Metodi diretti per matrici sparse

Davide Pietrasanta - 844824
Davide Mammarella - 865536
Andrea Premate - 829777

https://github.com/davidepietrasanta/Direct-methods-for-sparse-matrices

Obiettivo

Definire l'alternativa migliore tra ***Software Proprietario*** **(*MATLAB)*** e ***Software Open Source*** per risolvere con metodi diretti sistemi lineari con matrici sparse di grandi dimensioni.

Metriche utilizzate per il confronto:

- Tempo necessario per calcolare la soluzione
- Errore relativo (tra soluzione calcolata e soluzione esatta)
- Memoria necessaria per calcolare la soluzione

Matrici Analizzate

Definite Positive (Cholesky Candidate):

- Hook 1498 - https://sparse.tamu.edu/Janna/Hook_1498
- G3 Circuit - https://sparse.tamu.edu/AMD/G3_circuit
- nd24k - https://sparse.tamu.edu/ND/nd24k
- bundle_adj - https://sparse.tamu.edu/Mazaheri/bundle_adj

Non Definite Positive:

- ifiss mat - https://sparse.tamu.edu/Embree/ifiss_mat
- TSC OPF 1047 - https://sparse.tamu.edu/IPSO/TSC_OPF_1047
- GT01R - https://sparse.tamu.edu/Fluorem/GT01R
- ns3Da - https://sparse.tamu.edu/FEMLAB/ns3Dahttps://sparse.tamu.edu/FEMLAB/ns3Da

Commenti sulle matrici (valori MATLAB)

- **bundle_adj**:
  - Il CONDEST(1-norm condition num estimate) è 6.1180e+15, alto -> errore relativo maggiore(1.1e-5).
  - 513,351x513,351 e 20,207,907 elementi non nulli: rapporto elem non nulli e n^2 risulta 7,7e-5 -> abbastanza sparsa.
  - Il fill in risulta contenuto dato che non c'è un significativo aumento della memoria*.
- **G3_circuit**:
  - Il CONDEST è 2.2384e+07, nella media -> l'errore relativo nella media(3.6e-12).
  - 1,585,478x1,585,478 e 7,660,826 elementi non nulli: rapporto elem non nulli e n^2 risulta 3e-6 -> molto sparsa.
  - Il fill in risulta elevato dato che c'è un significativo aumento della memoria(x15 rispetto alla dim della matrice).

*dal momento che MATLAB lavora in parallelo l'aumento della memoria ed il fill-in potrebbero non essere così strettamente collegati poichè l'aumento di memoria potrebbe essere causato semplicemente dal parallelismo

Commenti sulle matrici (valori MATLAB)

- **ifiss_mat**:
  - Il CONDEST è 8.0858e+04, contenuto -> errore relativo basso(3.3e-14).
  - 96,307x96,307 e 3,599,932 el non nulli: rapporto elem non nulli e n^2 risulta 3,9e-4 -> abbastanza sparsa.
  - Il fill in risulta alto dato che c'è un significativo aumento della memoria(x15 dim della matrice).
- **nd24k**:
  - Il CONDEST è 2.5226e+07, nella norma -> errore relativo non così alto(4.5e-11).
  - 72,000x72,000 e 28,715,634 el non nulli: rapporto elem non nulli e n^2 risulta 5.5e-3 -> poco sparsa.
  - Il fill in risulta molto elevato dato che c'è un significativo aumento della memoria(x18 dim della matrice).
- **ns3Da**:
  - Il CONDEST è 9.1164e+03, il più piccolo -> errore relativo più piccolo(9.3e-16).
  - 20,414x20,414 righe e colonne e 1,679,599 el non nulli: rapporto elem non nulli e n^2 risulta 4e-3 -> poco sparsa.
  - Il fill in risulta abbastanza elevato dato che c'è un significativo aumento della memoria(x8 rispetto alla dimensione della matrice).

