è una matrice SPD con una dimensione pari a (6867, 6867). Ha un numero di condizionamento pari a  $1.4326e+13$ , misurato da Matlab, e presenta 9871 elementi diversi da zero.

Come mostrato in figura 3, nonostante la minima differenza, i tempi di caricamento della matrice sono a vantaggio di Matlab sia su Windows che su Linux. Nei tempi di risoluzione del sistema lineare invece tutte le configurazioni di ambiente e OS si comportano allo stesso modo, eccezion fatta per Matlab su Windows dove è evidente un tempo peggiore, seppur minimo vista la quantità di tempo totale che oscilla tra 0.01 a 0.03 secondi. Analizzando il grafico in figura 4 è invece evidente come per Matlab sia presente un errore relativo minore che per la controparte Open source. È importante sottolineare come **l'errore relativo sia comunque alto congruentemente con l'osservazione di un numero di condizionamento molto alto**, riportato poco sopra. Per la memoria utiliz-

zata invece risulta essere Matlab ad andare in svantaggio consumando quasi il doppio rispetto all'alternativa Open source per entrambi i sistemi operativi.

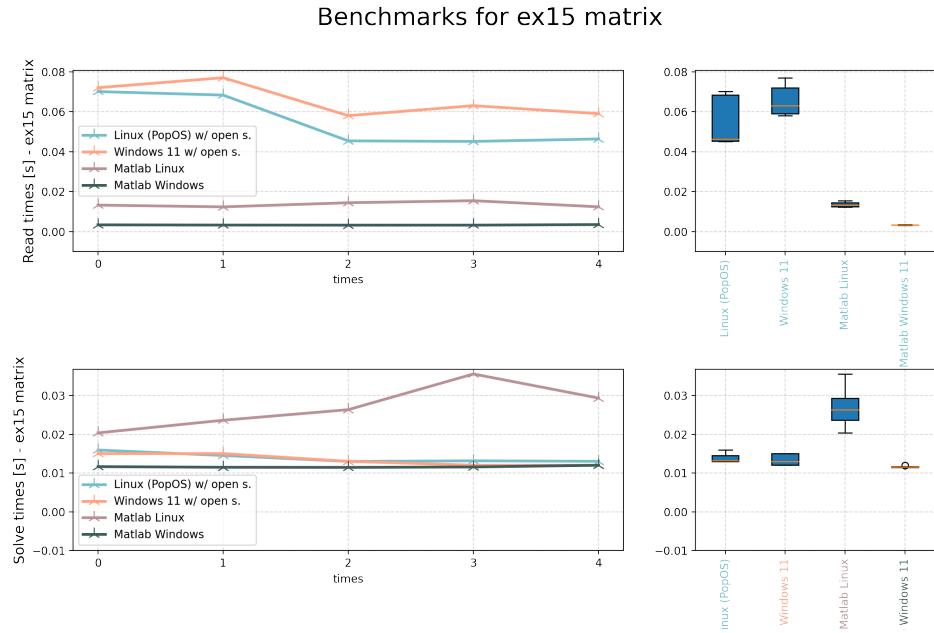


Figure 3: *Ex15* read and solve times

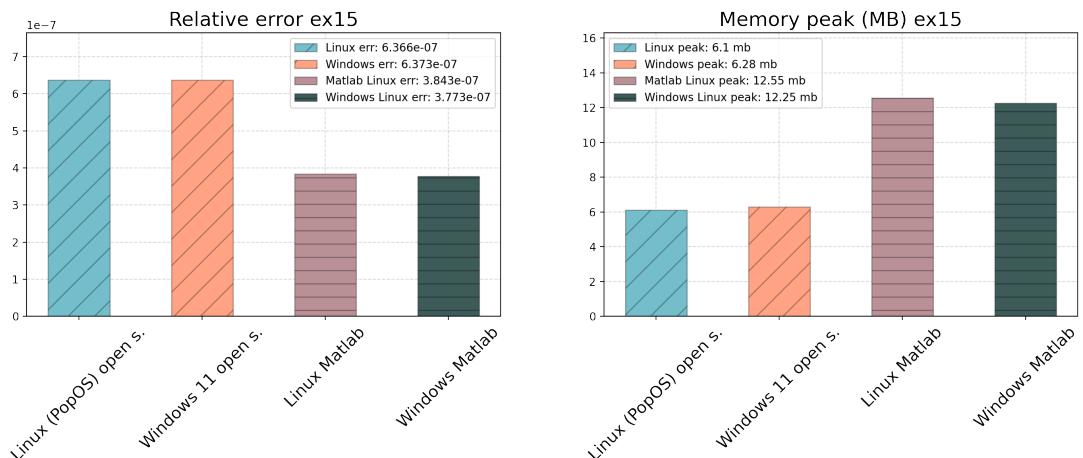


Figure 4: *Ex15* error and peak memory

**Shallow water1** La seconda matrice sparsa presentata è denominata *Shallow water1* ed è una matrice SPD con una dimensione pari a  $(81920, 81920)$ . Ha un numero di condizionamento pari a 3.6280, misurato da Matlab, e presenta 327680 elementi diversi da zero.

Come mostrato in figura 5 i tempi di caricamento sono a vantaggio di Matlab su entrambi i sistemi operativi mentre per i tempi di risoluzione, su Windows con l'impiego della libreria Scikit-sparse, risultano più lenti degli altri. Dal grafico sull'errore relativo riportato in figura 6 è invece notevole un **errore relativo generalmente basso grazie al numero di condizionamento più basso della matrice** *Shallow water1* rispetto a tutte le altre presenti. Anche l'utilizzo della memoria, come evidente dal grafico nella stessa figura, è a favore di Matlab sia su Linux che su Windows.

