Metodi del Calcolo Scientifico

Risoluzione di sistemi lineari tramite il metodo di Cholesky

Edoardo Silva 816560
Bryan Zhigui 816335
Davide Marchetti 815990

A.A. 2019/2020

Abstract

Questo progetto si prefigge lo scopo di studiare l'implementazione del metodo di Choleski per la risoluzione sistemi lineari per matrici simmetriche definite positive sparse in un ambiente di programmazione open source e di compararli con l'implementazione closed-source di MATLAB.

Il confronto avverrà in termini di tempo, accuratezza, impiego della memoria e anche facilità d'uso sia in ambiente Linux che Windows, eseguendo il codice su diverse matrici derivate da problemi reali e raccolte nella **SuiteSparse Matrix Collection**[1].

[1] https://sparse.tamu.edu/

1 Analisi dell'implementazione

1.1 MATLAB

Per la decomposizione di cholesky, MATLAB mette a disposizione il modulo **cholesky.matlab** contenente tutto il necessario. In particolare è stata utilizzata la funzione **chol**

Utilizzo

`R = chol(A, [triangle])`: Fattorizza la matrice A simmetrica definita positiva in una matrice triangolare superiore R tale che $A = R^{-1}R$. Il parametro `triangle` permette di scegliere se attuare la decomposizione in una matrice triangolare superiore (opzione di default) o triangolare inferiore. In quest'ultimo caso, la matrice R risultante dall'equazione soddisferà l'uguaglianza $A = RR^{-1}$.

Manutenzione

La libreria è stata rilasciata per la prima volta nell'aggiornamento R2013a MATLAB. Attualmente, è ancora supportata e non presenta lacune o problemi che sono stati riscontrati durante il suo l'utilizzo.

1.1.1 Documentazione

La documentazione ufficiale di MATLAB è ben strutturata e di facile consultazione. Vengono anche proposti molti esempi di utilizzo delle diverse funzioni in varie casistiche.

1.1.2 Problemi riscontrati

La criticità principale riguarda il supporto del modulo `memory` esclusivamente per sistemi operativi Windows. Difatti, è stato necessario utilizzare due metodi di profiling della memoria diversi a seconda del sistema operativo.

Licenza

Essendo MATLAB un software closed-source, non è possibile accedere al codice sorgente del modulo.

1.2 Open-Source (C++)

Dopo un'attenta analisi e comparazione di diverse opzioni, l'implementazione in C++ è stata costruita utilizzando **Eigen**, libreria che si pone l'obiettivo di essere leggera ed offire supporto alle operazioni su vettori e matrici dense e sparse.

Utilizzo

`Eigen::loadMarket(A, filename)`: Importa i valori di una matrice sparsa memorizzata in un file `.mtx` nella matrice fornita come primo argomento. Nel nostro programma, `A` è definita come `Eigen::SparseMatrix<double>`.

Il modulo `unsupported/Eigen/SparseExtra` che contiene queste funzionalità è attualmente deprecato.

1.2.1 Manutenzione

Eigen è in sviluppo attivo, tuttavia, alcuni moduli sono marcati come deprecati e non ne è garantito il loro pieno funzionamento. Un esempio di questi è la classe `MarketIO`, che permette di effettuare operazioni di input e ouput con file in formato `Matrix Market (.mtx)`.

1.2.2 Documentazione

La documentazione di Eigen è di facile consulto e abbastanza completa. Tuttavia, non si può fare la stessa considerazione rispetto ai moduli `unsupported`, per i quali la documentazione è spesso riferita a versioni precedenti o non presente.

1.2.3 Problemi riscontrati

Durante lo sviluppo, l'utilizzo di una classe deprecata ha inizialmente rallentato lo sviluppo. Infatti, delle matrici importate tramite `MarketIO` veniva ignorato il fatto che fossero salvate come simmetriche o meno.

Questo inconveniente è stato risolto modificando lo script di conversione MATLAB `mmwrite` per trasformare file `.mat` in `.mtx` e rigenerando le matrici assicurandosi che tutti gli elementi venissero salvati correttamente.

Tuttavia, questa trasformazione aggiuntiva comporta uno spreco di spazio su disco per memorizzare il doppio dei valori rispetto ad un semplice indicatore che specifiche qualora la matrice sia simmetrica, e di tempo per la lettura dell'intera matrice da file.

1.2.4 Licenza

Eigen è un software gratuito ed open-source rilasciato con licenza Mozilla Public License 2.0 (MPL2: simple weak copyleft license) dalla versione 3.1.1.