Commenti sulle matrici (valori MATLAB)

- **TSC_OPF_1047**:
  - Il CONDEST è 2.3256e+10, alto -> errore relativo tra i più alti(2.4e-11).
  - 8,140x8,140 e 2,012,833 elementi non nulli: rapporto elem non nulli e n^2 risulta 3e-2 -> poco sparsa rispetto alle altre.
  - Il fill in risulta contenuto dato che non c'è un significativo aumento della memoria.
- **GT01R**:
  - Il CONDEST è 6.5752e+07, nella media -> l'errore relativo è basso (3.8e-15) probabilmente per dimensioni ridotte.
  - 7,980x7,980 e 430,909 elementi non nulli: rapporto elem non nulli e n^2 risulta 6.7e-3 -> poco sparsa rispetto alle altre.
  - Il fill in risulta abbastanza elevato dato che c'è un significativo aumento della memoria(x7 dim della matrice).
- **Hook_1498**:
  - calcolo CONDEST non riuscito(>16GB necessari), probabilmente basso per l'errore relativo molto basso(7.4e-14).
  - 1,498,023x1,498,023 e 59,374,451 elementi non nulli: rapporto elem non nulli e n^2 risulta 2.6e-5 -> piuttosto sparsa.
  - Il fill in risulta piuttosto elevato dato che c'è un significativo aumento della memoria (x13 rispetto alla dim della matrice).

Ambienti di Programmazione

Abbiamo voluto analizzare le matrici proposte usando diversi linguaggi di programmazione:

- **MATLAB**
- **Python**
- **C++**
- **R**

Scegliendo linguaggi usati nell'ambito di Machine Learning e del Data Science.

Abbiamo poi confrontato i tempi su due Sistemi Operativi:

- **Windows 10**
- **Ubuntu 20.04**

Usando una Virtual Machine, tramite Virtual Box, con ***16GB di RAM*** e un processore ***AMD Ryzen 7 3700X***.

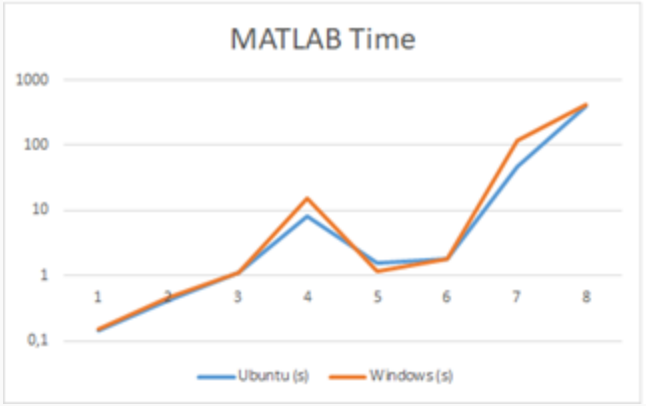
Considerazioni

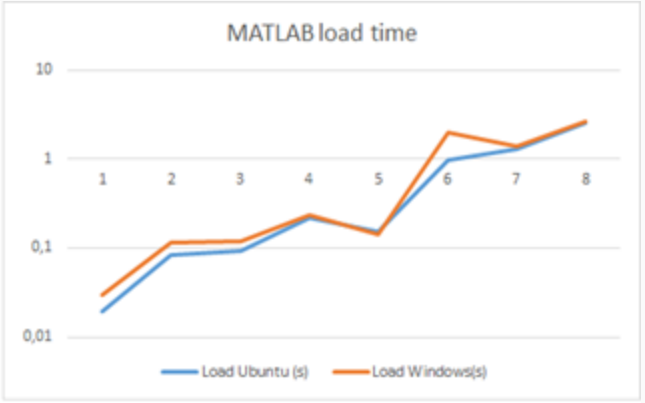
- **Matrici sparse** possono essere memorizzate in modo compatto ovvero tenendo conto solo degli elementi diversi da zero
- Decomposizione di ***Cholesky*** risulta essere più performante della fattorizzazione ***PA=LU*** ma é applicabile solo a determinate condizioni:
  - Matrice Simmetrica
  - Matrice Definita Positiva
- Un algoritmo efficiente dovrebbe verificare autonomamente queste condizioni ed applicare la fattorizzazione piú consona
  - ***MATLAB*** riesce nell'intento
  - ***Librerie Open Source*** non sempre riescono nell'intento (Soluzione "manuale": eseguire la fattorizzazione Cholesky ed caso di errori identificare la matrice come non candidata possibile per Cholesky)

MATLAB

- **Non necessita l'installazione di librerie** poiché gli operatori built-in permettono la risoluzione della matrice in modo ottimale
- Tramite **un'unica funzione** verifica la struttura di A e calcola la soluzione utilizzando la fattorizzazione piú opportuna
- Ha **risolto tutte le matrici** proposte
- Presenta **interfaccia user friendly** e **documentazione estesa**
- **Necessita una licenza** che costa circa 800 euro annui o 2000 euro a tempo indeterminato