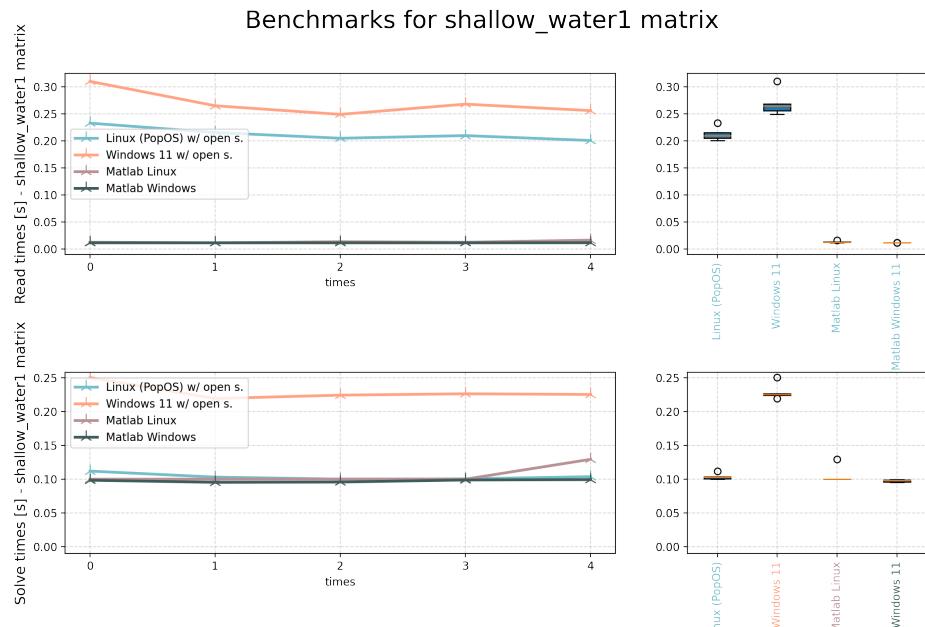


Figure 5: *Shallow water1* read and solve times

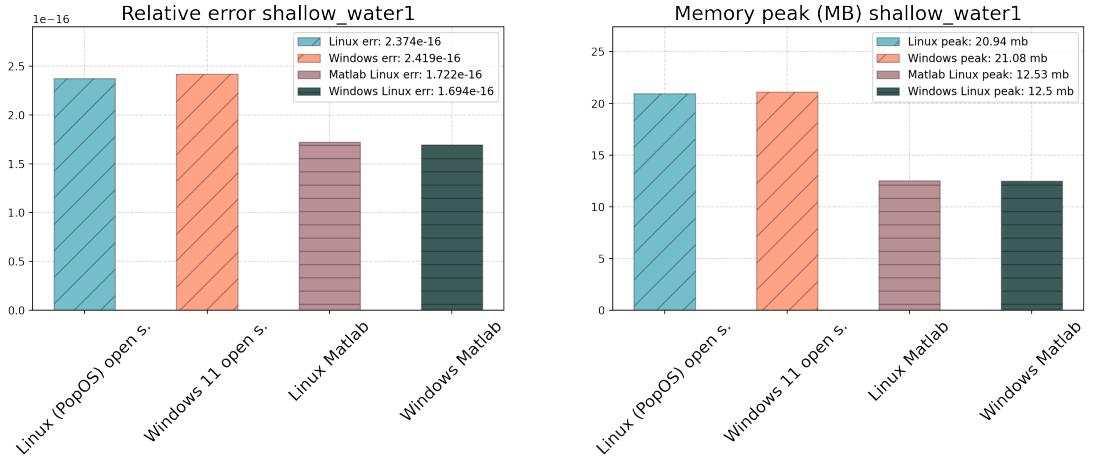


Figure 6: *Shallow water1* error and peak memory

**Parabolic fem** La terza matrice sparsa presentata è denominata *Prabolic fem* ed è una matrice SPD con una dimensione pari a (525825, 525825). Ha un numero di condizionamento pari a  $2.1108e + 05$ , misurato da Matlab, e presenta 3674625 elementi diversi da zero.

Come mostrato in figura 7, analizzando anche i boxplot a lato, è evidente come Scikit non riesca a eguagliare l'efficienza nella gestione iniziale di lettura della matrice di Matlab. Discorso diverso invece per i tempi di risoluzione del sistema dove solo Windows, con la libreria open source, risulta essere più lento rispetto a tutte le altre configurazioni. Questi risultati sono probabilmente dovuti all'ottimizzazione spiegata nel capitolo 2 relativamente all'**impiego di Math Kernel Library** ottimizzato soprattutto su kernel Linux. Per quanto riguarda l'errore relativo in figura 8 quasi tutte le configurazioni sono a pari merito ad eccezione di Windows lievemente peggiore. Ancora una volta invece Matlab si dimostra più efficiente in termini di memoria secondo le metriche e gli strumenti adottati per il test.

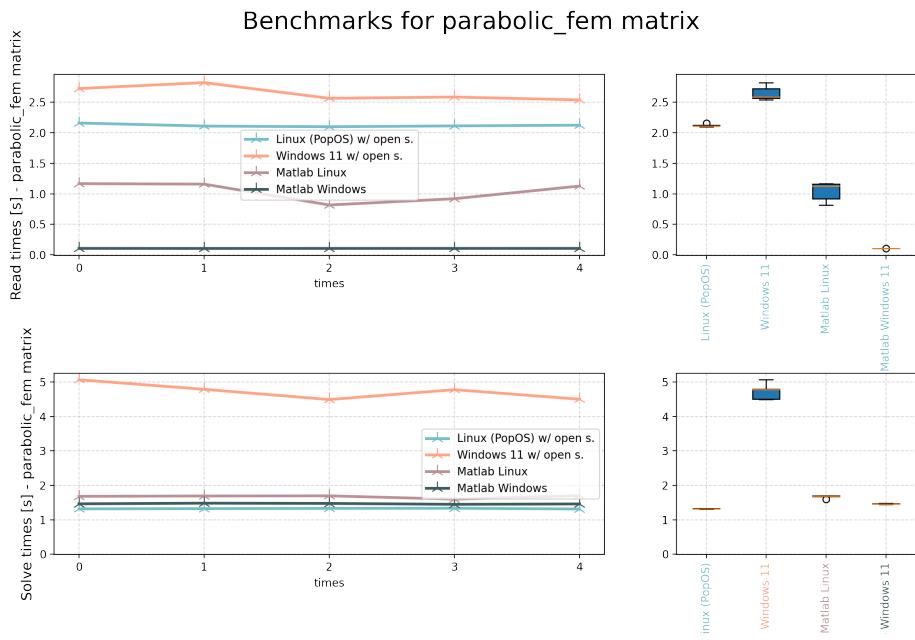


Figure 7: *Parabolic fem* read and solve times

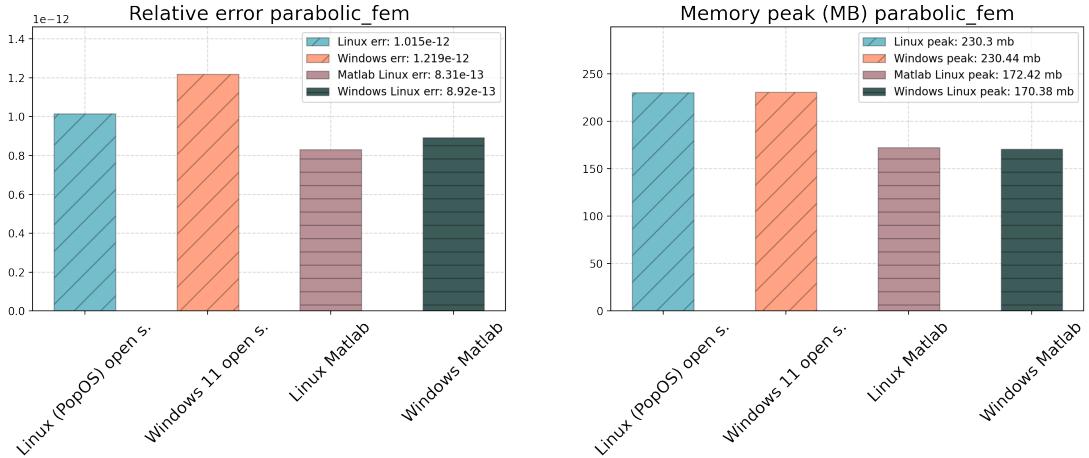


Figure 8: *Parabolic fem* error and peak memory

**Apache 2** La quarta matrice sparsa presentata è denominata *apache 2* ed è una matrice SPD con una dimensione pari a (715176, 715176). Ha un numero di condizionamento pari a  $5.3169e + 06$ , misurato da Matlab, e presenta 4817870 elementi diversi da zero.

Come mostrato in figura 9, e analogamente a quanto visto in precedenza,

la libreria Scikit si mostra più lenta nella gestione iniziale in lettura della matrice. Anche qui, come per *Parabolic fem*, l'unica libreria a ottenere delle performance peggiori è quella open source su Windows. Osservando i grafici relativi all'errore relativo e ai picchi di memoria osservati in figura 10 si può notare una quasi parità per tutti con un lieve vantaggio per la libreria open relativamente all'errore relativo e, viceversa, un lieve vantaggio per Matlab per l'utilizzo della memoria RAM.