2 Dettagli implementativi

Riporteremo di seguito le sezioni semplificate delle parti principali di ciascun'implementazione.

2.1 MATLAB

TODO

Listing 1: MATLAB: Algoritmo principale

```
function [rows, memory_delta, solve_time, error] = chol_solve(name)
    load(fullfile('', 'matrix_mat', name), "Problem");

    [user] = memory;
    proc_memory_start = user.MemUsedMATLAB;

    rows = size(Problem.A, 1);
    x_es = ones(rows, 1);
    b = Problem.A * x_es;

    tic;
    R = chol(Problem.A);
    x_ap = R \ (R' \ b);
    solve_time = toc;

    [user] = memory;
    memory_delta = user.MemUsedMATLAB - proc_memory_start;

    error = norm(x_ap - x_es) / norm(x_es);
end
```

2.2 C++

TODO

Listing 2: C++: Algoritmo principale

```
result analyze_matrix(std::string filename) {
    result r;
    SpMat A; // Eigen::SparseMatrix<double>
```

```
    ull start_mem, end_mem;

    Eigen::loadMarket(A, filename);
    r.size = A.rows();

    start_mem = memory::process_current_physical();

    Eigen::VectorXd x_es = Eigen::VectorXd::Ones(A.rows());
    Eigen::VectorXd b(A.rows());
    b = A*x_es;

    auto chol_start = std::chrono::high_resolution_clock::now();
    Eigen::SimplicialCholesky<SpMat> chol(A);
    Eigen::VectorXd x_ap = chol.solve(b);
    auto chol_finish = std::chrono::high_resolution_clock::now();

    end_mem = memory::process_current_physical();

    r.memory_delta = end_mem - start_mem;
    r.solve_time = chol_finish - chol_start;
    r.relative_error = (x_ap - x_es).norm() / x_es.norm();
}
```

TODO

Listing 3: C++: Struct per la memorizzazione del risultato

```
typedef unsigned long long ull;
struct result {
    unsigned int size;
    ull memory_delta;
    std::chrono::duration<double> solve_time;
    double relative_error;
};
```

3 Specifiche hardware

La piattaforma utilizzata per la produzione dei risultati riportati nelle sezioni 5 e 6, è composta come segue:

- **CPU**: AMD Ryzen 5 3600 - 6 Core / 12 Threads - 3.60Ghz /4.20Ghz
- **RAM**: Crucial Ballistix DDR4-3000C15 2*8Gb (16Gb) a 3800MHz
- **HDD**: Western Digital Green 1TB HDD
- **GPU**: Sapphire RX 580 Pulse (8GB VRam)

Il disco utilizzato per l'esecuzione in ambiente Windows è un **SSD Samsung 850 EVO (250Gb)**, mentre Linux è installato su un **NVMe Sabrent (256Gb)**.

4 Matrici analizzate

L'analisi si è concentrata sulle seguenti matrici:

Nome	**Dimensione**	**NNZ**
ex15	6.867	98.671
shallow_water1	81.920	327.680
cfd1	70.656	1.825.580
cfd2	123.440	3.085.406
parabolic_fem	525.825	3.674.625
apache2	715.176	4.817.870
G3_circuit	1.585.478	7.660.826
StocF-1465	1.465.137	21.005.389
Flan_1565	1.564.794	114.165.372

Tabella 1: Matrici analizzate

5 Risultati per sistema operativo

5.1 Windows

Tempo

All'incremento delle dimensioni della matrice di input non si manifesta una crescita lineare né costante in termini di tempo. Ciononostante, i tempi di esecuzione nell'implementazione in MATLAB risultano essere meno variabili rispetto alla controparte in C++).