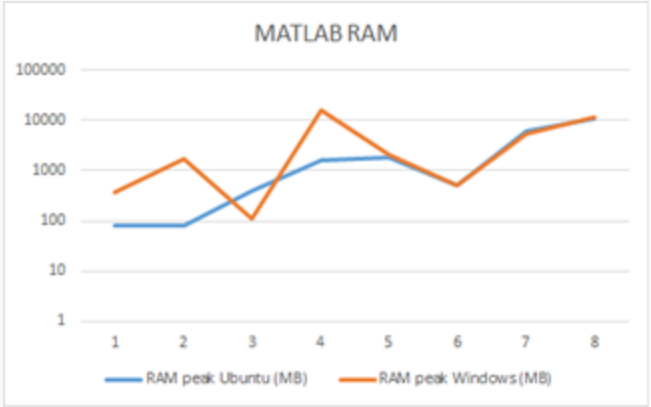
MATLAB - Misure

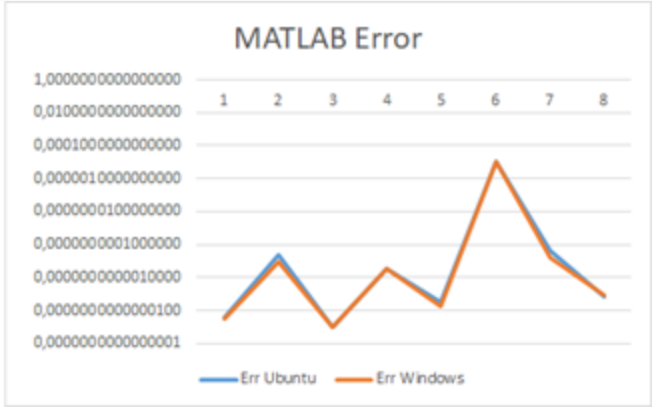




Come si vede dalle immagini, in termini di tempo, MATLAB tende ad avere migliori performance su Ubuntu rispetto che su Windows.

MATLAB - Misure





Come si vede dalle immagini, in termini di memoria, MATLAB, in media, tende ad avere migliori performance su Ubuntu rispetto che su Windows.

Non ci sono particolari differenze invece per quanto riguardo l' errore relativo.

Python

Per Python abbiamo usato due librerie ***scipy*** e ***scikit-sparse***.

La prima libreria presenta la funzione ***spsolve*** che risolte sistemi lineare usando la fattorizzazione PA=LU.

La seconda libreria presenta la funzione ***cholesky*** che risolte sistemi lineare usando la fattorizzazione di Cholesky.

Per Python bisogna dunque installare due librerie diverse e non esiste un'unica funzione per risolvere tutti i tipi di sistemi lineari nel modo più efficiente. È però molto semplice tenere traccia del tempo e della memoria usati durante il calcolo.

Purtroppo nessuna delle due librerie è riuscita a risolvere la matrice "Hook 1498" e si pensa sia necessaria altra RAM, per un totale di almeno 64GB circa.

Python - scipy

La libreria ***scipy*** è ben documentata, piena di esempi e tutt'oggi mantenuta e aggiornata. Facile da Installare sia su Ubuntu che su Windows, tramite il comando **pip install scipy** e pesa solo 23.7 MB.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.spsolve.html#scipy.sparse.linalg.spsolve

Python - scikit-sparse

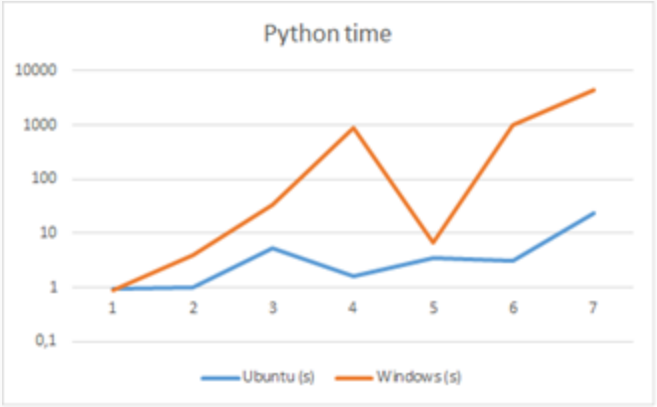
La libreria ***scikit-sparse*** è ben documentata, piena di esempi e tutt'oggi mantenuta e aggiornata.

