Metodi del Calcolo Scientifico

Risoluzione di sistemi lineari tramite il metodo di Cholesky

Silva Edoardo 816560
Zhigui Bryan 816335
Marchetti Davide 815990

A.A. 2019/2020

Abstract

Lo scopo di questo progetto è di studiare l'implementazione del metodo di Choleski per la risoluzione sistemi lineari per matrici sparse, simmetriche e defite positive in ambienti di programmazione open source e di compararli con l'implementazione 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 sparse derivate da problemi reali e raccolte nella **SuiteSparse Matrix Collection**.

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 traiangolare 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.

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<Type>`.

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

- **Eigen::VectorXd::Ones(A.rows()):** dichiara matrice di dimensioni fissate (prese dalle dimensioni della matrice A), package 'VectorXd' usato per le operazioni su matrici dinamiche di double.
- **Eigen::SimplicialCholesky<SpMat> chol(A):** Pacchetto creato per gestire matrici di grandi dimensioni con pochi elementi diversi da 0. Implementa uno schema di rappresentazione e gestione dei valori diversi da 0 con uso di poca memoria e alte prestazioni.
  Il metodo chol(A) implementa la fattorizzazione di Cholesky della matrice A.
- **Eigen::VectorXd x_ap = chol.solve(b):** Applicazione del risolutore iterativo per risolvere la fattorizzazione.

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 il modulo MarketIO, che permette di effettuare operazioni di Input e Ouput con file in formato Matrix Market (.mtx).

1.2.2 Problemi

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.

1.2.3 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 Specifiche hardware

La piattaforma utilizzata per la produzione dei risultati riportati nelle sezioni 3 e 4, 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)

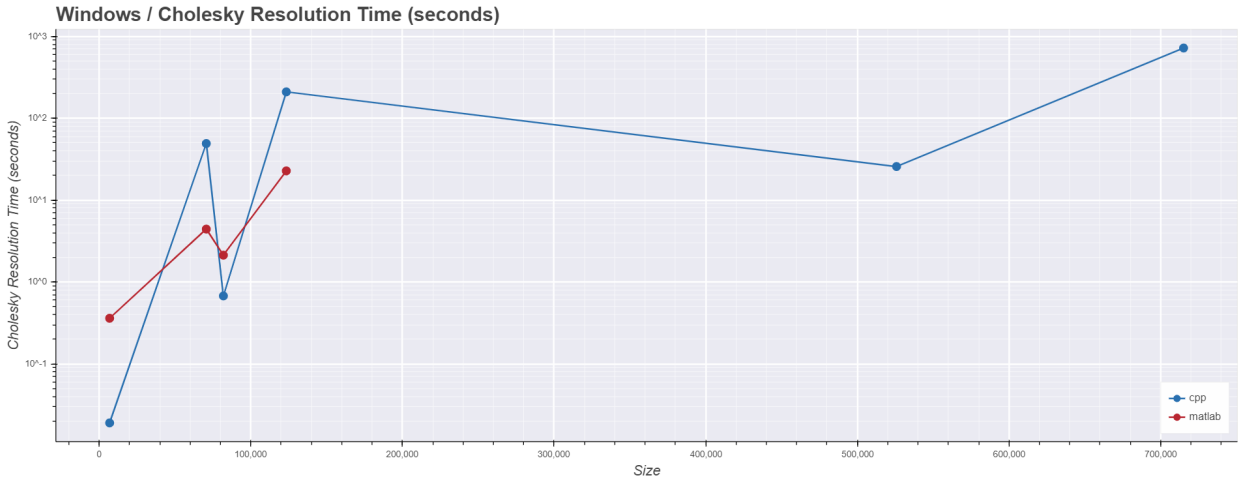
Il disco utilizzato per l'esecuzione in ambiente Windows è un **SSD Samsung 850 EVO (250Gb)**, mentre Linux è installato su un **NVMe Sabrent (256Gb)**.