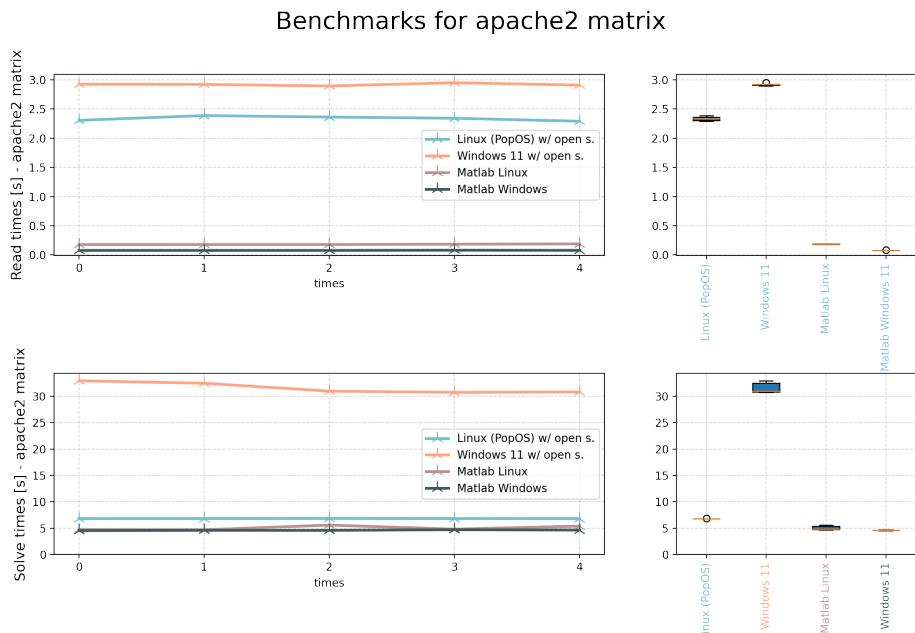


Figure 9: *Apache2* read and solve times

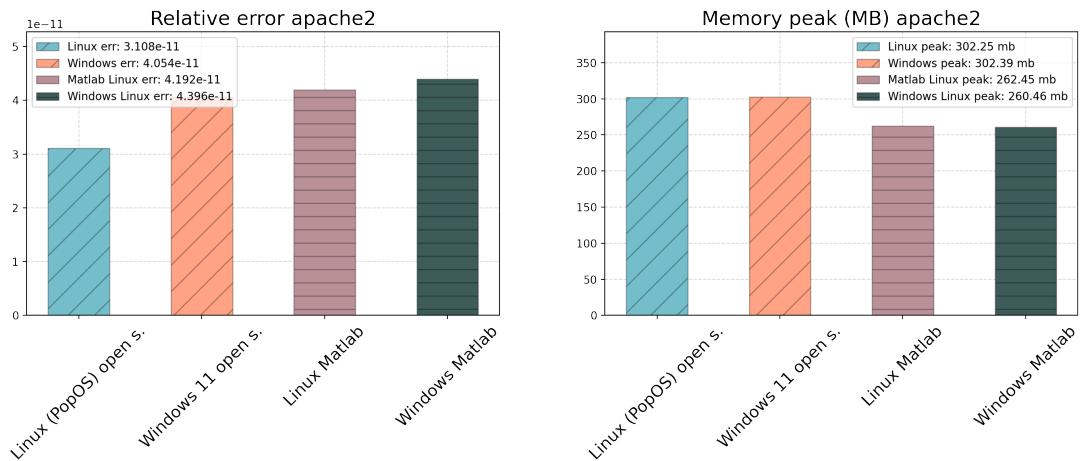


Figure 10: *Apache2* error and peak memory

**G3 circuit** La quinta matrice sparsa presentata è denominata *G3 circuit* ed è una matrice SPD con una dimensione pari a  $(1585478, 1585478)$ . Ha un numero di condizionamento pari a  $2.2384e + 07$ , misurato da Matlab, e presenta 7660820 elementi diversi da zero.

Come mostrato in figura 11 ancora una volta la libreria SciKit su Windows risulta essere la più lenta sia per i tempi di lettura che per i tempi di risoluzione del sistema lineare. Su linux invece la libreria open riesce a eguagliare le performance del software proprietario Matlab. L'errore relativo in figura 12 è invece in vantaggio per Scikit-sparse ma, come visto precedentemente, la memoria impiegata è a vantaggio di Matlab.

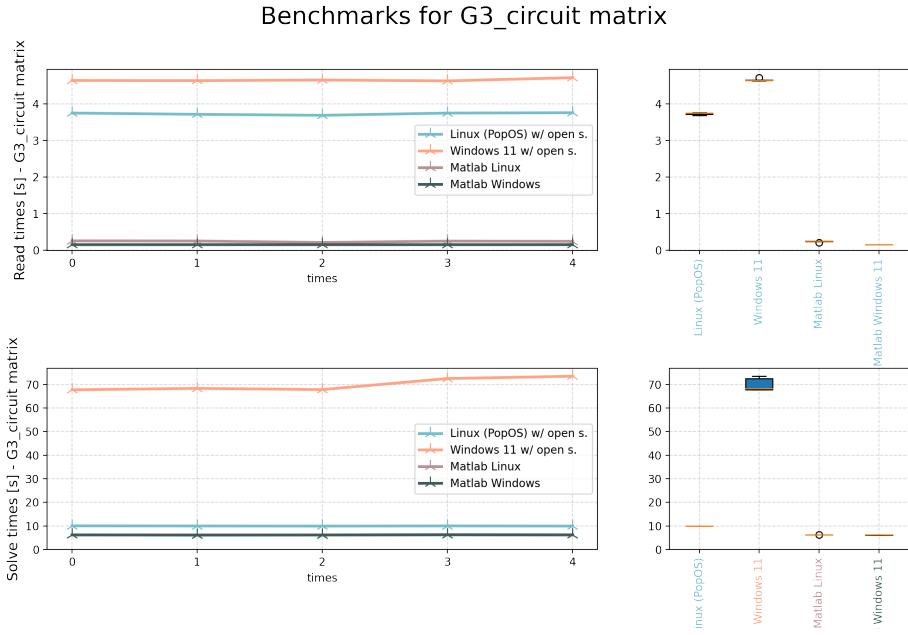


Figure 11: *G3 circuit* read and solve times

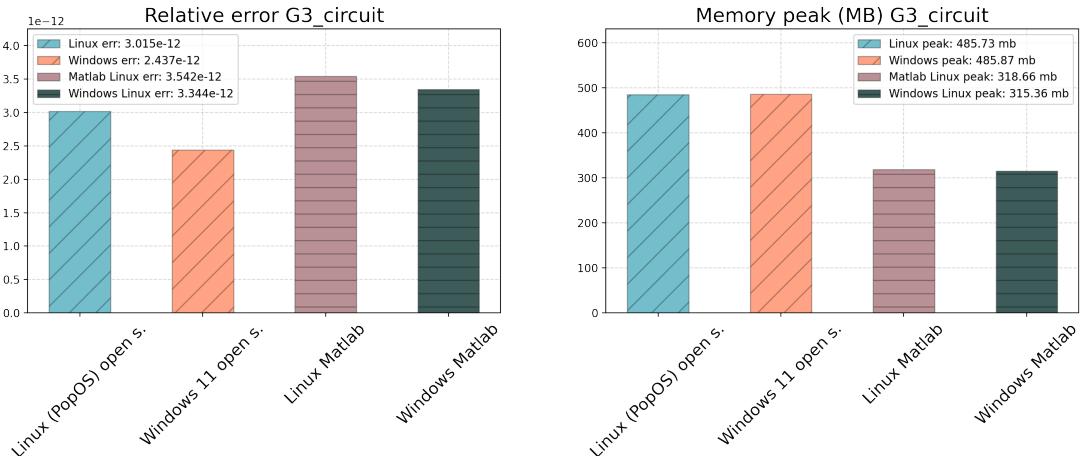


Figure 12: *G3 circuit* error and peak memory

**Cfd1** La sesta matrice sparsa presentata è denominata *cfd1* ed è una matrice SPD con una dimensione pari a (70656, 70656). Ha un numero di condizionamento pari a  $1.3351e + 06$ , misurato da Matlab, e presenta 1825580 elementi diversi da zero.