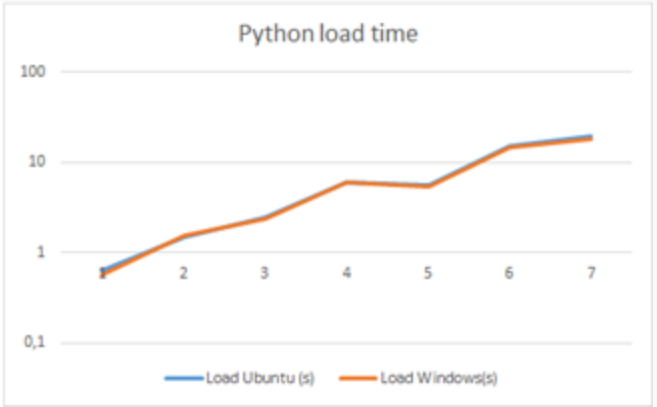
Facile da installare su Ubuntu, tramite il comando **conda install scikit-sparse**, e leggera 441 MB.

Difficile da installare su Windows, dove richiede l'installazione di Visual Studio C++ e una serie di passaggi e pesa più di 2.5 GB.

https://scikit-sparse.readthedocs.io/en/latest/cholmod.html

Python - Misure

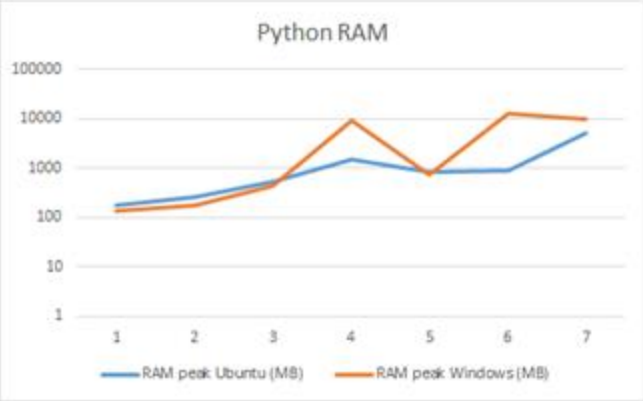


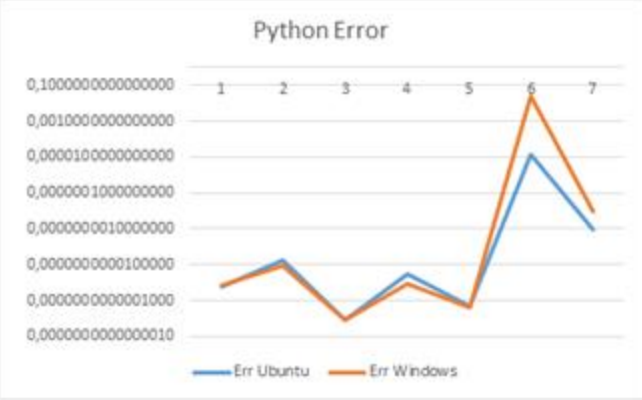


Come si vede dalle immagini, in termini di tempo, Python tende ad avere migliori performance su Ubuntu rispetto che su Windows.

Si nota anche come il tempo di caricamento delle matrici non differisca tra Ubuntu e Windows.

Python - Misure





Come si vede dalle immagini, in termini di memoria e errore relativo, Python, in media, tende ad avere migliori performance su Ubuntu rispetto che su Windows.

C++ Eigen

Per C++ abbiamo usato la libreria ***Eigen***.

La funzione ***SparseLU*** risolve sistemi lineari utilizzando la fattorizzazione PA=LU. Nella documentazione è indicato che tale funzione è ottimizzata per grandi e piccoli problemi con pattern irregolari. Per minimizzare il fill-in viene applicata una permutazione delle colonne.

La funzione ***SimplicialLDLT*** risolve sistemi lineari utilizzando la fattorizzazione di Cholesky nella versione LDLT. Tale scomposizione si ottiene con algoritmi del tutto simili alla scomposizione LLT ma ha il vantaggio di non dover calcolare le radici. Nella documentazione è raccomandata rispetto alla ***SimplicialLLT***, specie in problemi molto sparsi. Per ridurre il fill-in, in ogni caso viene eseguita una permutazione simmetrica P prima della fattorizzazione, in modo tale che la matrice fattorizzata sia PAP^-1.