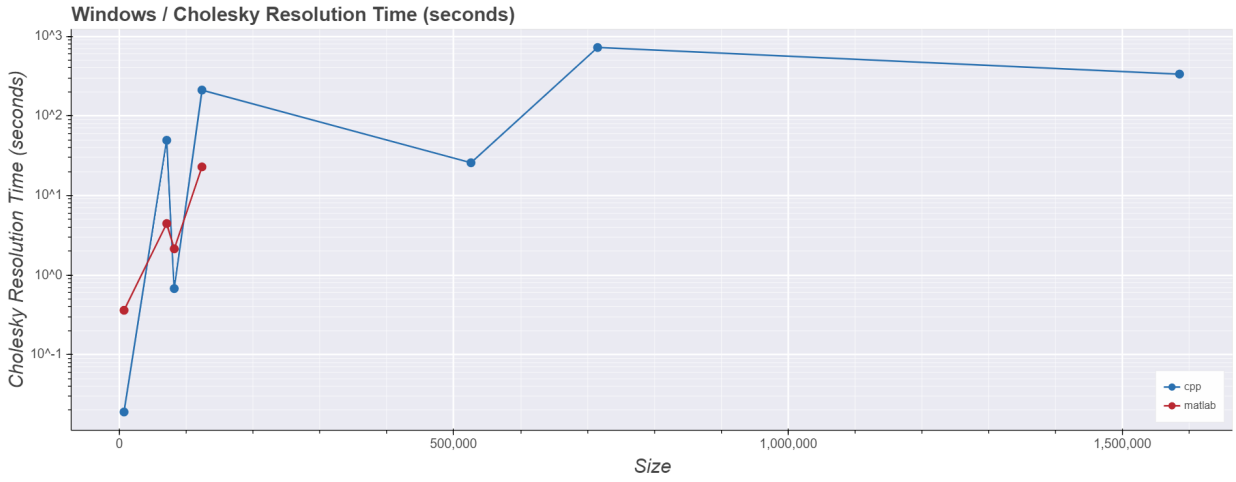

Figura 1: Tempo di risoluzione su Windows

Memoria

Come illustrato in fig. 2, l'implementazione in C++ occupa molta meno memoria rispetto a quella in MATLAB, in particolare al crescere della dimensione della matrice e del numero di elementi contenuti in essa.

Difatti, tutte le matrici analizzate in MATLAB per le quali non è presente un risultato nel grafico, durante la decomposizione di Cholesky hanno comportato un errore `Out of memory`.

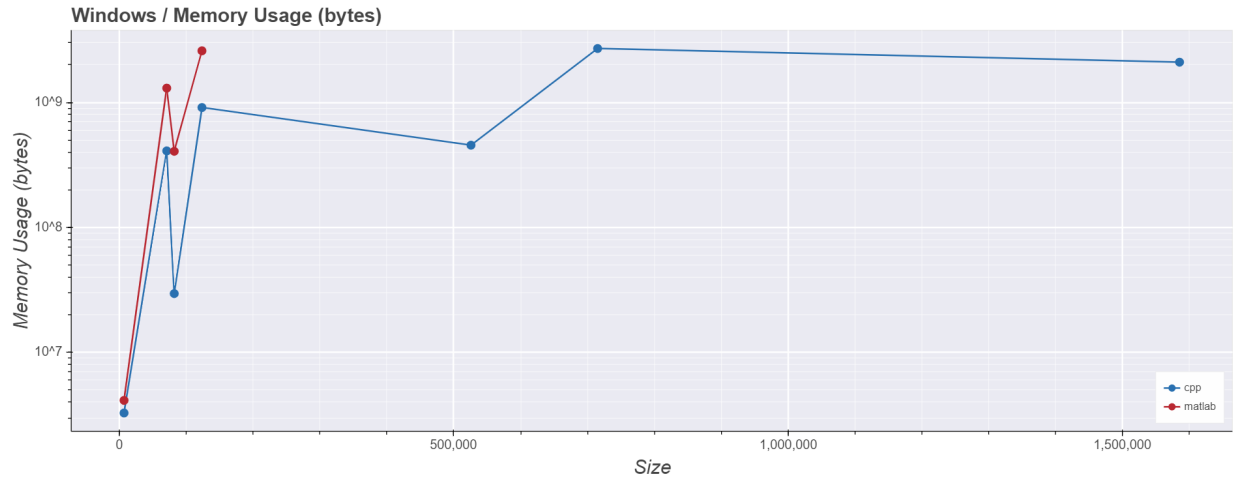

Figura 2: Utilizzo della memoria su Windows

Errore Relativo

Entrambe le implementazioni presentano un errore relativo molto simile. In fig. 3 si nota come al crescere della dimensione della matrice, in C++ l'errore relativo sembra stabilizzarsi tra 10^{-10} e 10^{-12}.