3 Risultati per sistema operativo

3.1 Windows

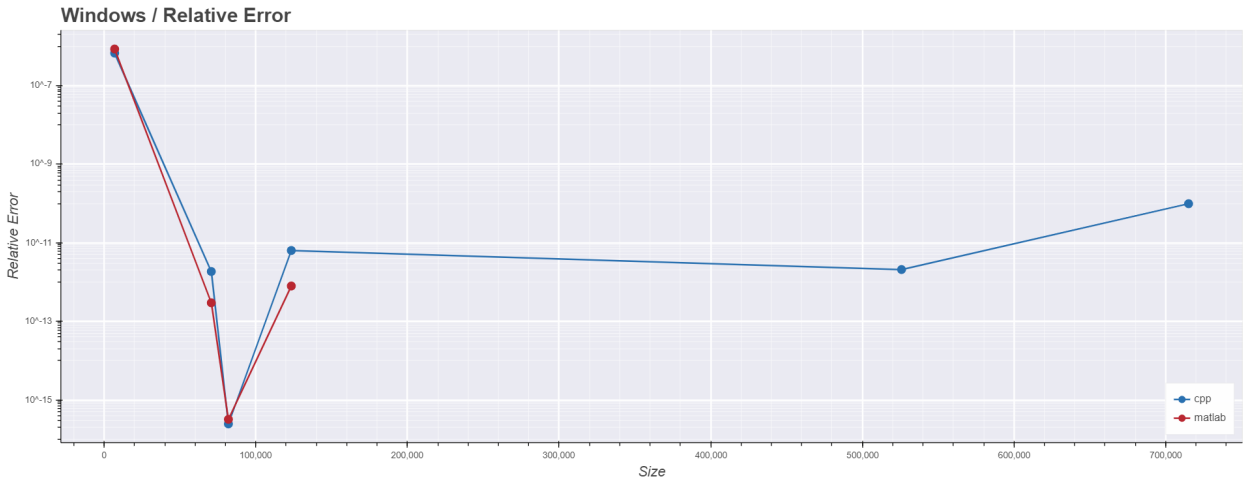
Tempo

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



Errore Relativo

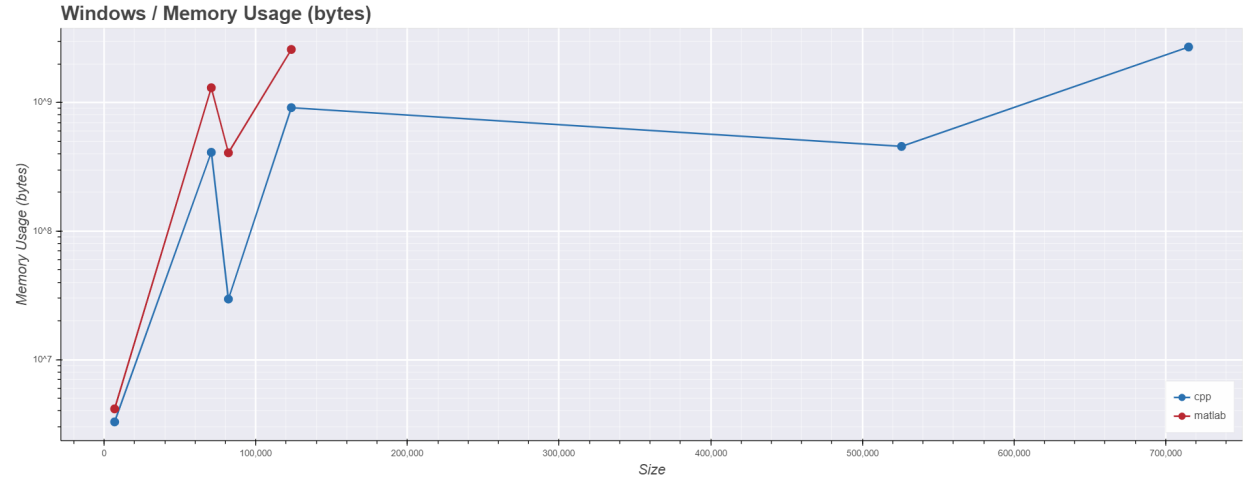
Entrambe le implementazioni presentano un errore relativo molto simile. Al crescere della dimensione della matrice, in C++ l'errore relativo sembra stabilizzarsi tra 10^{-10} e 10^{-12}.