C++ Eigen

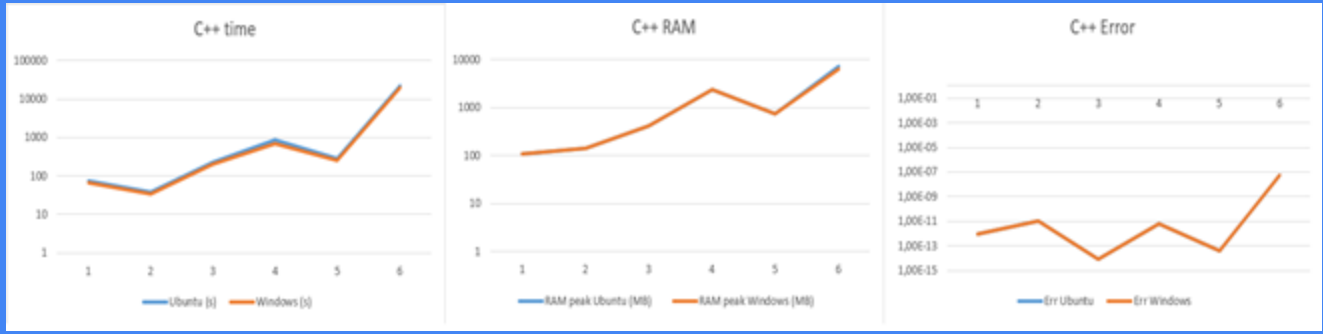
Abbiamo quindi costruito un'unica funzione per risolvere i sistemi lineari scegliendo automaticamente la decomposizione: viene provata la decomposizione di Cholesky e, se fallisce, viene fatta la LU.

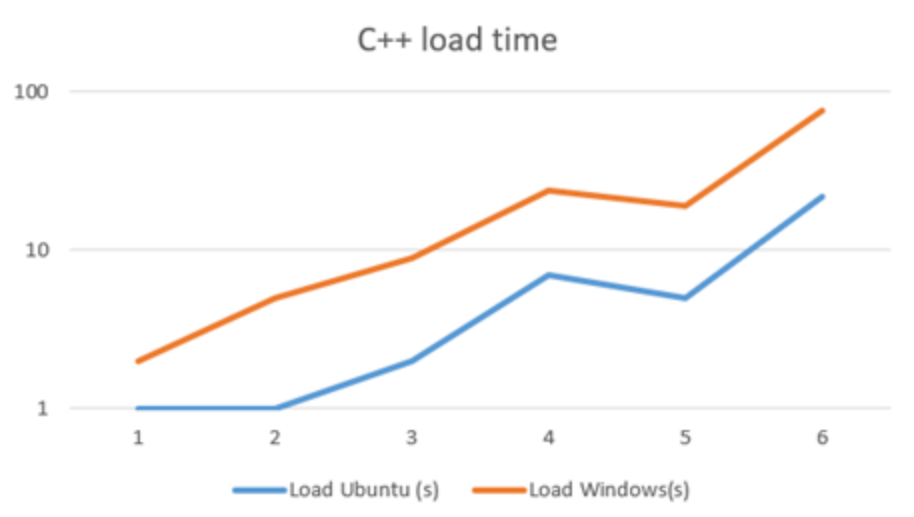
In C++ è risultato semplice tenere traccia del tempo e della memoria usati durante il calcolo.

Purtroppo la libreria non è riuscita a risolvere la matrice "Hook 1498" e la matrice "bundle_adj". In entrambi i casi il problema è probabilmente legato alla memoria. Su Ubuntu infatti viene restituito l'errore "segmentation fault (core dumped)".

Per utilizzare la libreria è necessario scaricare il folder e includere il percorso in fase di compilazione (e ovviamente inserire gli #include). La libreria risulta discretamente documentata ma per caricare le matrici da file mtx è stato necessario utilizzare una porzione di libreria non più supportata che ha creato problemi: le matrici simmetriche venivano caricate solo per la metà triangolare superiore.

C++ - Misure





Le performance riscontrate sono state del tutto simili su Ubuntu e Windows. Le uniche differenze significative sono sul tempo di caricamento delle matrici. Approssimativamente su Ubuntu si impiega ¼ del tempo impiegato su Windows. La scala logaritmica mette in mostra tale costante moltiplicativa.