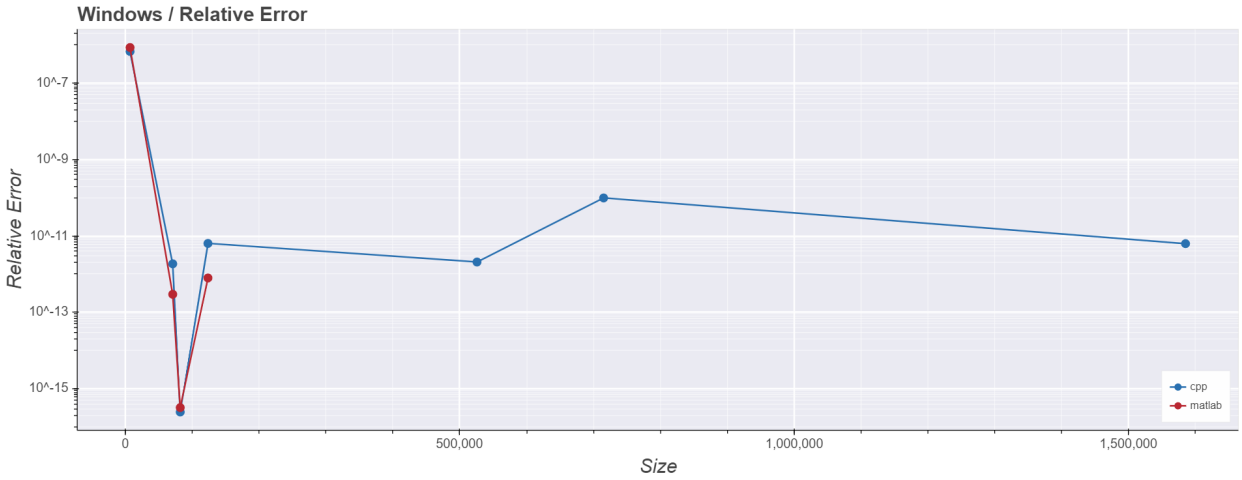

Figura 3: Errore relativo su Windows

5.2 Linux

Tempo

L'andamento del tempo di risoluzione del sistema non risulta strettamente dipendente dalla dimensione della matrice. Inoltre, l'implementazione in MATLAB riesce a mantenere tempi di completamento meno variabili.

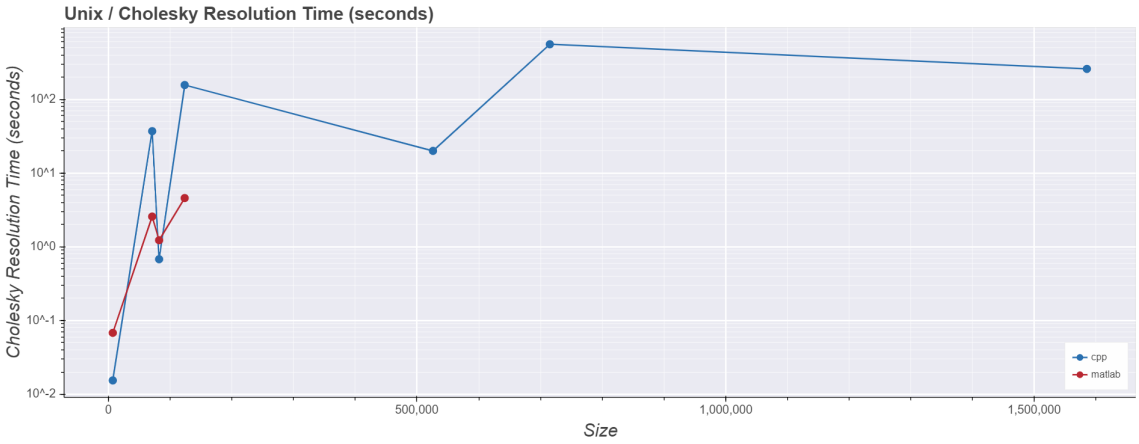

Figura 4: Tempo di risoluzione su Linux

Memoria

L'utilizzo della memoria, a differenza di quanto visto in precedenza nella fig. 2 presenta una lettura iniziale molto bassa nell'implementazione MATLAB.