Come mostrato in figura 13, come nel caso precedente, Windows con la

libreria open source impiega molto più tempo sia nella risoluzione che nella lettura della matrice. In figura 14 è invece evidente come l'errore relativo sia a vantaggio della libreria Scikit mentre la memoria utilizzata a vantaggio di Matlab.

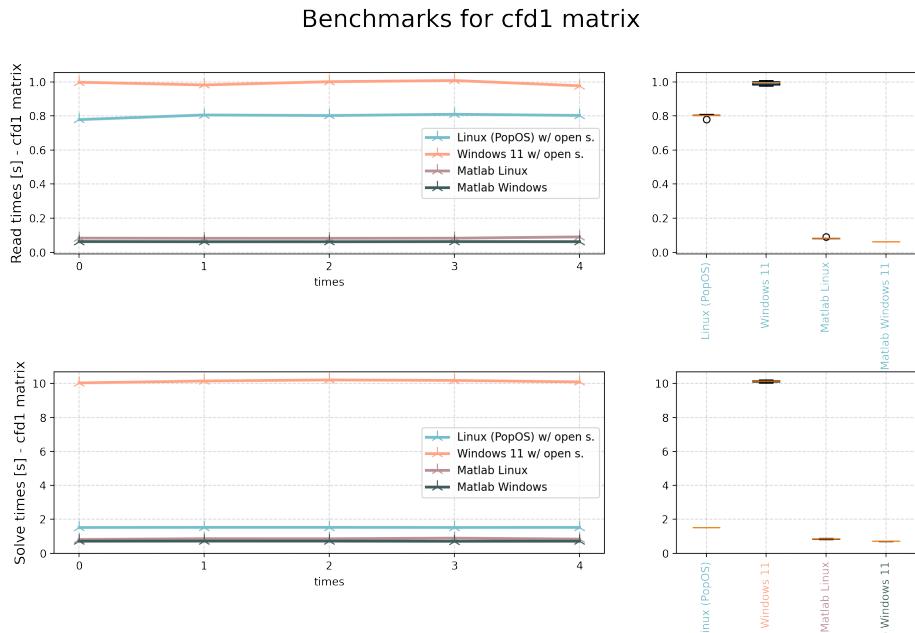


Figure 13: *cf1d* read and solve times

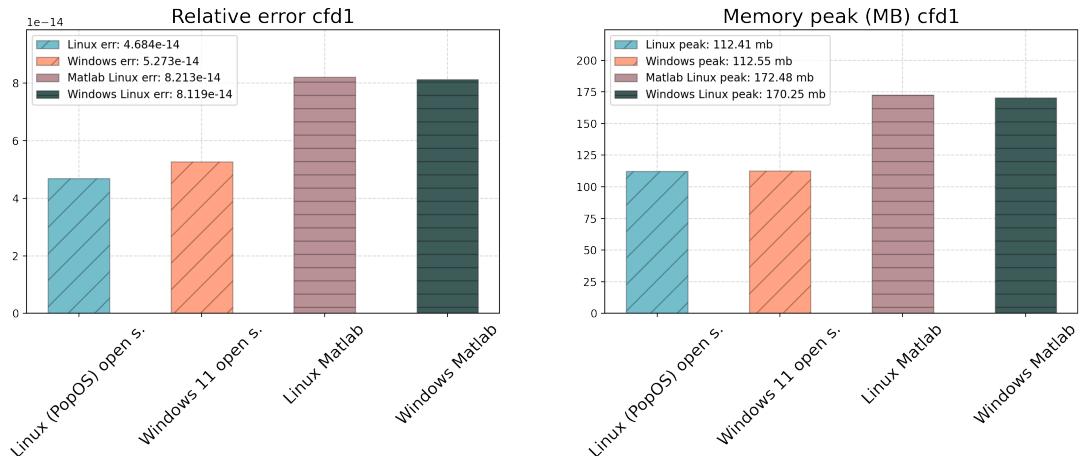


Figure 14: *cf1d* error and peak memory

**Cfd2** La settima matrice sparsa presentata è denominata *cf2d* ed è una matrice SPD con una dimensione pari a (123440, 123440). Ha un numero di condizionamento pari a  $4.2012e + 06$ , misurato da Matlab, e presenta 3085406 elementi diversi da zero.

Come mostrato in figura 15, analogamente a quanto visto precedentemente, anche qui Windows con la libreria open source risulta essere svantaggiato rispetto alla controparte Linux che, soprattutto nei tempi di risoluzione, si ottengono performance identiche per la libreria open source rispetto all'ambiente proprietario Matlab. L'errore relativo mostrato nel grafico a barre in figura 16 mostra come Windows con la libreria open non si comporta bene se confrontata con tutte le altre configurazioni. Il picco di memoria risulta invece essere molto simile per tutte le configurazioni.

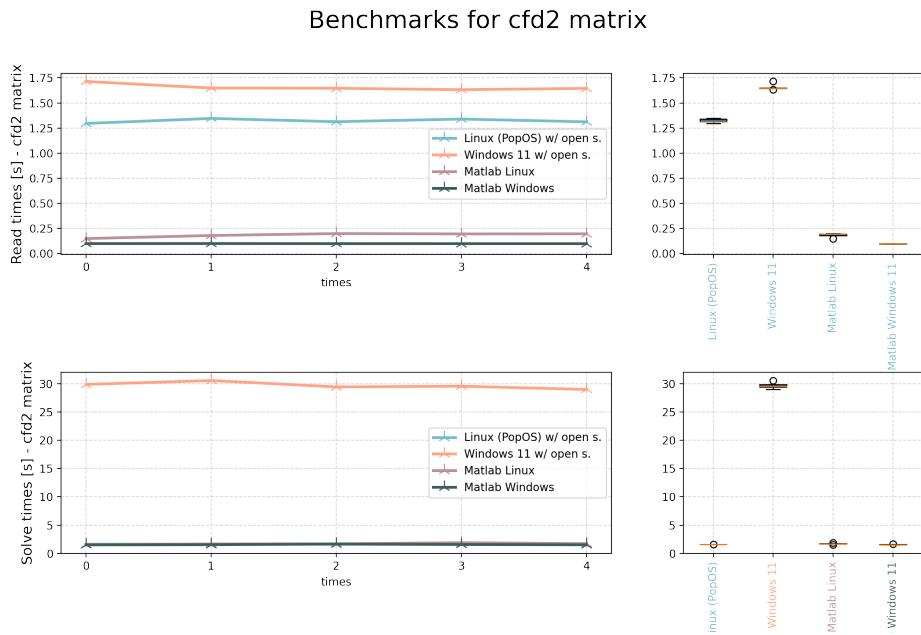


Figure 15: *cf2d* read and solve times

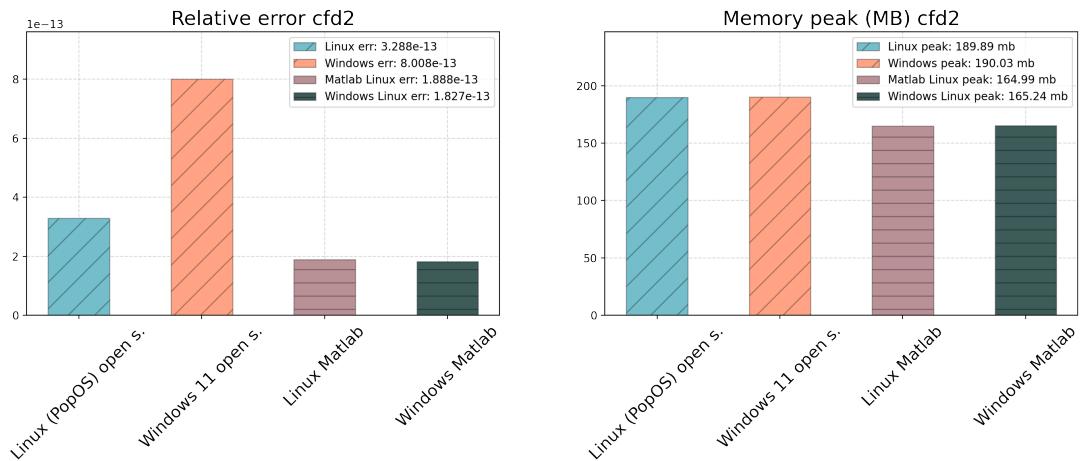


Figure 16: *cf2d* error and peak memory

**Flan 1565** L'ottava matrice sparsa presentata è denominata *flan 1565* ed è una matrice SPD con una dimensione pari a (1564794, 1564794). Il suo numero di condizionamento non viene calcolato ma presenta 114165372 elementi diversi da zero.