R - Matrix

- **R:** ecosistema open-source pensato per l'analisi statistica dei dati che puó essere esteso al calcolo scientifico tramite l'utilizzo di vari **packages facili da includere**

- Package scelto: ***Matrix***
  - Estende le classi base delle matrici fornendo classi per **matrici sparse** e **dense**
  - ***readMM*** permette di caricare le matrici in formato sparso "dgCMatrix", ovvero in formato CSC (Compressed Sparse Column)
  - ***solve*** permette la risoluzione di sistemi lineari del tipo **Ax=b** tramite la fattorizzazione piú opportuna (Cholesky od LU)
- Molto ben documentato (219 pagine) e costantemente mantenuto (ultima release: 01/06/2021)
  - https://cran.r-project.org/web/packages/Matrix/Matrix.pdf

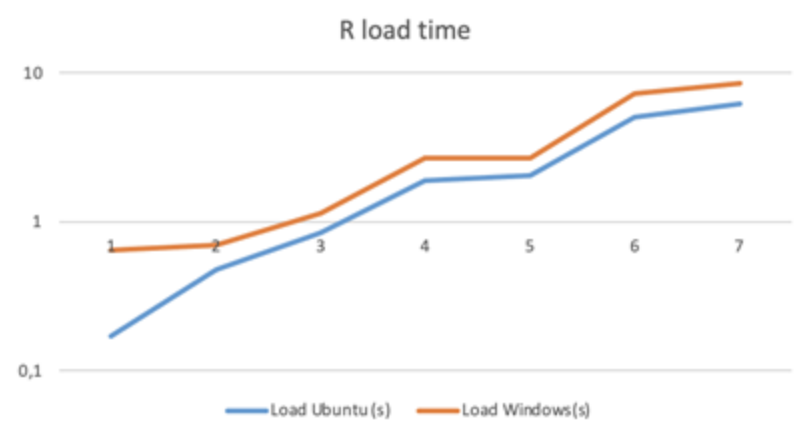
R - Matrix - solve()

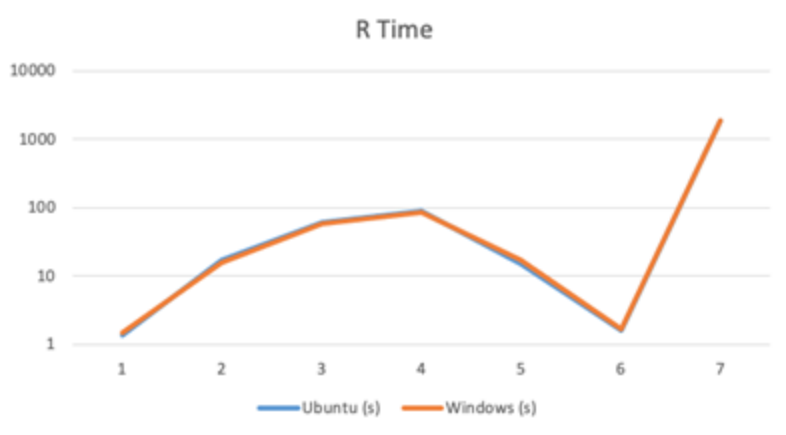
- Metodo utilizzato per la risoluzione di sistemi lineari del tipo **Ax=b** dove:
  - A deve essere una matrice quadrata (densa o sparsa)
  - b deve essere un vettore o una matrice (densa o sparsa)
- **signature (sparse= TRUE, tol =.Machine$double.eps)**: controlla la struttura di A e calcola la soluzione tramite la fattorizzazione piú opportuna:
  - **A simmetrica definita positiva**: fattorizzazione di Cholesky
  - **A simmetrica non definita positiva**: fattorizzazione LU
  - **A non simmetrica**: fattorizzazione LU

R

- Durante le analisi sono stati implementati anche i packages ***SparseM*** ed ***SPAM***
  - Il primo puó essere utilizzato come aggiunta di ***Matrix*** ma ai fini del progetto non vi è nessuna miglioria, risultando in un aumento della complessitá del codice
  - Il secondo seppur essendo piú rapido nella risoluzione, necessita una discriminazione a priori per la fattorizzazione e presenta tempi di caricamento delle matrici di gran lunga superiori
- Package ***Matrix*** risulta semplice da installare, ben documentato e ben mantenuto
- Package ***Matrix*** presenta un'unica funzione per risolvere tutti i tipi di sistemi lineari nel modo più efficiente, a patto di specifiche accortezze
- Package ***bench*** ci ha permesso di raccogliere metriche sul tempo e la memoria utilizzata durante il calcolo in maniera semplice
- Purtroppo ***Matrix*** non è riuscito a risolvere la matrice "Hook 1498" e si pensa sia necessaria altra RAM, per un totale di 64GB circa