Memoria

Come illustrato in sezione 3.1, 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.

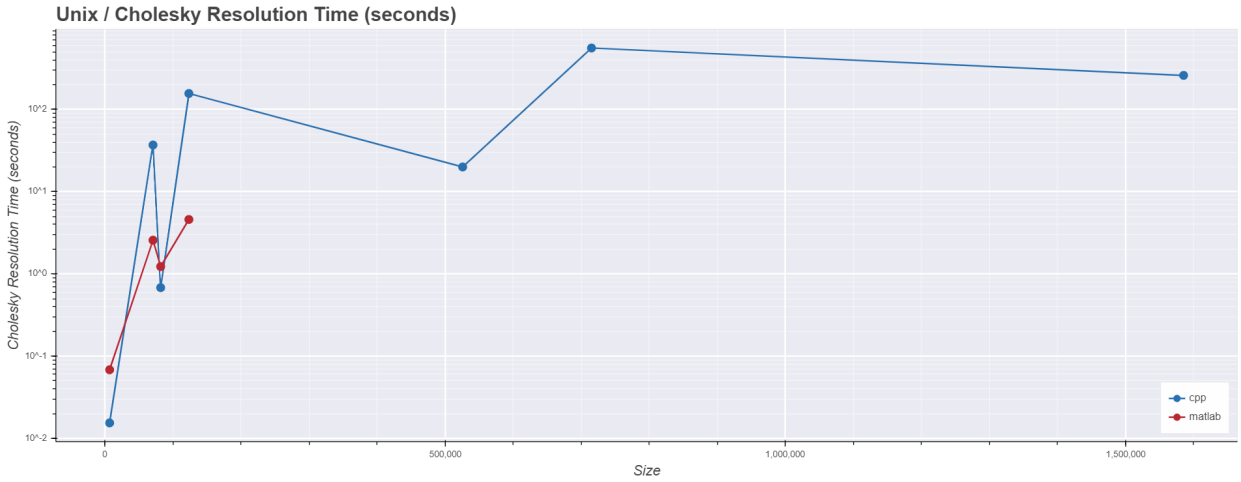
Difatti, tutte le matrici analizzate in MATLAB per le quali non è presente un risultato nel grafico, durante la decomposizione di Cholesky hanno comportato un errore `Out of memory`.