Come mostrato in figura 17 l'ambiente Windows con la libreria Open source non riesce a risolvere la matrice, non riuscendo quindi a mostrare i risultati vista la grandezza della stessa. Su ambiente Linux la lettura della matrice è circa 5 volte più lenta per Scikit rispetto e Matlab che però recupera in termini di tempo di risoluzione, risolvendo il sistema in circa un sesto del tempo di Matlab. Questa performance però arriva ad un costo per quanto riguarda l'errore relativo mostrato in figura 18 dove il risultato ottenuto da Scikit è di molto peggiore se confrontato con quello di Matlab. Anche il picco di memoria risulta essere più alto per l'alternativa Open Source.

Benchmarks for Flan\_1565 matrix

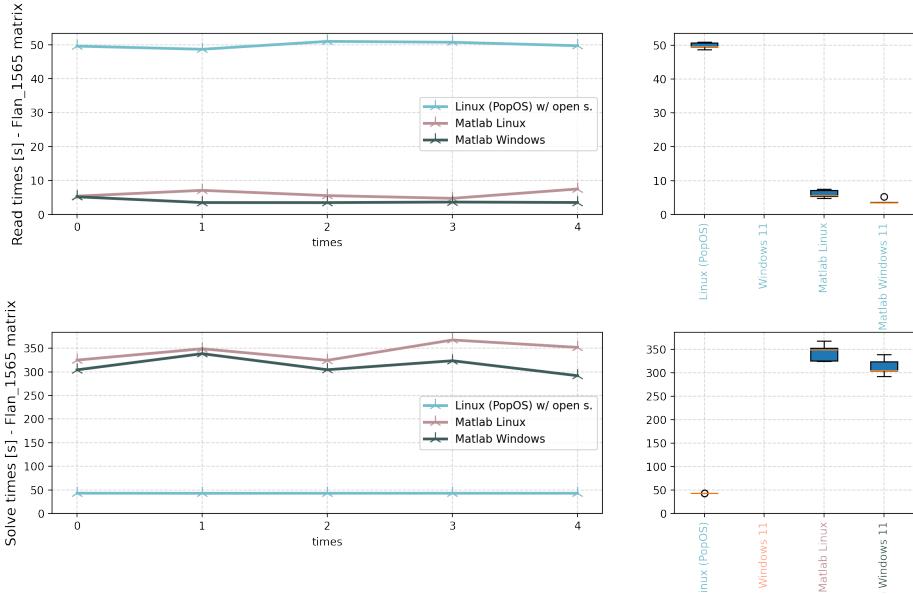


Figure 17: *Flan 1565* read and solve times

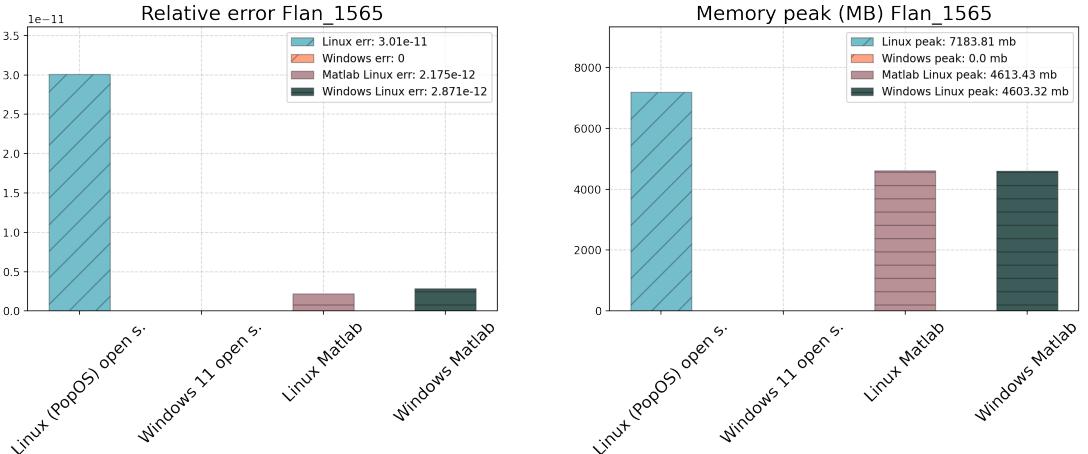


Figure 18: *Flan 1565* error and peak memory

**StocF 1465** La nona matrice sparsa presentata è denominata *StocF 1465* ed è una matrice SPD con una dimensione pari a  $(6867, 6867)$ . Ha un numero di condizionamento pari a  $1.4326e + 13$ , misurato da Matlab, e presenta 9871 elementi diversi da zero.

Come mostrato in figura 19 ancora una volta Windows non riesce a risolvere

il sistema vista la grandezza della matrice. Anche qui Scikit riesce a risolvere il sistema in molto meno tempo se confrontato con Matlab a fronte di un errore relativo più alto mostrato in figura 20. Anche il picco di memoria, come per i casi precedenti, è a svantaggio della libreria Open Source.

