

# Progetto Metodi del Calcolo Scientifico

Matteo Colella - 794028

Matteo Costantini - 795125

Dario Gerosa - 793636

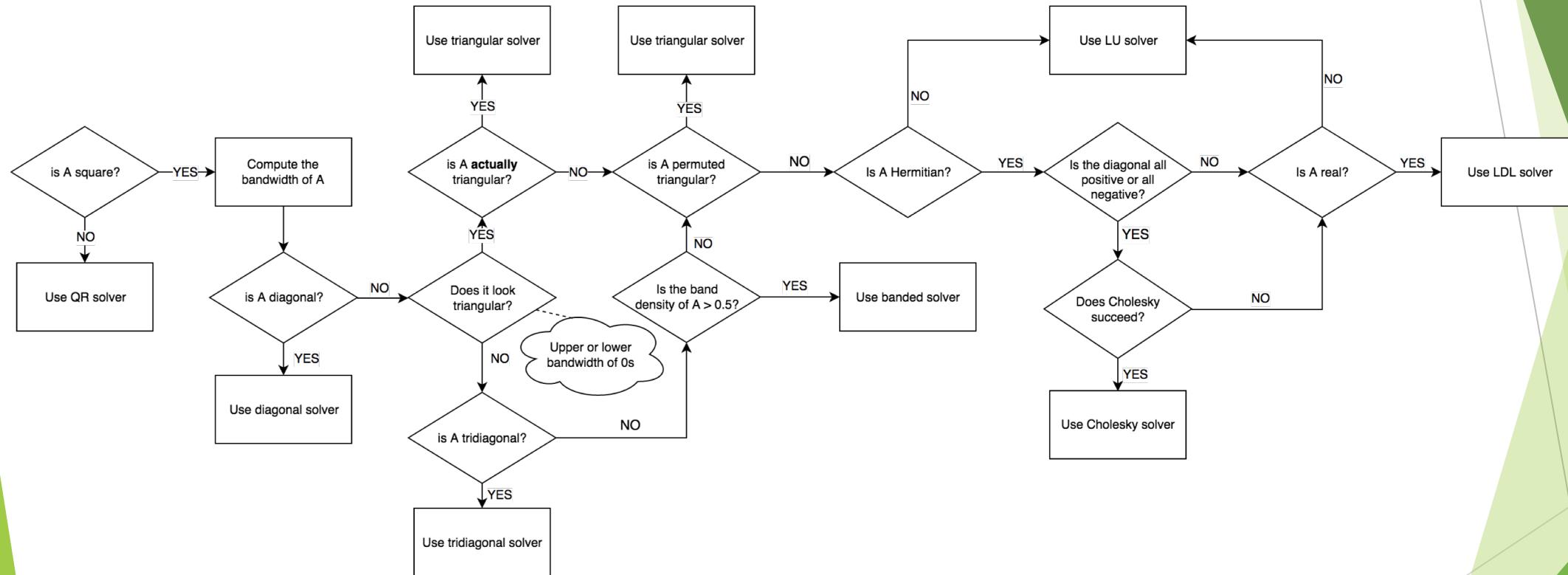
# Obiettivo

- ▶ Confrontare due diverse soluzioni per la risoluzione di sistemi lineari con matrici sparse su due diversi sistemi operativi (Windows 10, Ubuntu Linux).
  - ▶ MATLAB: funzione standard `mldivide`
  - ▶ Python: Libreria `scipy`
- ▶ Parametri osservati:
  - ▶ Errore relativo della soluzione calcolata rispetto alla soluzione esatta
  - ▶ Memoria utilizzata durante la risoluzione del sistema
  - ▶ Tempi di calcolo
  - ▶ Chiarezza della documentazione
  - ▶ Semplicità d'uso

# MATLAB

- ▶ In MATLAB è disponibile una funzione standard per la risoluzione di sistemi lineari sparsi e non: `mldivide` (o l'operatore `\`). La documentazione relativa alla funzione è reperibile al seguente indirizzo: <https://it.mathworks.com/help/matlab/ref/mldivide.html>
- ▶ L'algoritmo di risoluzione da utilizzare viene scelto in base alle caratteristiche della matrice passata in input secondo il seguente schema

# MATLAB



# Python scipy

- ▶ Ecosistema open-source nato nel 2001 e composto da diverse librerie (NumPy, SciPy library, Matplotlib, Ipython, Sympy, Pandas)
- ▶ Utilizzato in matematica, scienze e ingegneria
- ▶ Composta da diversi package che offrono supporto per: clustering, trasformata di Fourier, interpolazione, algebra lineare, matrici sparse, programmazione lineare, trattamento di segnali ...
- ▶ Attivamente mantenuta (ultima release 5/10/18) e documentata (<https://www.scipy.org/docs.html>)
  - ▶ Sorgente: <https://github.com/scipy/scipy>

# Python scipy

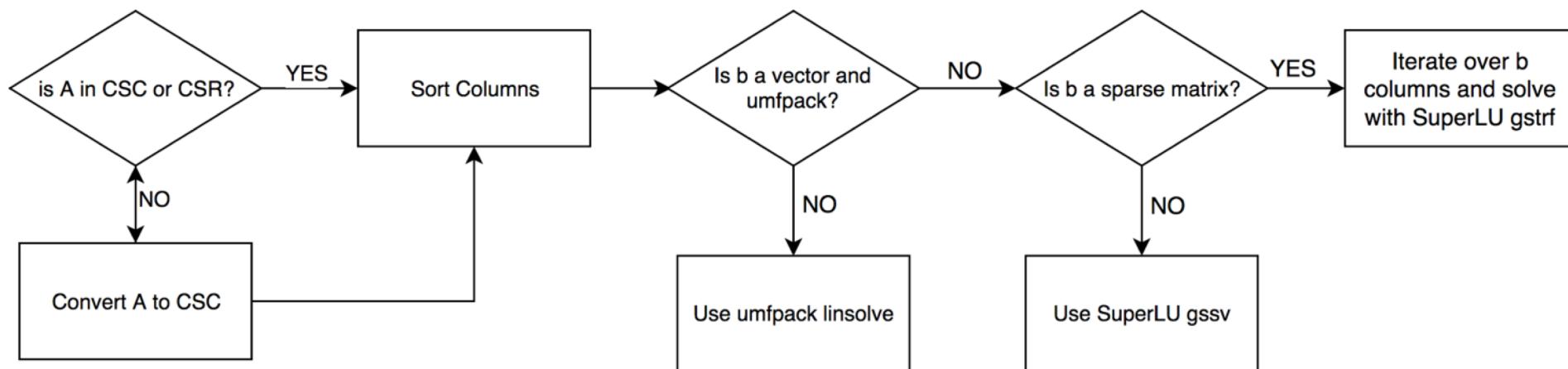
- ▶ In particolare abbiamo utilizzato:
  - ▶ `scipy.io`: Lettura delle matrici in formato .mtx
  - ▶ `scipy.sparse.linalg`:
    - ▶ `spsolve()`: Risoluzione di sistemi lineari sparsi
    - ▶ `norm()`: Calcolo della norma per matrici sparse
  - ▶ `numpy`: indirettamente. Utilizzato per la rappresentazione delle strutture dati