3.2 Linux

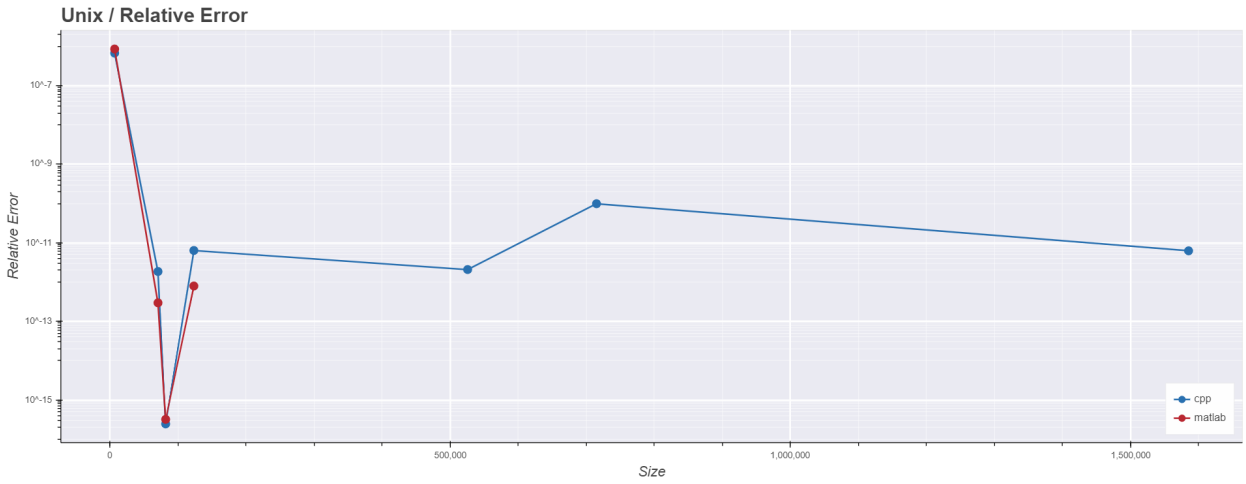
Tempo

Si nota che, all'incremento delle dimensioni della matrice di input, non si rileva un incremento lineare ne costante in termini di tempo (in un punto anche decremento), e non si rilevano differenze eccessive tra le librerie MatLab e C++ (seppure Matlab abbia tempo meno "disperso" della controparte).



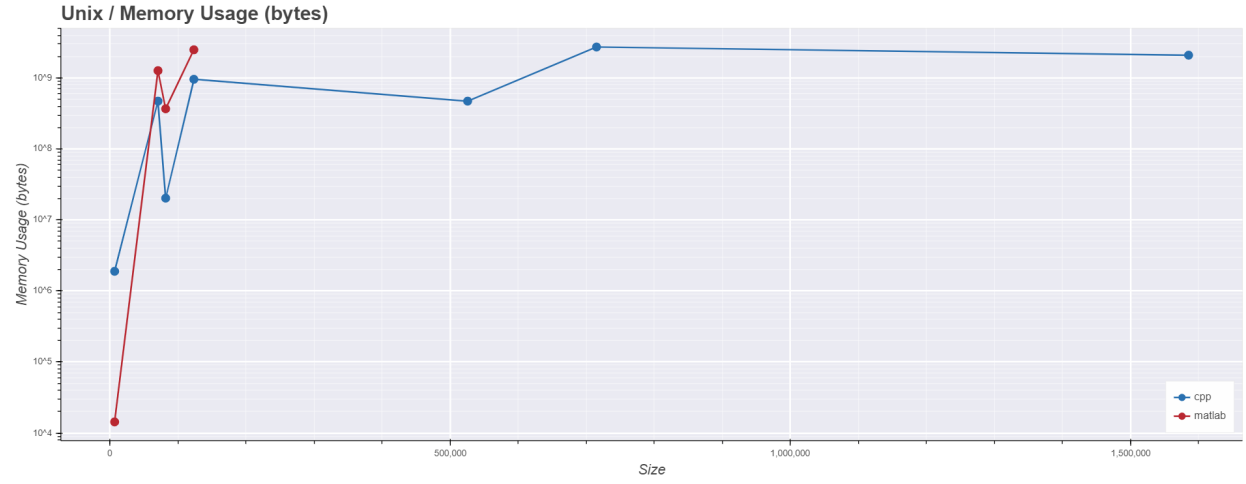
Errore relativo

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.



Memoria

Si nota che la libreria c++ occupa meno memoria rispetto a quella di MatLab sulle matrici più grandi(paragonabile ai risultati windows), e che la memoria occupata non scala linearmente (facendo presumere sia più un fattore di elementi contenuti nelle matrici sparse che di dimensioni?).