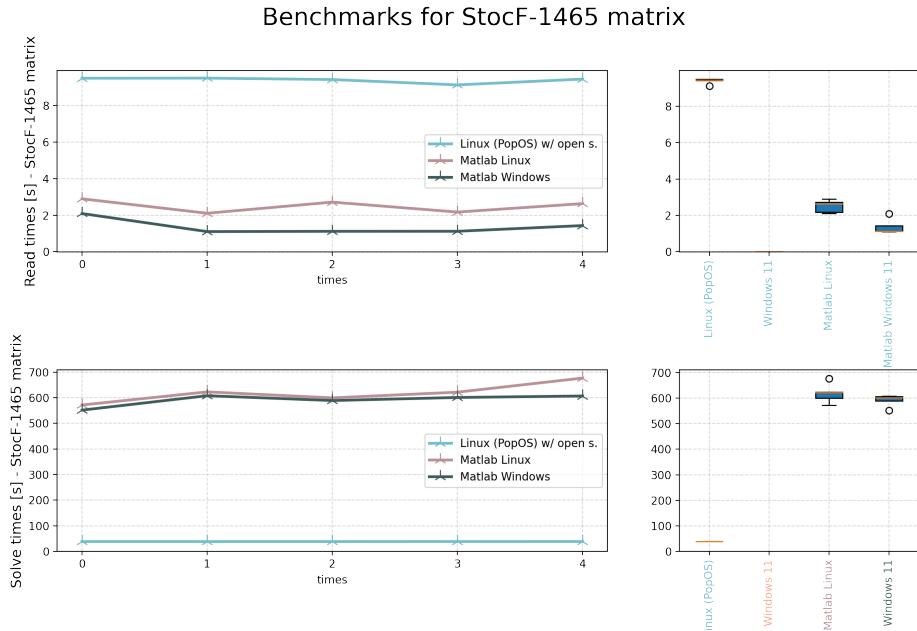


Figure 19: *StocF 1465* read and solve times

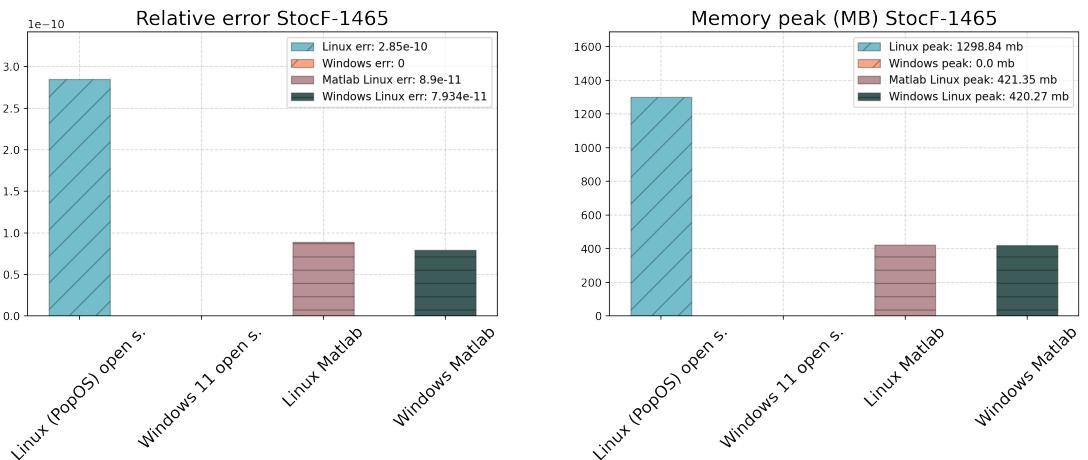


Figure 20: *StocF 1465* error and peak memory

# 5 Considerazioni conclusive

## 5.1 Grafici riepilogativi

Vengono in seguito presentati diversi grafici che mettono a confronto i diversi dati raccolti, indicando dove possibile la media e la varianza e comparando i tempi di lettura in figura 21, i tempi di risoluzione del sistema lineare in figura 22, gli errori relativi in figura 23 e, infine, il picco di memoria raggiunto durante la risoluzione del sistema in figura 24.

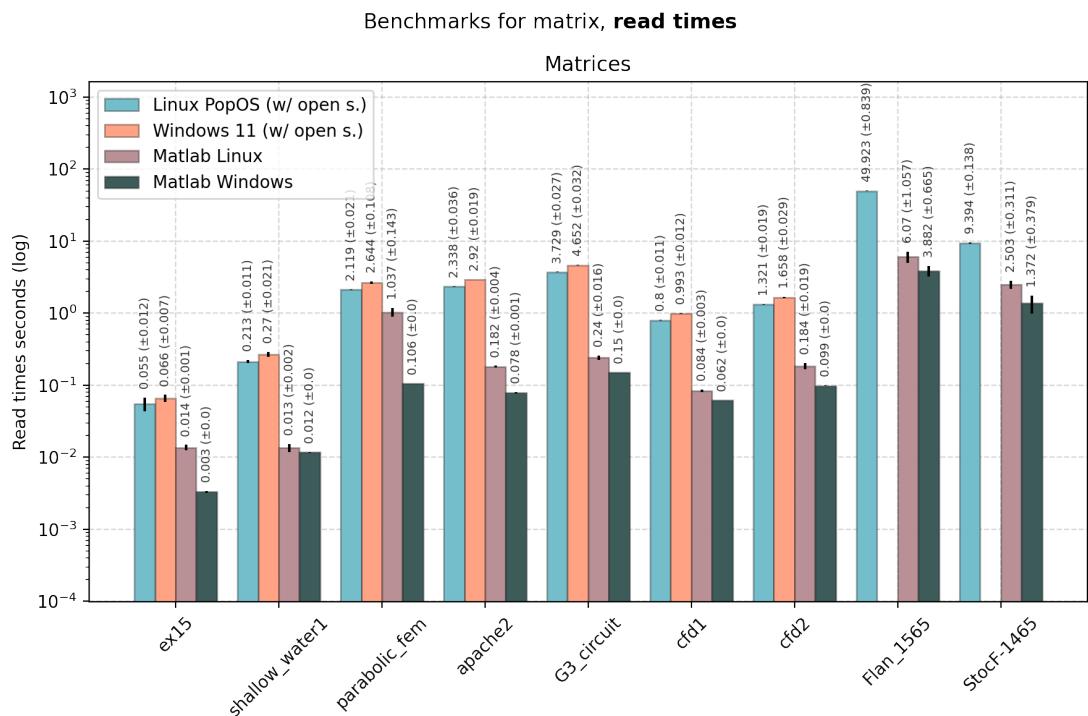


Figure 21: Per ogni matrice viene confrontata, usando una scala logaritmica, i tempi di lettura per le diverse configurazioni

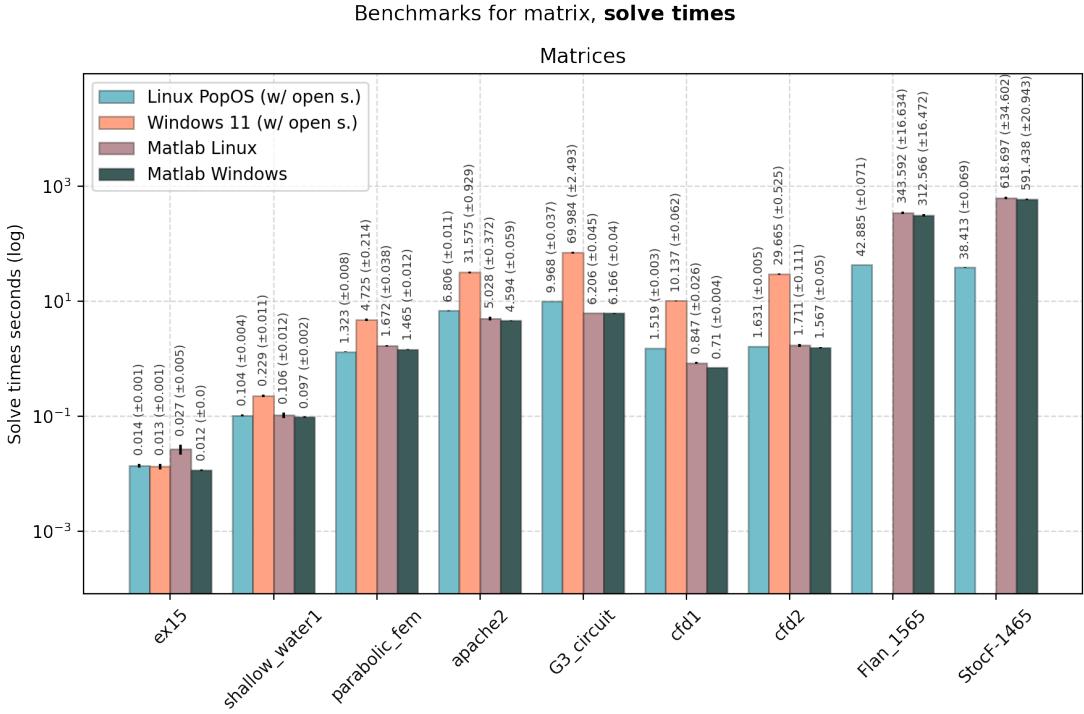


Figure 22: Per ogni matrice viene confrontata, usando una scala logaritmica, i tempi di risoluzione del sistema lineare per le diverse configurazioni

## 5.2 Sviluppi futuri

Lungo il corso del progetto sono stati confrontati due diversi ambienti di risoluzione e due diversi sistemi operativi. In questa sezione viene esplorata invece una diversa architettura, basata su ARM, che sembra essere promettente per la sua velocità generale di calcolo, compreso quello destinato ad un ambito scientifico.