# Spsolve

- ▶ Permette di risolvere sistemi lineari del tipo  $Ax=b$  dove  $A$  è una matrice sparsa e  $b$  può essere una matrice. In questo caso risolve un sistema lineare per ogni colonna di  $b$ .
- ▶ Se la matrice  $A$  non è già in formato *CSC* o *CSR* (*Compressed Sparse Columns/Rows*) viene automaticamente convertita in formato *CSC*
- ▶ È possibile specificare la politica di riordinamento delle colonne utilizzata per minimizzare il *fill-in*.
- ▶ È possibile specificare la libreria da utilizzare per la risoluzione tra SuperLU e umfpack (default)
- ▶ Nel caso in cui  $x$  e  $b$  siano matrici ( $AX=B$ ), la funzione assume che la soluzione  $X$  sia a sua volta sparsa. Se così non fosse risulta più efficiente utilizzare la versione del modulo `scipy.linalg` per la risoluzione di matrici non sparse.

# Spsolve

- ▶ Algoritmo molto più semplice rispetto a quello utilizzato da MATLAB



# Parametri analizzati

- ▶ Errore relativo:

- ▶ `relative_error = norm(x - xe, 2) / norm(xe, 2)`

Dove  $xe$  è la soluzione esatta del sistema, ovvero il vettore avente tutte le componenti pari a 1 e `norm` è la norma 2 del vettore.

- ▶ Tempo di esecuzione

- ▶ Python: Modulo standard `datetime`
  - ▶ MATLAB: `tic; toc;`

# Parametri analizzati

- ▶ Memoria utilizzata per la risoluzione del sistema:
  - ▶ Metrica intrinsecamente imprecisa
  - ▶ Stessa metodologia sia per MATLAB che per Python
  - ▶ Script Python che utilizza la libreria psutil per ottenere dal sistema operativo la memoria utilizzata da un processo
  - ▶ Avendo poche matrici da analizzare non abbiamo automatizzato questa fase. Lo script deve essere quindi lanciato manualmente

# Sorgente - Memoria utilizzata

```
import sys, psutil, time

# Frequenza fetch
delay = 1 if len(sys.argv) < 3 else float(sys.argv[1])
# PID del processo
pid = int(sys.argv[1]) if len(sys.argv) == 2 else
int(sys.argv[2])

proc = psutil.Process(pid)
# Memoria massima utilizzata
max_mem = 0
while True:
    time.sleep(delay)
    curr_mem = proc.memory_info()[0]
    maxmm = curr_mem if curr_mem > max_mem else max_mem
    print(f"{curr_mem} - {max_mem}")
```

# Sorgente - Python

```
import scipy, scipy.io
from datetime import datetime
from scipy.sparse.linalg import norm, spsolve
from scipy.linalg import norm

A = scipy.io.mmread(path).tocsc() # Matrice da risolvere

xe = scipy.ones(n_rows)           # Soluzione esatta
b = A * xe                      # Vettore termini noti

start_time = datetime.now()
x = spsolve(A, b, use_umfpack=use_umfpack)
t = datetime.now() - start_time
relative_error = norm(x - xe, 2) / norm(xe, 2)
```

# Sorgente - MATLAB

```
mat = load(path);
solve_matrix(mat);

function solve_matrix(mat)
    A = mat.Problem.A;

    xe = ones(length(A), 1);
    b = A * xe;

    x = A \ b;
    t = toc;
    relative_error = norm(x - xe) / norm(xe);
end
```

# Macchina utilizzata

- ▶ ASUS Zenbook UX330
  - ▶ CPU: Intel i7 7500U @ 2.70 GHz
  - ▶ RAM: 8GB DDR3
  - ▶ Memoria fisica: SSD M.2 512GB
  - ▶ Sistemi operativi:
    - ▶ Windows 10
    - ▶ Ubuntu Linux 16.04

# Matrici Testate

## ► Non Definite Positive

- ▶ PR02R (161070)
- ▶ ex19 (12005)
- ▶ graham1 (9035)
- ▶ kim2 (456976)
- ▶ raefsky3 (21200)
- ▶ torso1 (116158)
- ▶ torso3 (259156)
- ▶ water\_tank (60740)