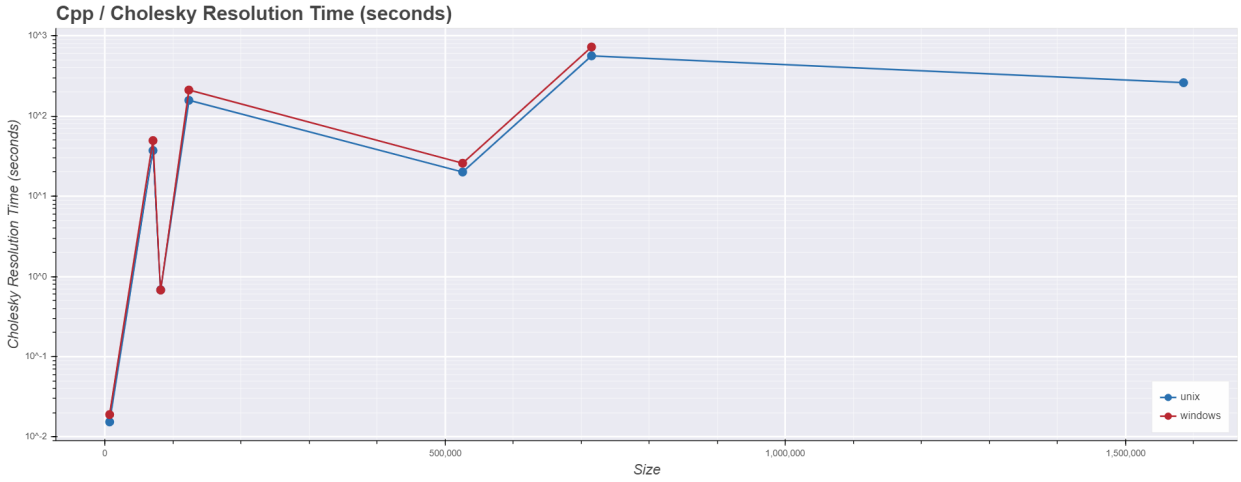


4 Risultati per implementazione

4.1 C++

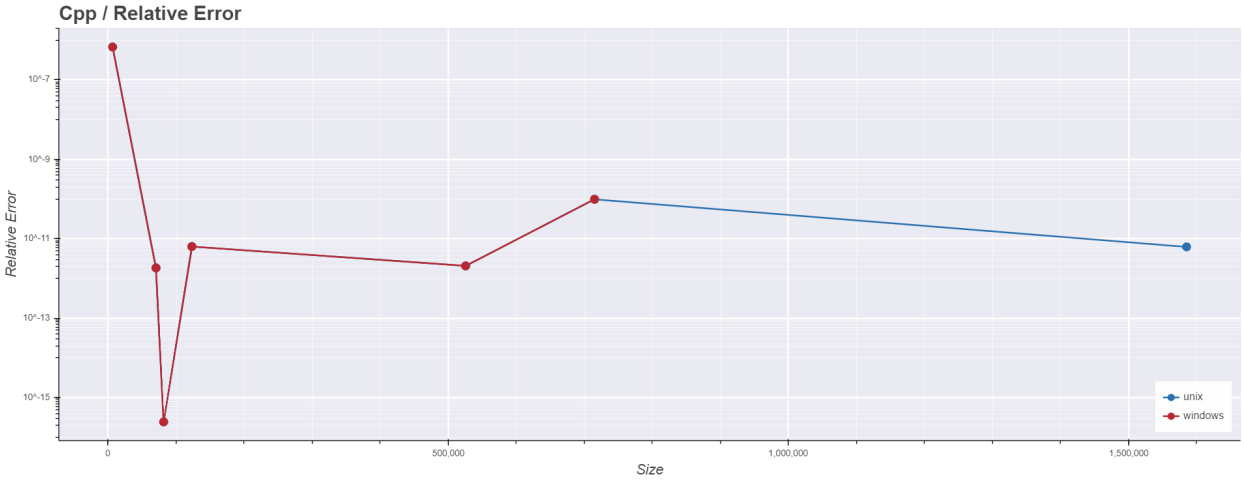
Tempo

TODO