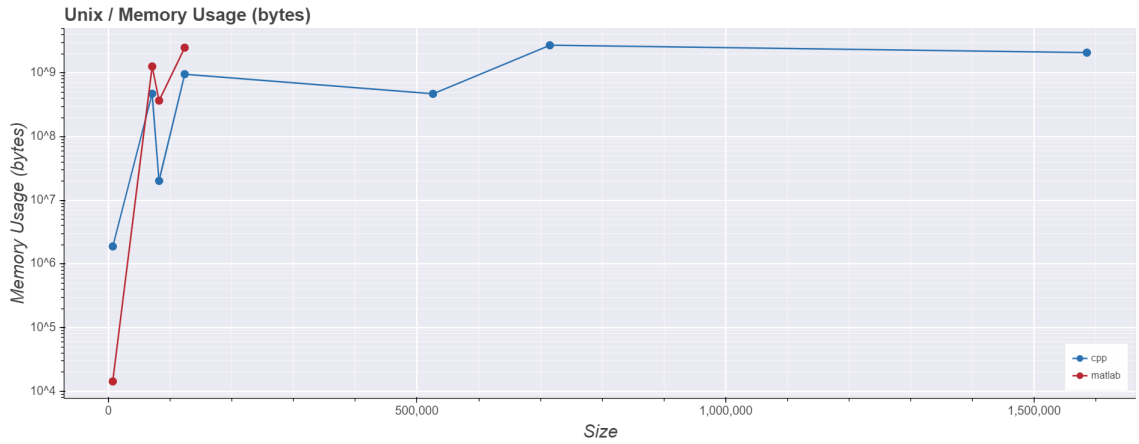

Figura 5: Utilizzo della memoria su Linux

Errore relativo

Si nota che, sia su c++ che con la libreria MatLab, si ha un errore comparabile/quasi uguale per entrambi i metodi, e che dopo un massimo ed un minimo nelle matrici più piccole, l'errore si stabilizza intorno a 10^{-11}, indipendentemente dalle dimensioni delle matrici che il programma riesce a caricare.

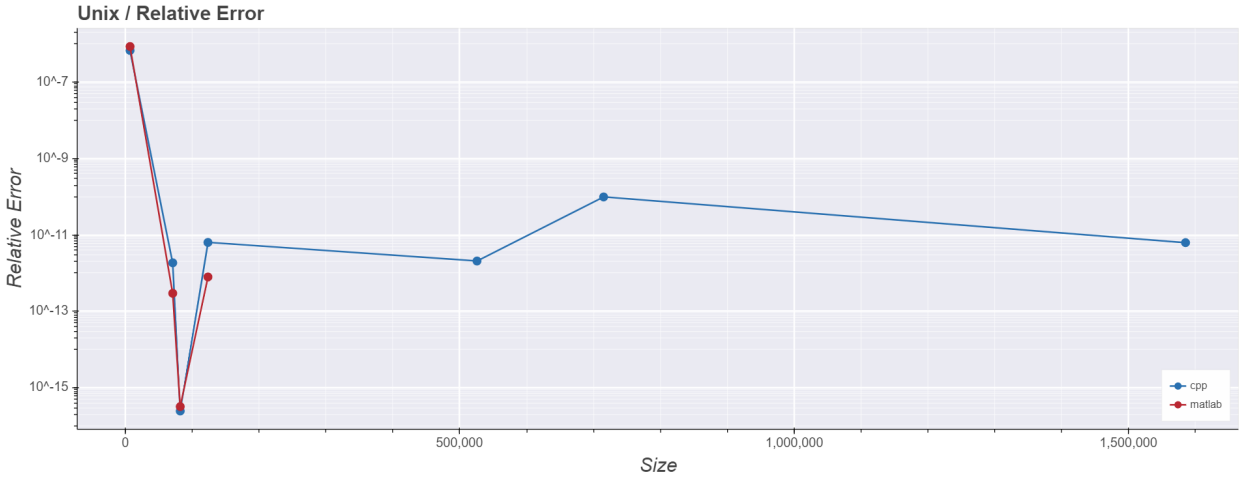

Figura 6: Errore relativo su Linux

6 Risultati per implementazione

6.1 C++

Tempo

Come si può notare nella figura sottostante, Linux è leggermente più efficiente di Windows a sfruttare la libreria Eigen per c++, di un valore irrisorio quasi costante.