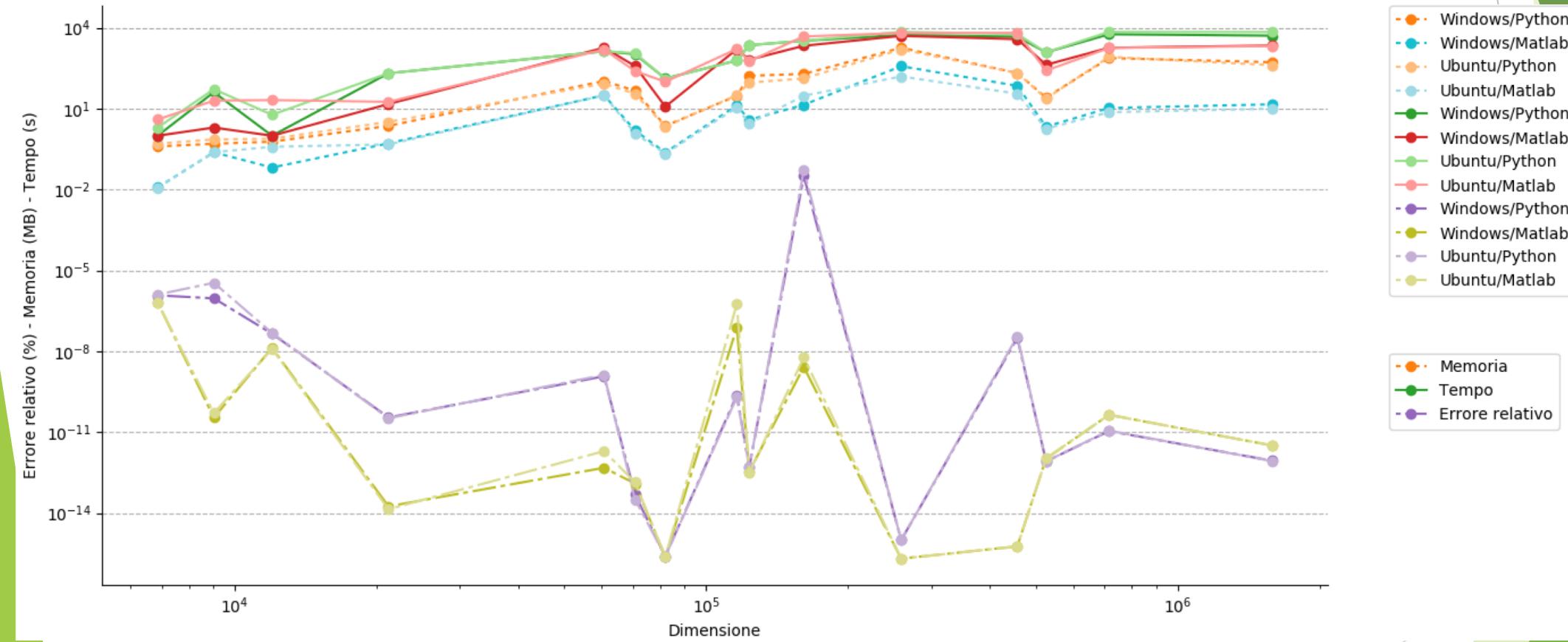
## ► Definite Positive

- ▶ G3\_circuit (1585478)
- ▶ apache2 (715176)
- ▶ cfd1 (70656)
- ▶ cfd2 (123440)
- ▶ ex15 (6867)
- ▶ parabolic\_fem (525825)
- ▶ shallow\_water1 (82920)

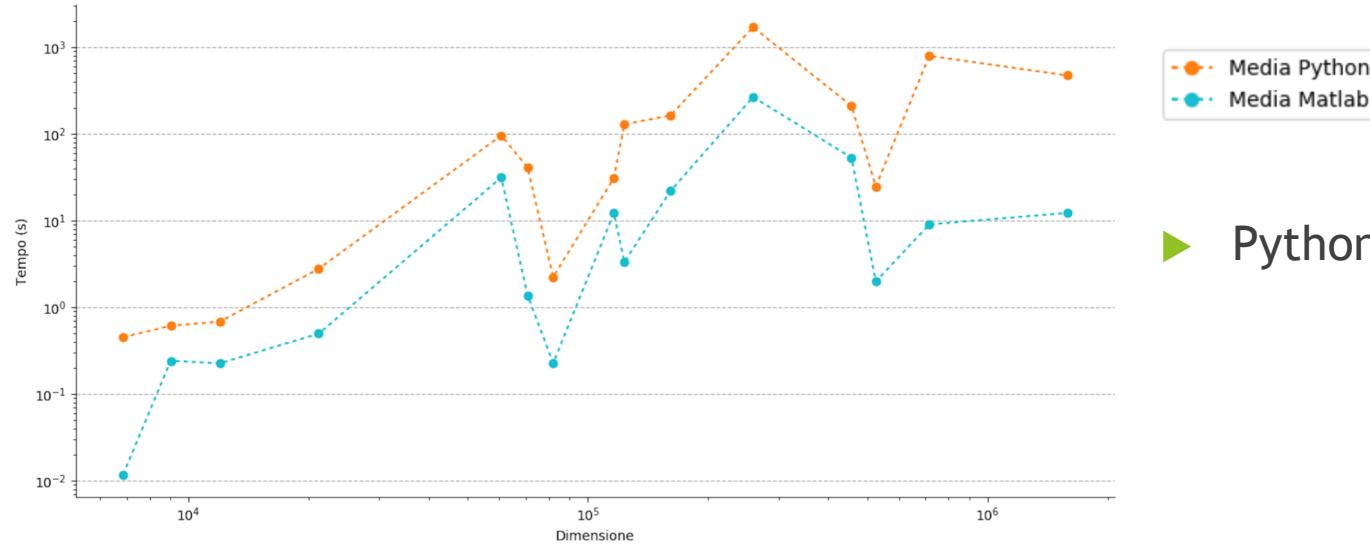
# Problemi Riscontrati

- ▶ Inizialmente abbiamo provato ad utilizzare la libreria umfpack su `scipy`. Pur essendo di gran lunga più veloce dell'alternativa SuperLU, ha dato problemi con matrici anche di ridotte dimensioni. L'errore è dovuto alla versione dell'algoritmo che viene utilizzata che non permette di allocare più di 4GB di memoria. L'errore è già segnalato ai gestori del progetto ma non è ancora disponibile un fix.
- ▶ La documentazione della funzione `spsolve` di `scipy`, seppur sufficiente per poter utilizzare quest'ultima, non è esaustiva e per sapere quali algoritmi utilizza è stato necessario consultare il sorgente.
- ▶ Non siamo riusciti a trovare alcuna documentazione riguardante la libreria SuperLU e le funzioni che vengono richiamate da `scipy` per risolvere.