Nel dettaglio, usando l'ultima versione disponibile di Python è stato possibile avviare lo stesso programma su un'architettura ARM based, implementata da Apple sotto il nome di Apple Silicon. È importante segnalare come l'installazione non è stata semplice seguendo i metodi tradizionali ma si è dovuta compilare la libreria direttamente dai sorgenti in una versione ancora non rilasciata come pacchettizzata e pronta all'uso sui repository ufficiali. I dettagli della macchina utilizzata per il test sono disponibili in tabella 1 e tutto è stato effettuato sul sistema operativo Mac OS, comunque Unix based. Purtroppo non è stato possibile eseguire ulteriori test che permettessero di avere un confronto equo con le altre configurazioni precedentemente affrontate, vista l'assenza di una versione di

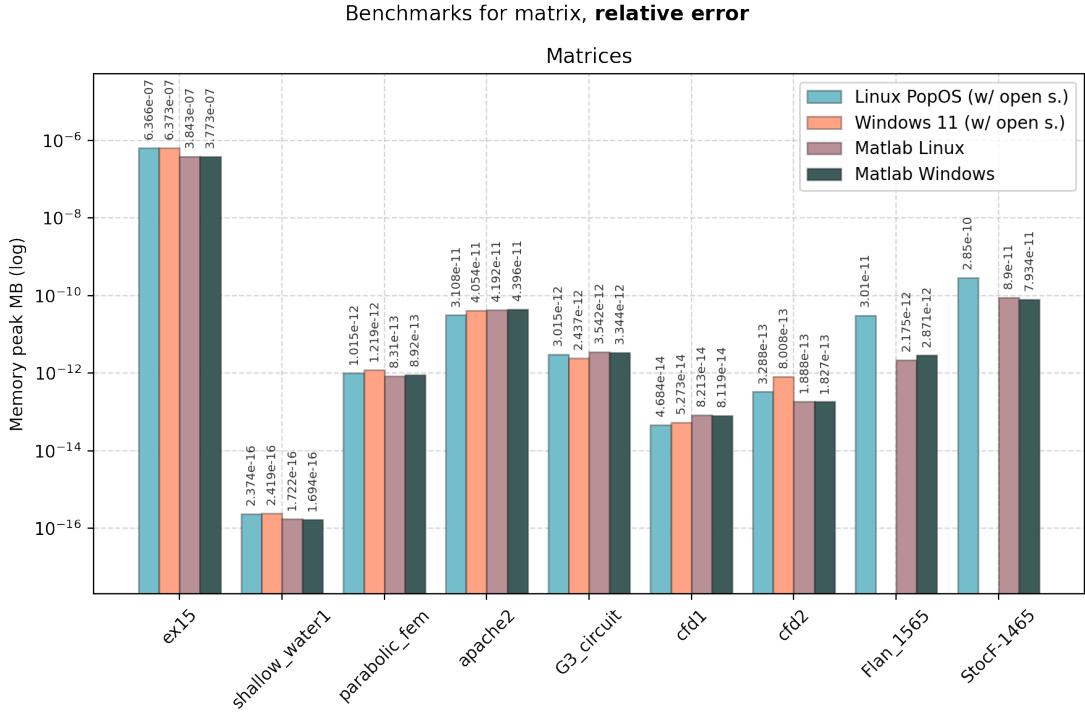


Figure 23: Per ogni matrice viene confrontata, usando una scala logaritmica, gli errori relativi sulla soluzione del sistema per le diverse configurazioni

Matlab compatibile con Apple Silicon (se non attraverso un layer di compatibilità *Rosetta 2* che penalizza però le prestazioni) oltre che l'assenza di una versione di Windows o di Linux ottimizzata per questa architettura.

Nome: MacBook Pro 16" 2021

CPU: Apple Silicon M1 Max - 10 Core (2 efficiency, 8 performance)

GPU: Apple Silicon M1 Max - 32 Core (w/Metal)

RAM: 32 GB

SSD: 1 TB

Table 1: Specifiche tecniche sul laptop usato per i benchmarks

I risultati ottenuti, come per gli esempi precedenti, sono riportati in figura 25 per i tempi di lettura delle matrici e in figura 26 per i tempi di risoluzione del sistema. Non vengono riportati i grafici relativi alla memoria occupata o all'errore relativo vista la quasi coincidenza con quanto riportato precedentemente con i

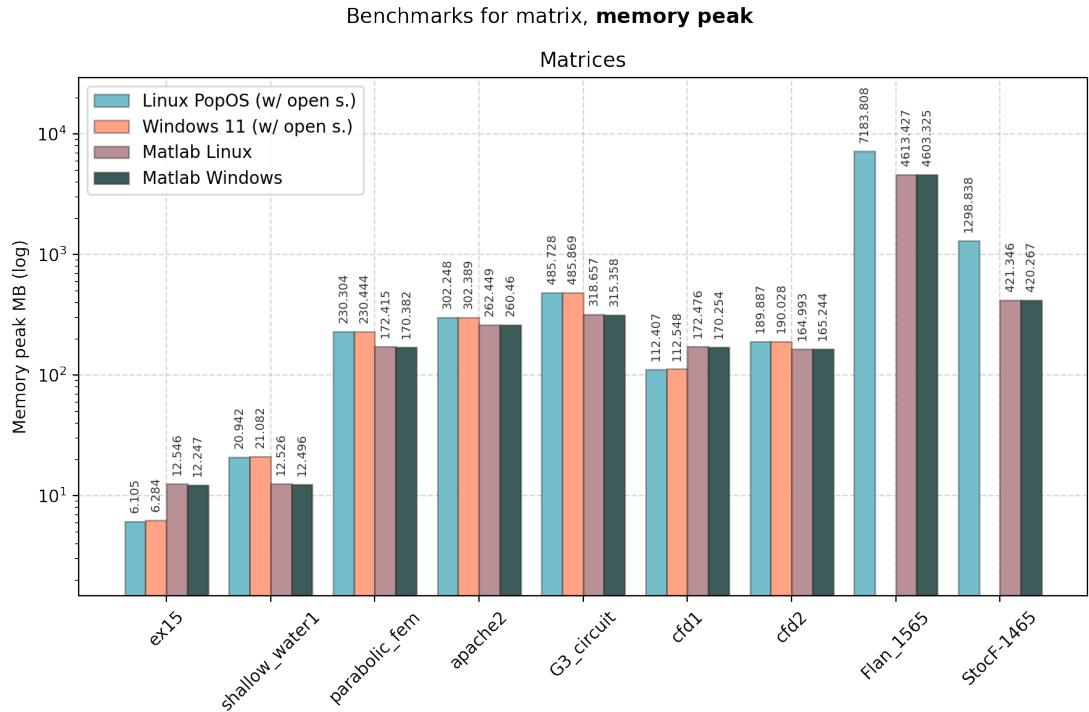


Figure 24: Per ogni matrice viene confrontata, usando una scala logaritmica, il picco di memoria utilizzata per la risoluzione dei sistemi dalle diverse configurazioni

risultati della libreria Open Source su Linux.

Dall’analisi dei risultati si evince come le prestazioni siano decisamente superiori rispetto alle altre piattaforme. Nonostante la CPU desktop utilizzata sia attualmente tra le più performanti per il mercato consumer la piattaforma ARM, su di un laptop, riesce ad ottenere performance decisamente superiori, dimezzando i tempi soprattutto per le matrici più grandi.

Si ricorda che i risultati non sono ovviamente confrontabili vista l’assenza di pari condizioni iniziali delle macchine ma è stato comunque deciso di inserire i risultati all’interno del report per segnalarne le potenzialità in eventuali sviluppi futuri sia dell’architettura ARM che del software creato ad hoc per essa.