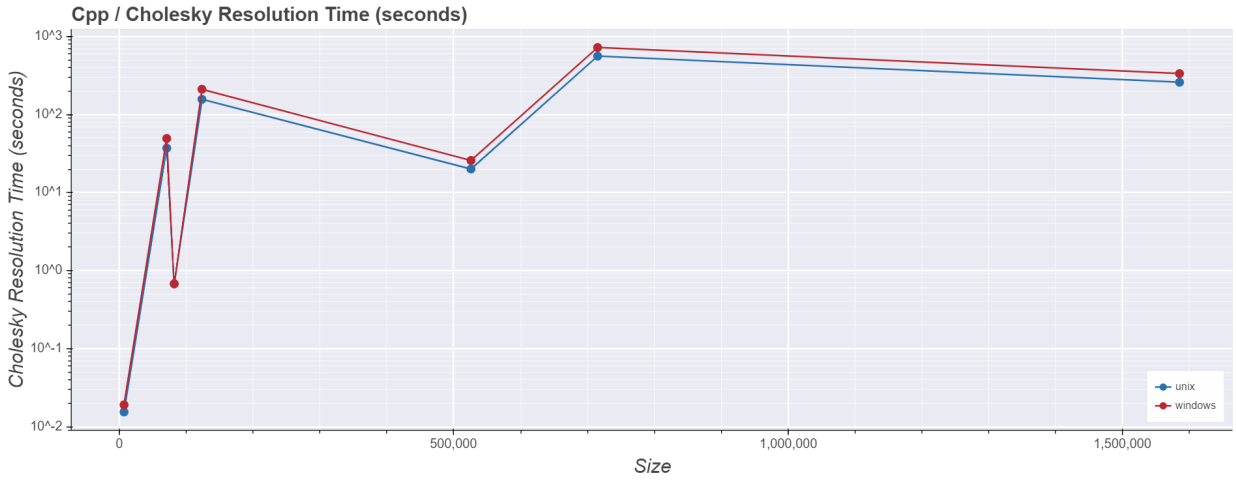

Figura 7: Tempo di risoluzione nell'implementazione C++

Errore Relativo

L'errore relativo calcolato con Eigen in entrambe le architetture è identico.

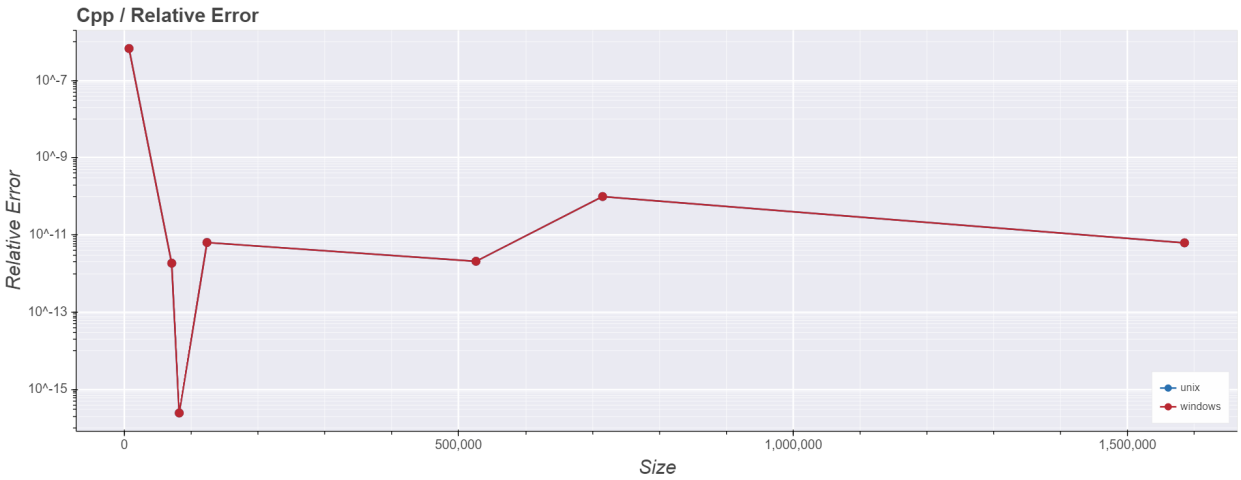

Figura 8: Errore relativo nell'implementazione C++

Memoria

È possibile vedere come la memoria usata sia identica su tra le due architetture per matrici di dimensioni superiori a 500000.
le differenze analizzate sono irrisorie.