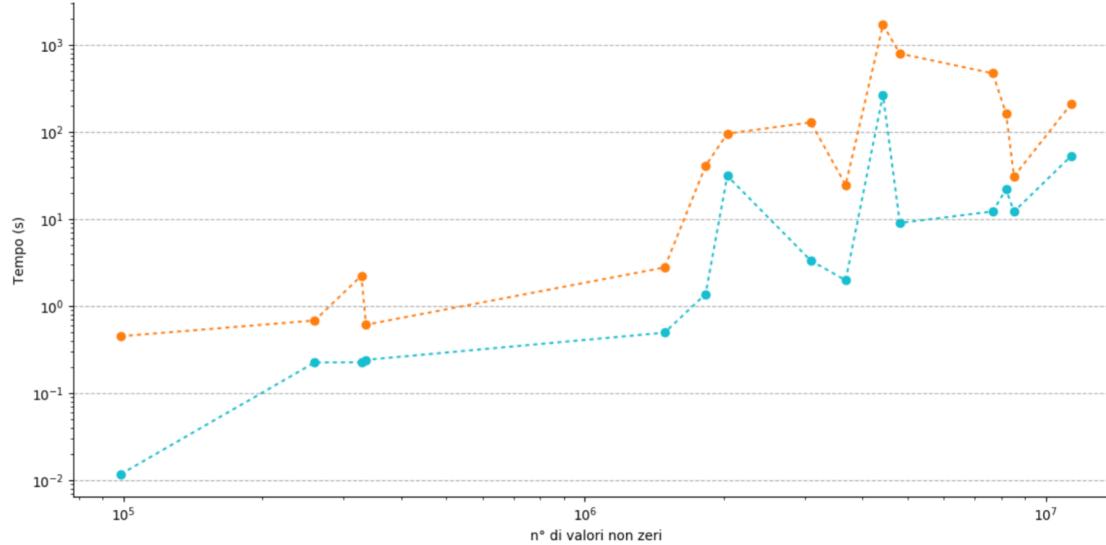
# Overview Risultati



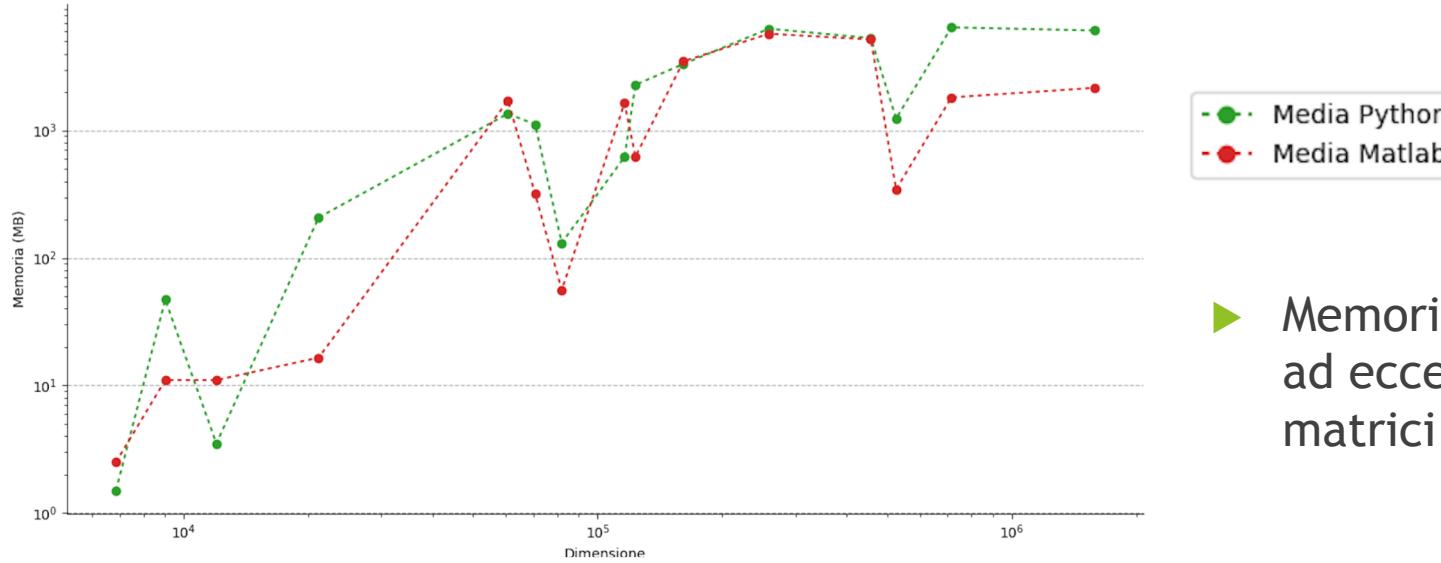
# Risultati MATLAB vs Python scipy



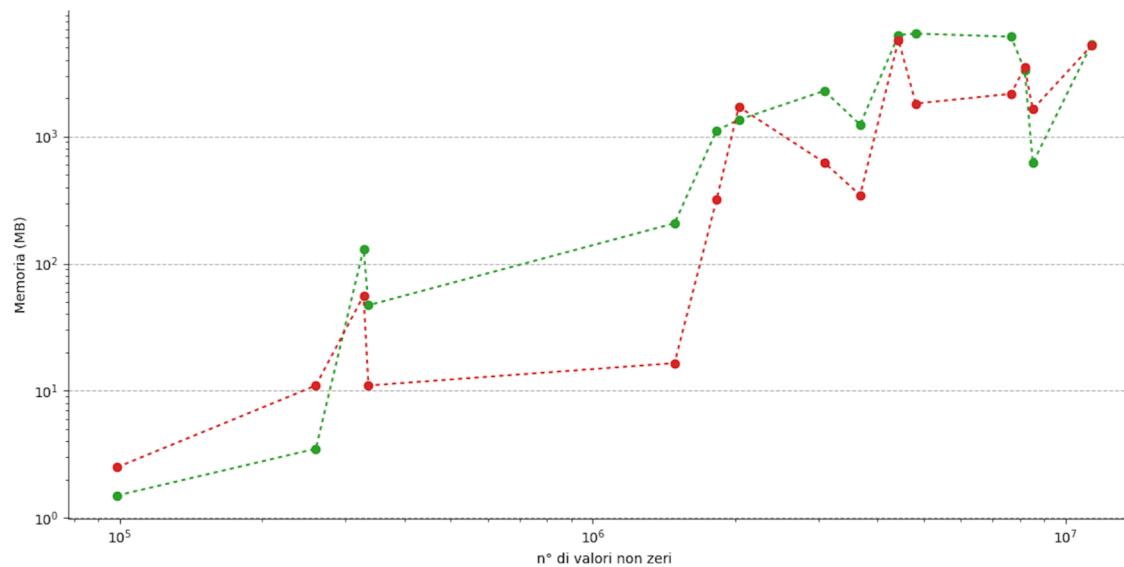
▶ Python ~18 volte più lento!