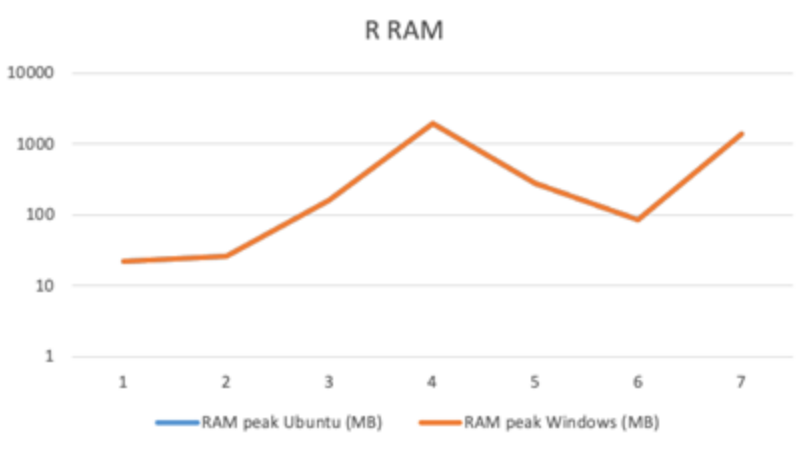
R - Misure (Tempi)

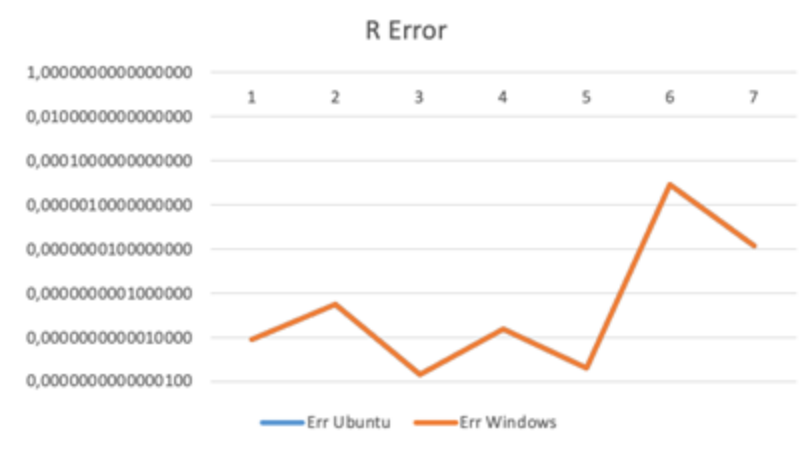




La differenza piú significativa si ha sul tempo di caricamento, avendo Windows come vincente, mentre i tempi di risoluzione risultano pressoché identici

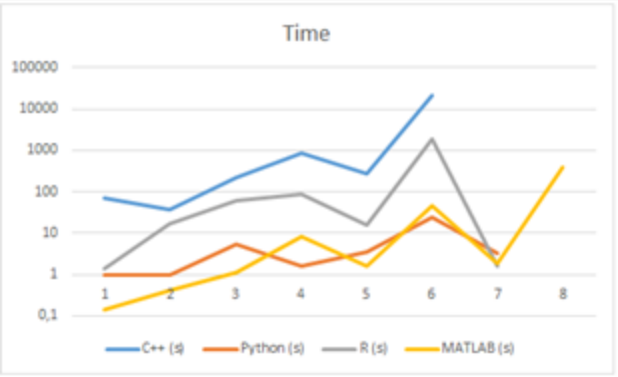
R - Misure (RAM ed Errore)