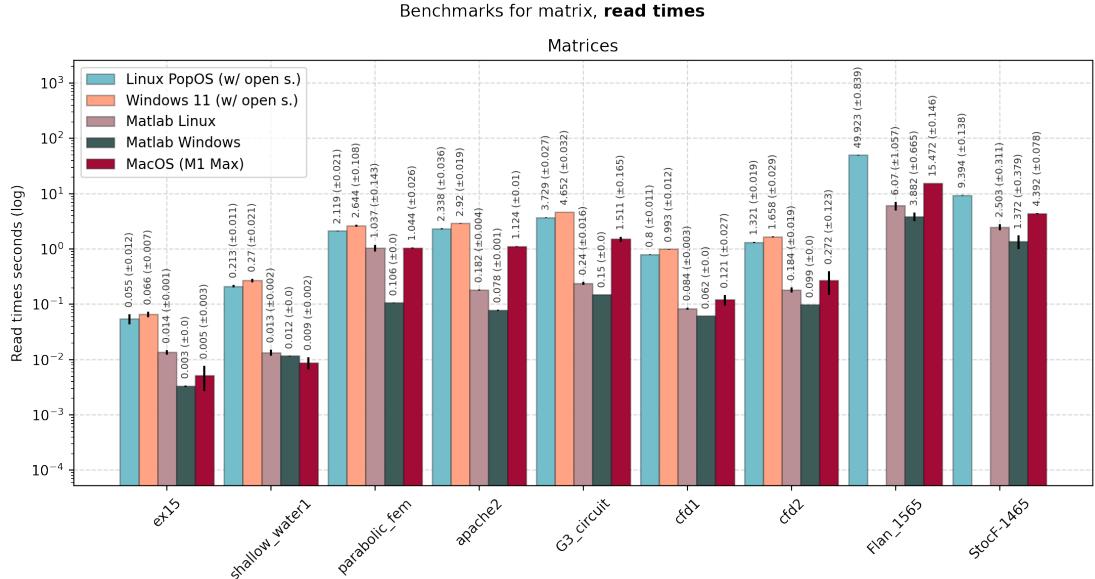


Figure 25: Risultati sui tempi di lettura della matrice della piattaforma ARM messi a confronto con gli esperimenti precedentemente effettuati su x86

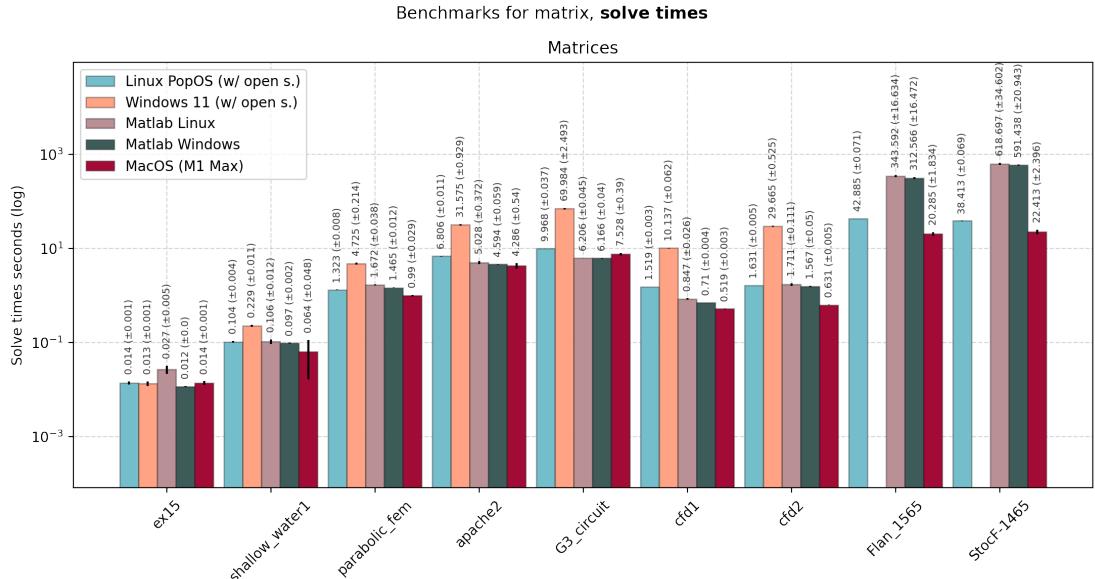


Figure 26: Risultati sui tempi di risoluzione del sistema della piattaforma ARM messi a confronto con gli esperimenti precedentemente effettuati su x86

### 5.3 Conclusioni

Visti i dati riportati è possibile fare delle osservazioni sulle diverse configurazioni analizzate.

In primo piano è utile evidenziare come non sia ovvia la scelta ma, di volta in volta, esistono opzioni più appropriate in base al caso di utilizzo. La migliore configurazione possibile è rappresentata da Matlab in ambiente Unix/Like, nello specifico con una distribuzione Linux e basandosi sulle ottimizzazioni mostrate nel capitolo 2.

Come prima variabile viene considerata la scelta del sistema operativo che, analizzando i dati riportati, risulta essere quasi ovvia. **Windows è il sistema che porta più svantaggi**, sia in termini di prestazioni, sia per la sua difficoltà nell'installazione delle diverse librerie e dipendenze riguardo gli ambienti open source. Considerando anche il suo costo di licenza e la quasi parità tra i risultati di Matlab è in definitiva nettamente svantaggiato se confrontato con gli ambienti Unix tendenzialmente gratuiti e con una maggiore diffusione. I sistemi operativi basati su Linux inoltre consentono una maggiore stabilità nel tempo (con distribuzioni e versioni *stable*) e una personalizzazione maggiore per aspetti più tecnici che, nelle mani di un esperto, possono rivelarsi utili per task particolarmente impegnativi. Quest'ipotesi, oltre a essere confermata dai dati qui presenti, è ulteriormente confermata dal largo impiego di Linux in ambito scientifico, principalmente installato su macchine remote il cui unico compito è quello di effettuare calcoli computazionalmente onerosi.

Il software di sviluppo invece è una variabile da analizzare più nel dettaglio. Osservando i tempi di esecuzione e risoluzione del codice è possibile notare che **Matlab risulta essere più veloce in molti casi** se confrontato librerie open source considerate. Un'analisi più approfondita rivela però che **Matlab restituisce in molti casi un errore relativo più alto** rispetto alla controparte Open Source che, al costo di un tempo di risoluzione più alto, fornisce un risultato qualitativamente migliore, anche se non di moltissimo. Questo **concetto è invertito soprattutto in presenza di matrici di grandi dimensioni** come le ultime due *Flan 1565* e *StocF 1465*: in questi due casi Matlab fornisce una soluzione più precisa di Scikit (ma comunque mantenendo all'incirca lo stesso ordine di grandezza) ma con un costo temporale decisamente più elevato.

Ultima variabile, anche se meno importante, è rappresentata dall'**utilizzo della memoria dove Matlab risulta essere leggermente in vantaggio**. In ogni caso l'ordine di grandezza è espresso in MB per la maggior parte delle

matrici e, solo nei casi più computazionalmente impegnativi, raggiunge i pochi GB. Presupposto che le matrici utilizzate durante i test rappresentino quanto più possibile un caso di impiego reale e se si considera il quantitativo di RAM generalmente impiegato in macchine destinate ad un utilizzo scientifico è immediato capire che la memoria utilizzata non risulta essere un problema così grave da affrontare.

**In definitiva**, per un ambiente produttivo, dove è necessario avere soluzioni accurate quanto più velocemente possibile Matlab in ambiente Linux risulta essere la soluzione migliore, anche considerata la sua semplicità maggiore data da un IDE già ampiamente conosciuto. Se invece si predilige la velocità di calcolo, un costo di mantenimento più basso (o quasi nullo) e una maggiore personalizzazione al costo di un risultato qualitativamente minore allora l'utilizzo di Python con la libreria Scikit-sparse risulta essere l'opzione migliore.