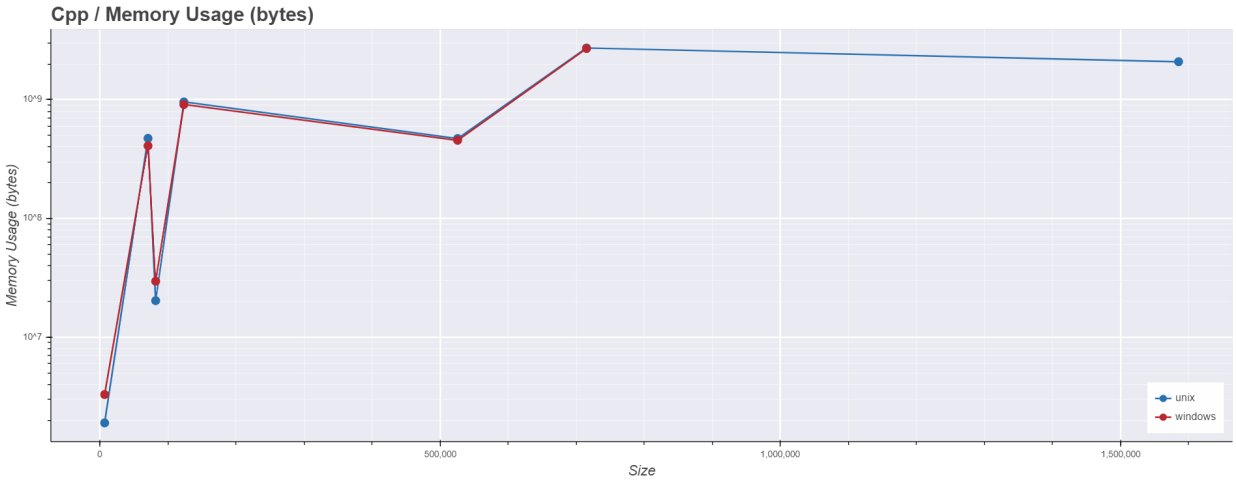
Errore Relativo

TODO



Memoria

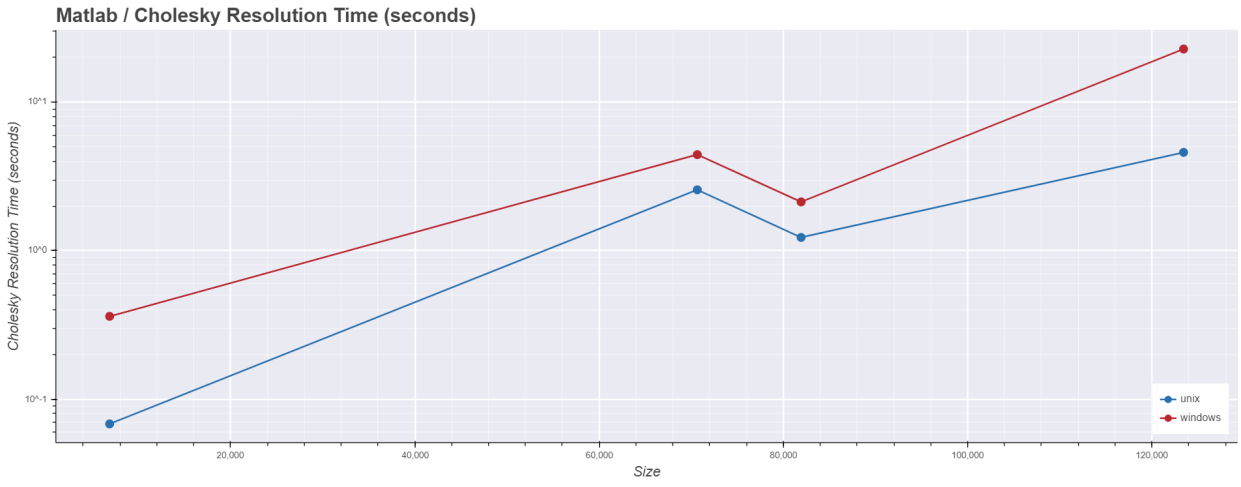
TODO



4.2 Matlab