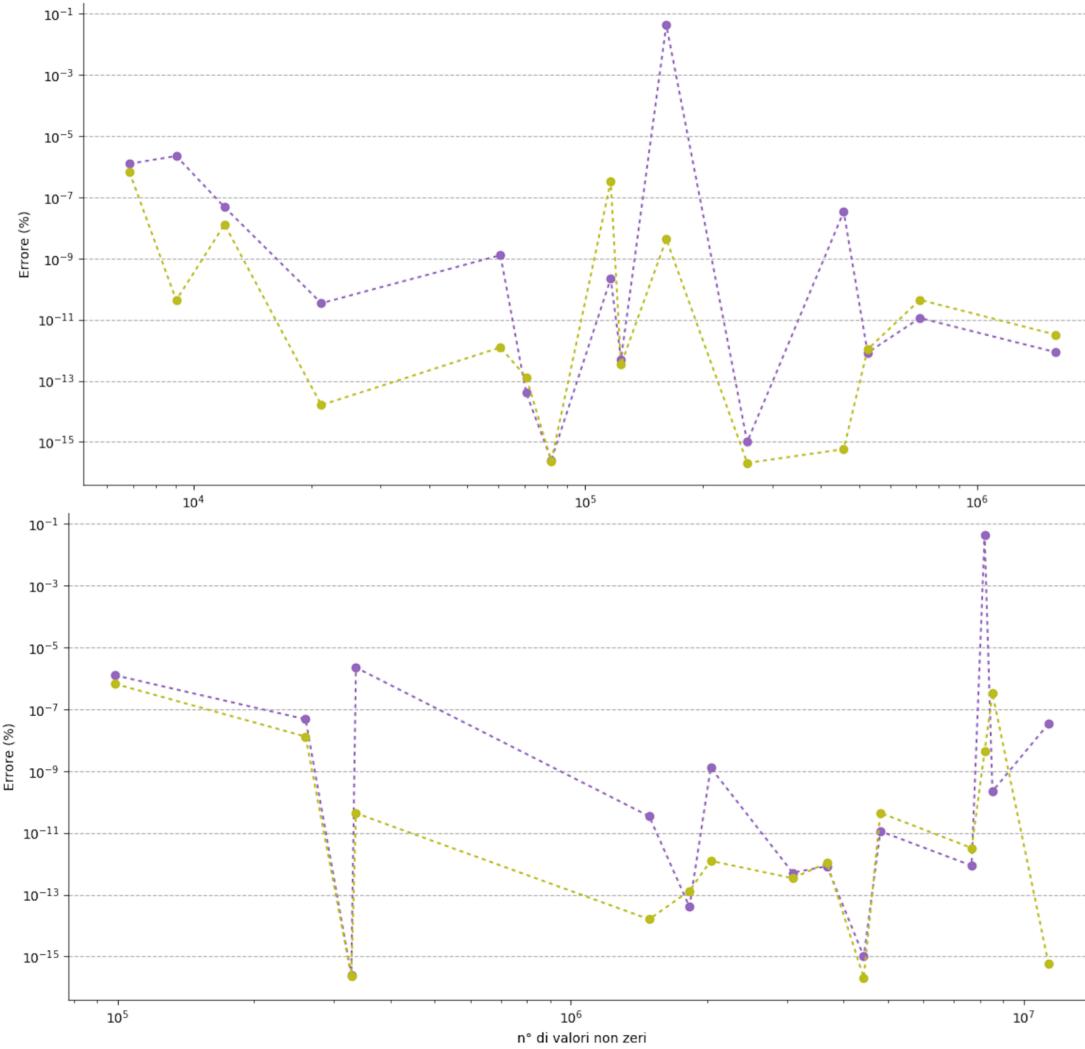
# Risultati MATLAB vs Python scipy



► Memoria utilizzata simile  
ad eccezione delle due  