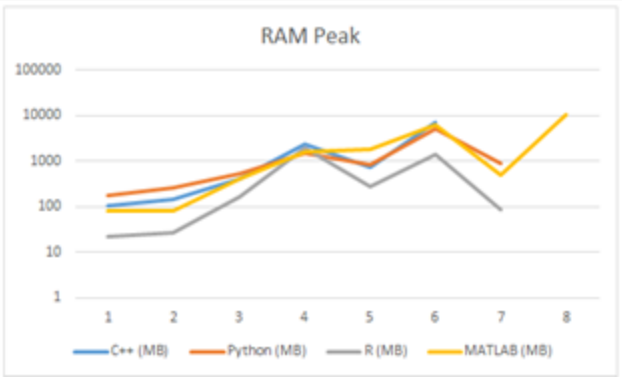


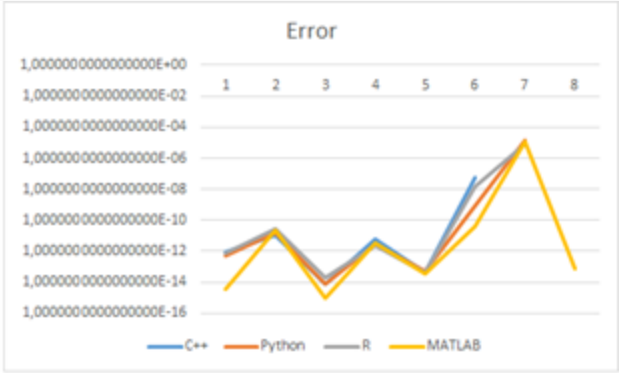


I sistemi operativi non hanno influenza su RAM ed Errore

Misure a confronto







Ecco le migliori misure a confronto, usando il sistema operativo migliore per ogni linguaggio di programmazione.

MATLAB - Voto

- Installazione Windows: 10/10
- Installazione Ubuntu: 10/10
- Documentazione: 10/10
- Mantenuta e Aggiornata: SI
- SO Migliore: Ubuntu

MATLAB - Classifica

Ubuntu

- Velocità caricamento: Primo
- Velocità elaborazione: Primo
- Memoria: Primo
- Errore relativo: Secondo

Windows

- Velocità caricamento: Primo
- Velocità elaborazione: Primo
- Memoria: Primo
- Errore relativo: Secondo

R - Voto Libreria

Matrix

- Installazione: 10/10
- Documentazione: 10/10
- Mantenuta e Aggiornata: SI
- SO Migliore: Equivalenti

R - Classifica

Ubuntu

- Velocità caricamento: Secondo
- Velocità elaborazione: Terzo
- Memoria: Secondo
- Errore relativo: Primo

Windows

- Velocità caricamento: Secondo
- Velocità elaborazione: Secondo
- Memoria: Secondo
- Errore relativo: Primo

Python - Voto Librerie

Scipy

- Installazione Windows: 10/10
- Installazione Ubuntu: 10/10
- Documentazione: 10/10
- Mantenuta e Aggiornata: SI
- SO Migliore: Ubuntu

Scikit-sparse

- Installazione Windows: 4/10
- Installazione Ubuntu: 9/10
- Documentazione: 10/10
- Mantenuta e Aggiornata: SI
- SO Migliore: Ubuntu

Python - Classifica

Ubuntu

- Velocità caricamento: Terzo
- Velocità elaborazione: Secondo
- Memoria: Terzo
- Errore relativo: Terzo

Windows

- Velocità caricamento: Terzo
- Velocità elaborazione: Terzo
- Memoria: Quarto
- Errore relativo: Terzo

C++ Eigen - Voto libreria

Eigen 3.3.9

- Installazione: 8/10
- Documentazione: 7/10
- Mantenuta e Aggiornata: NO (non completamente)
- SO Migliore: Ubuntu (~ equivalenti)

C++ Eigen - Voto

Ubuntu

- Velocità caricamento: Quarto
- Velocità elaborazione: Quarto
- Memoria: Quarto
- Errore relativo: Quarto

Windows

- Velocità caricamento: Quarto
- Velocità elaborazione: Quarto
- Memoria: Terzo
- Errore relativo: Quarto

Classifica finale - performance

Complessivamente le classifiche per i quattro linguaggi rimangono uguali per i due sistemi operativi.

1. MATLAB
2. R
3. Python
4. C++

Conclusioni

Le cose da valutare sono:

- Performance (tempi, RAM, matrici risolte, etc)
- Facilità utilizzo
- Costi
- Problemi da risolvere a livello aziendale

In conclusione:

- Si sceglie MATLAB se l'algebra lineare è parte attiva dall'azienda
- Si sceglie R se i problemi di algebra lineare vengono risolti con sporadicità