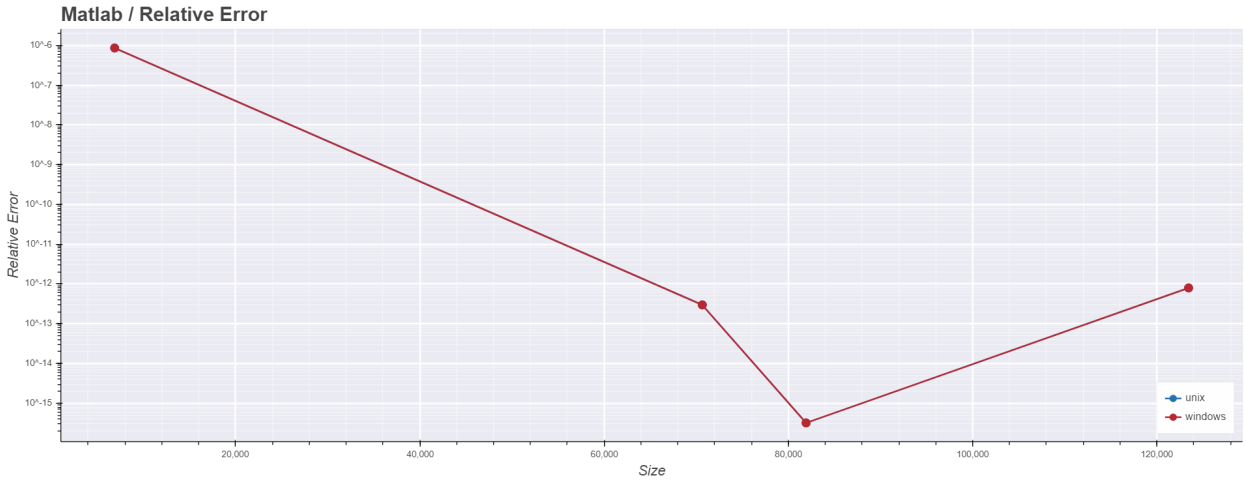
Tempo

TODO



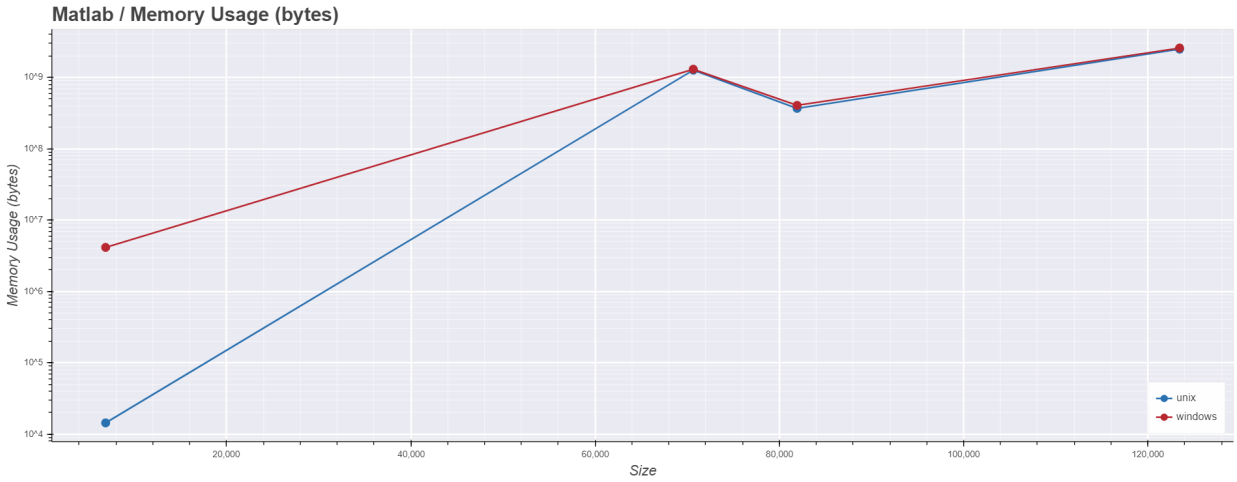
Errore Relativo

TODO



Memoria

TODO



5 Conclusioni

6 Code