matrici più grandi



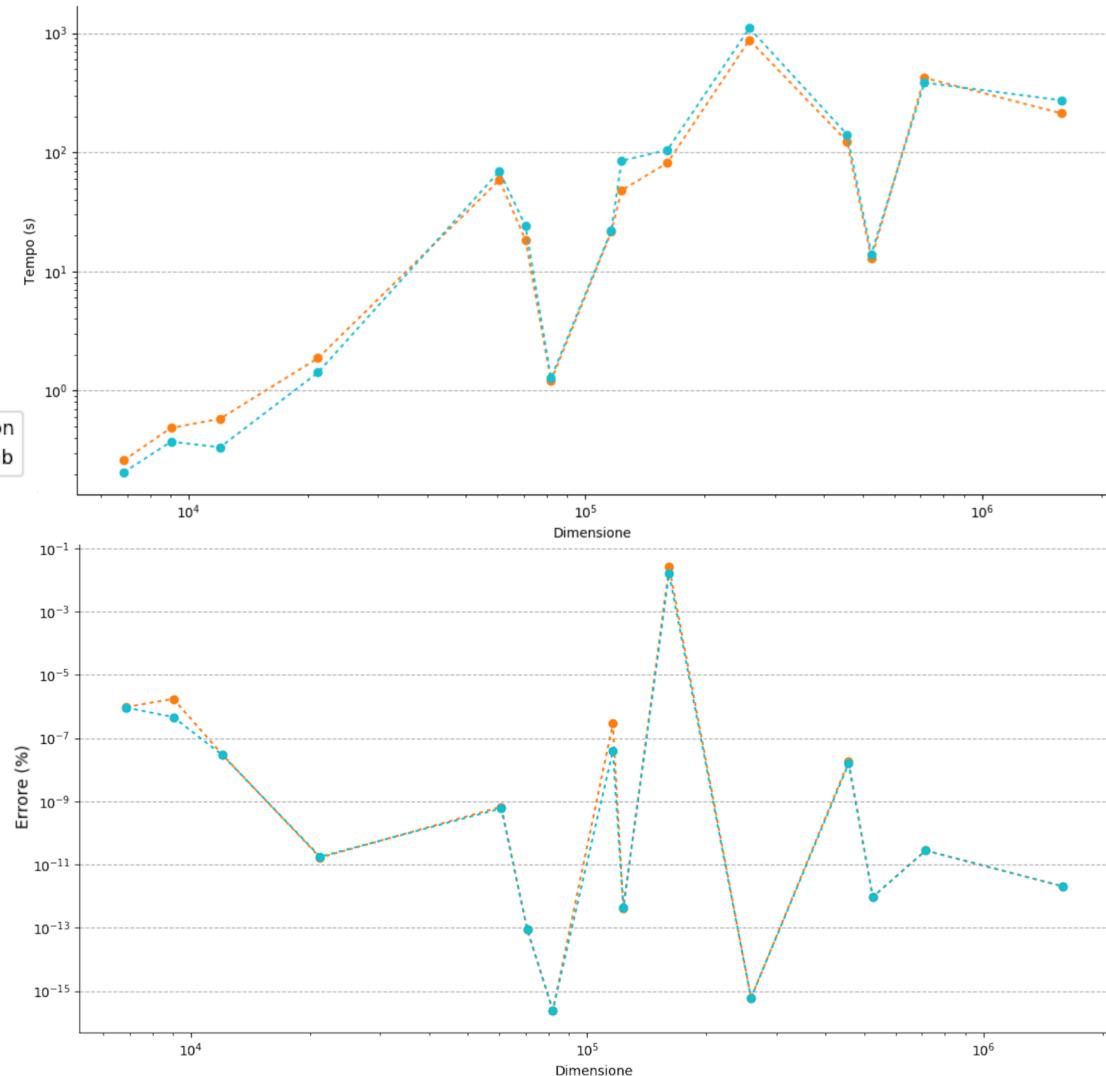
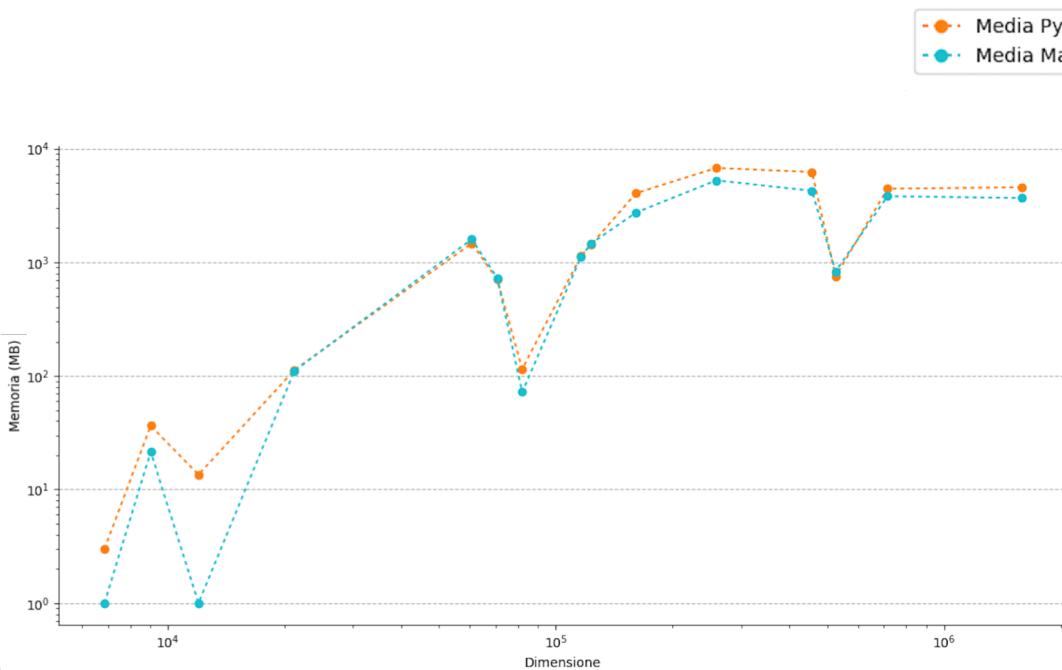
# Risultati MATLAB vs Python scipy