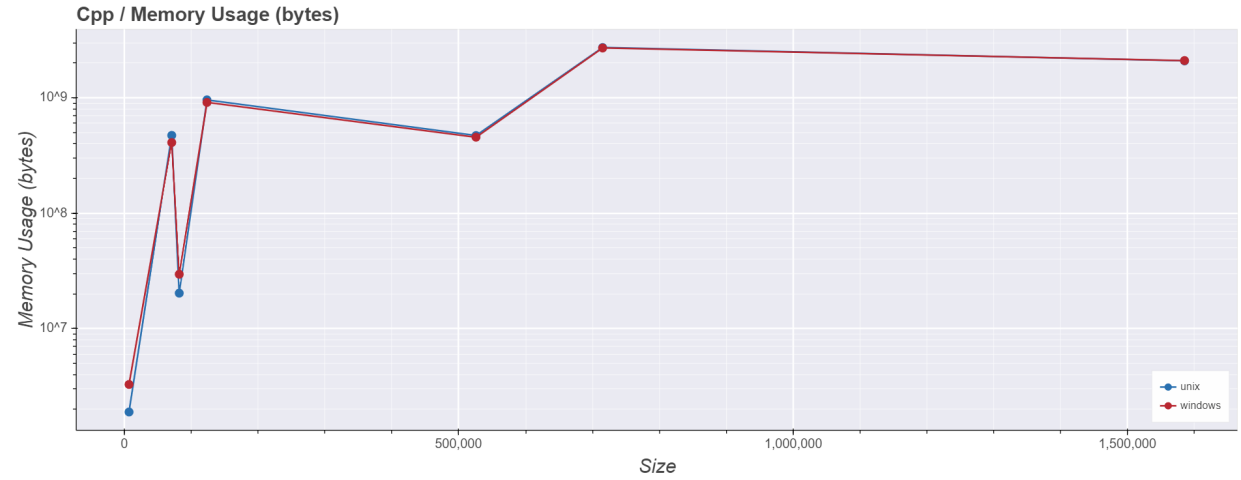

Figura 9: Utilizzo della memoria nell'implementazione C++

6.2 Matlab

Tempo

La figura seguente mostra come Matlab sia decisamente più veloce su un architettura Unix di circa mezzo ordine di grandezza.

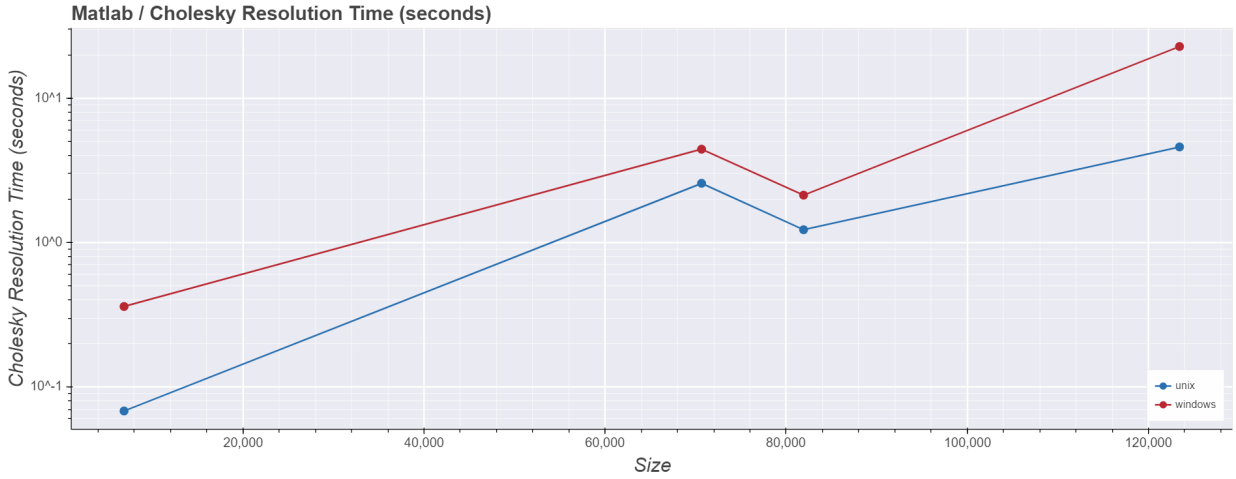

Figura 10: Tempo di risoluzione nell'implementazione MATLAB

Errore Relativo

L'errore relativo dei risultati della libreria Matlab non cambia a seconda delle architettetture.