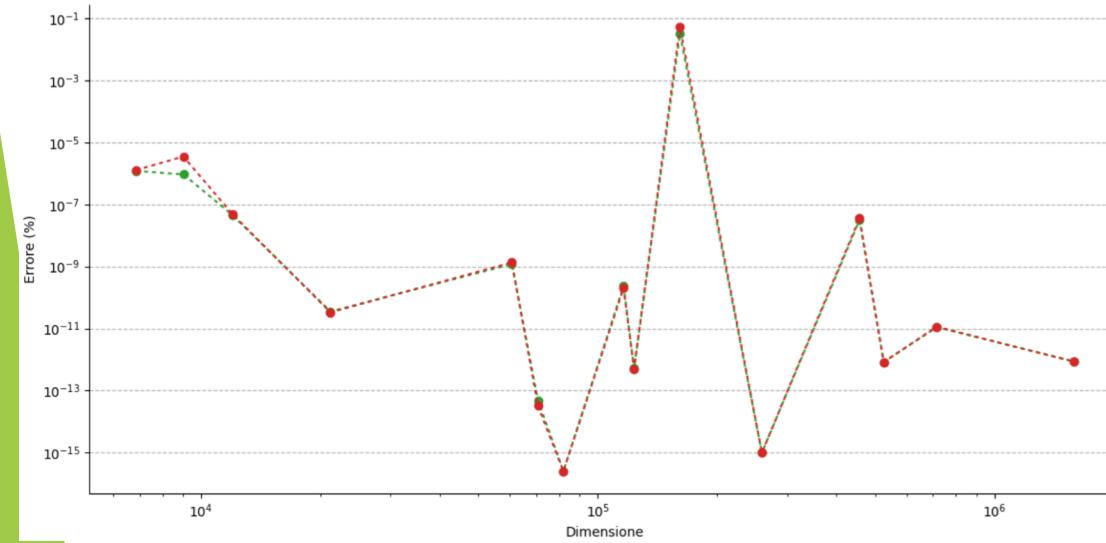
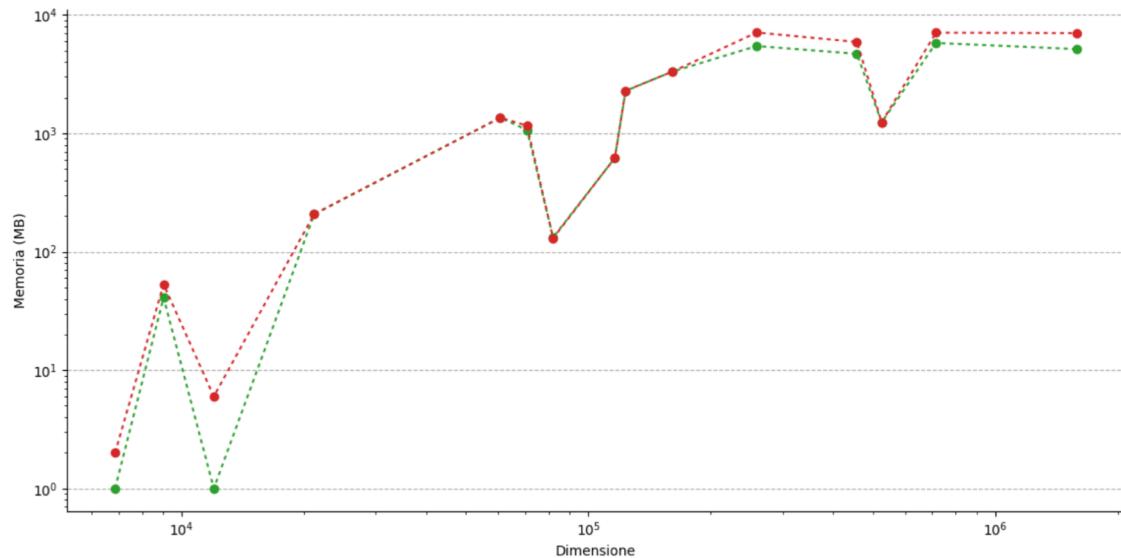
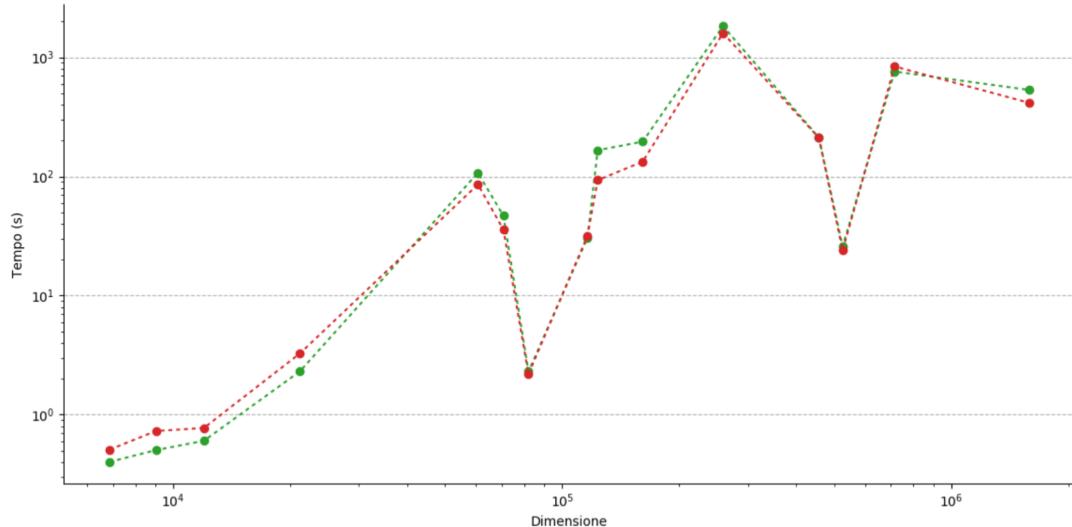
- ▶ L'errore relativo è tendenzialmente basso in entrambi i casi anche se Python tende ad aver errori più alti, in particolare per una delle matrici.

# Risultati Windows vs Linux

- ▶ Risultati sostanzialmente analoghi per entrambi i sistemi operativi. Windows leggermente più lento e contenuto nell'utilizzo della memoria.