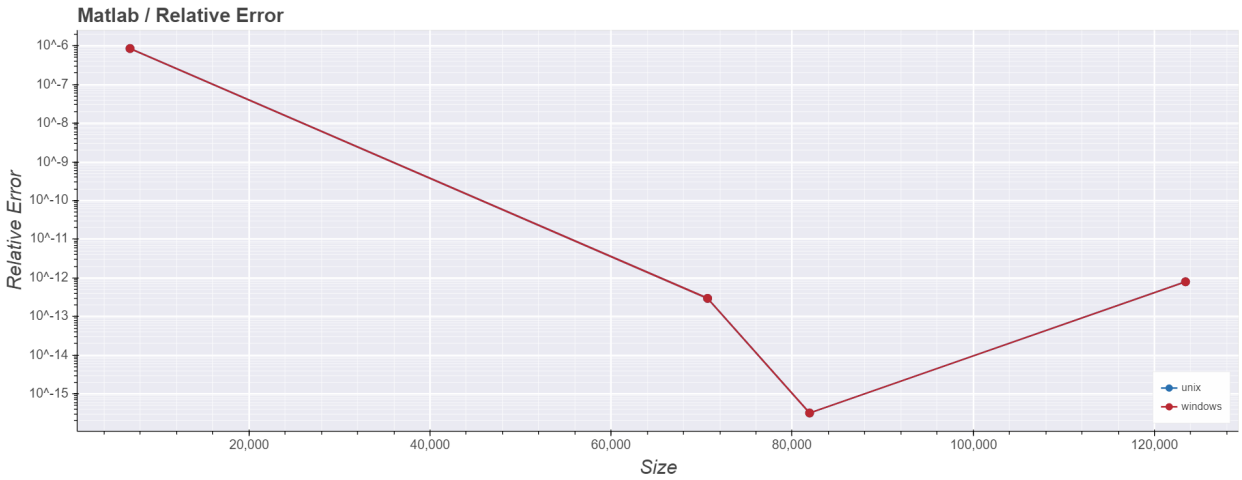

Figura 11: Errore relativo nell'implementazione MATLAB

Memoria

La differenza di memoria utilizzata nelle due architetture è visibile solo sulla prima matrice, favorendo Unix.
È affermabile che l'architettura su cui viene utilizzato MatLab sia irrilevante.

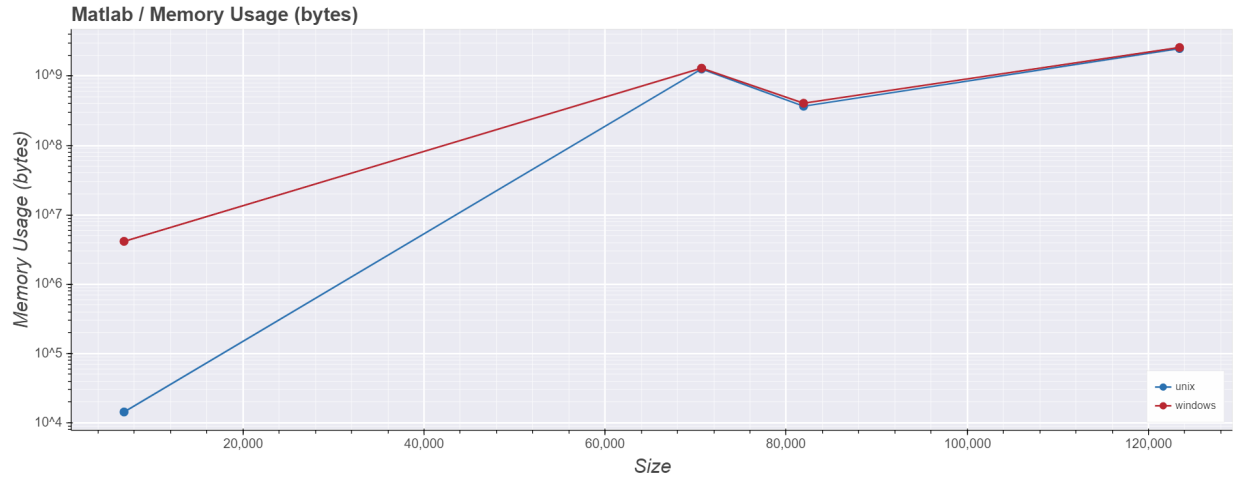

Figura 12: Utilizzo della memoria nell'implementazione MATLAB

7 Conclusioni

TODO Pare che l'architettura Unix sia marginalmente migliore di Windows in merito alle prestazioni.
I risultati mostrano come la libreria open source Eigen per C++ abbia risultati quasi sempre migliori rispetto a MatLab, ma ottimizzi la memoria e ciò permette di lavorare su matrici di dimensioni maggiori a parità di hardware. A discapito di un utilizzo più complesso e di maggior codice necessario.

7.1 Suddivisione del lavoro

Durante la realizzazione del progetto tutti i componenti del gruppo hanno partecipato attivamente alla sua realizzazione. In particolare:

- **Edoardo Silva** si è occupato...
- **Davide Marchetti** si è occupato...
- **Bryan Zhigui** si è occupato...

Tutti i componenti del gruppo hanno lavorato alla ... e collaborato alla stesura di questo documento.