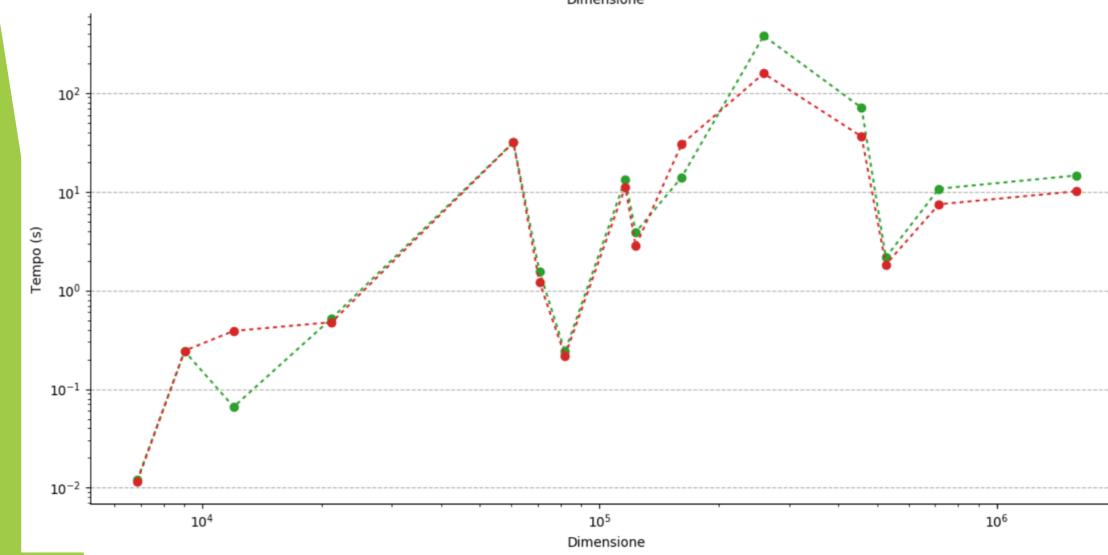
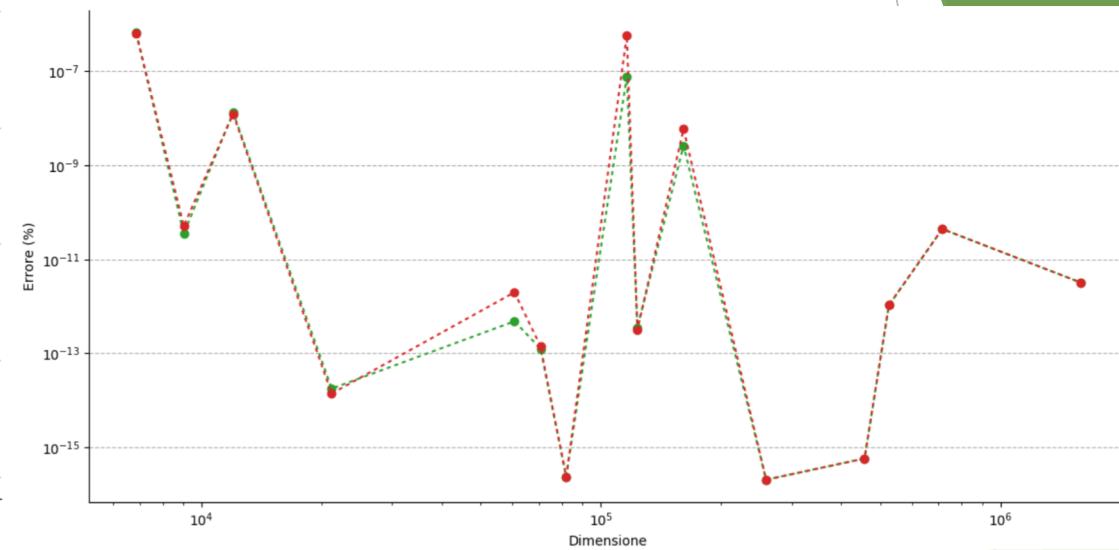
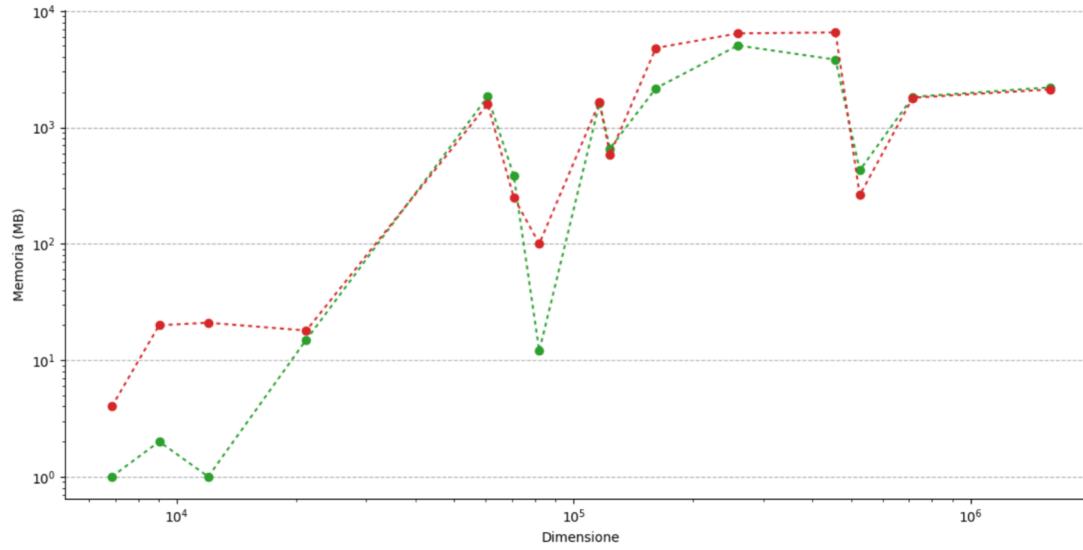


# Python - Windows/Ubuntu



■ Windows/Python  
■ Ubuntu/Python

# MATLAB - Windows/Ubuntu



● Windows/Matlab  
● Ubuntu/Matlab

# Conclusioni

- ▶ Per quanto la riguarda la scelta del sistema operativo le differenze sono veramente minime. Windows è leggermente più parsimonioso per quanto riguarda la memoria, mentre Ubuntu risulta più veloce.
- ▶ Potendo scegliere tra i due la nostra scelta ricade su Ubuntu che ha il vantaggio di essere un sistema operativo open-source, gratuito e UNIX-like.
- ▶ Per quanto riguarda il linguaggio, MATLAB è estremamente più performante per quanto riguarda i tempi di calcolo. Leggermente meno oneroso per quanto riguarda la memoria e tendenzialmente più stabile negli errori di calcolo.
- ▶ A meno che non sia richiesta la risoluzione di matrici particolarmente piccole per quanto riguarda il numero di righe, tra i due linguaggi MATLAB è sicuramente la scelta più adeguata.