# Sparse Linear System Solvers Comparison

Ferri Marco - 807130
Habbash Nassim - 808292

**Università degli Studi di Milano-Bicocca**

Course: *Methods of Scientific Computation*

# 1. INTRODUCTION

# WHICH IS THE BEST TOOL TO SOLVE SPARSE LINEAR SYSTEMS WITH **CHOLESKY DECOMPOSITION**?

- **Windows or Linux?** We'll test both.
- **MATLAB or some other open source software?** We'll try out C++ and Python.

# 2.
# CHOLESKY AVAILABLE LIBRARIES

# MATLAB

MATLAB has built-in capabilities to solve sparse linear problems using the Cholesky method.

As you can find here, whenever you do `x = A \ B` (where A and B are matrices of equal number of rows), MATLAB internally operates a resolution choice based on matrices shapes and symmetries, that always gives the optimal solver, thus the minimum computation time.

# C++

# PYTHON

# BLAS ???

Non è meglio dirlo solo alla fine, per evitare che ci venga detto “perché non avete usato direttamente le BLAS”?

Guarda le conclusioni.

# 3.
# COMPARISON TESTS

# OPERATION

All tests you can find in this presentation are made solving the equation Ax = b, where b is chosen for having the exact solution composed as xe = [1 1 1 1 1...]. In other words, b = A * xe.

# SAMPLES

All input sample are positive-definite and symmetric matrices that come from the SuiteSparse Matrix Collection.

In particular, the following matrices have been used:

- ex15
- shallow_water1
- cfd1
- cfd2
- parabolic_fem
- apache2
- StocF-1465
- Flan_1565
- G3_circuit

# PERFORMANCES

In order to declare the best library and operating system that fit your needs, we have developed some scripts you can find in this GitHub repository and later on inside this document.

For each library, we'll consider three performances:

- Resolution Time
- Used Memory
- Relative Error

# ENVIRONMENTS

We'll soon present the results of our study. We ran our tests on Azure Virtual Machines.

The following are both the Windows and Linux virtual machines technical specifications:

- 8 VCPUs
- 56 GB of RAM


We used Ubuntu 18.04 and Windows 10 Pro 1803.

# 4.
# SOURCE CODES

Also available on *https://github.com/dodicin/slss-comparison-test*

# MATLAB

```
load(inputfile);
[rows, columns] = size(Problem.A);

xe = ones(columns, 1);
b = Problem.A * xe;
x = Problem.A \ b;

error = norm(x - xe) / norm(xe);
```

# C++ Eigen

```
typedef SparseMatrix<double, ColMajor, long long> SpMat;

SpMat A;
loadMarket(A, argv[1]);
VectorXd xe = VectorXd::Constant(A.cols(), 1);
MatrixXd b = A.selfadjointView<Lower>() * xe;

SimplicialLDLT<SpMat> chol(A.selfadjointView<Lower>());
VectorXd x = chol.solve(b);
double rel_error = ((VectorXd)(x - xe)).norm()/xe.norm();
```

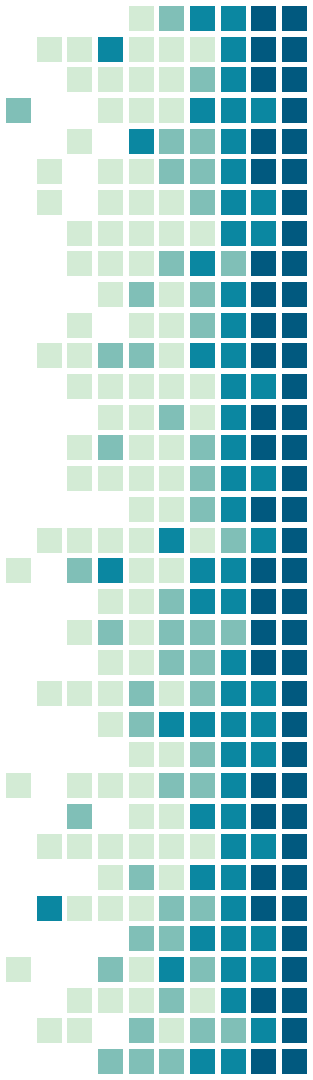
# Python Scikit-Sparse

```
mat = io.mmread(sys.argv[1])

A = sp.sparse.csc_matrix(mat)
xe = np.ones(A.shape[0])
b = A.dot(xe)

factor = cholesky(A)
x = factor.solve_A(b)

rel_error = np.linalg.norm(x-xe)/np.linalg.norm(xe)
```



# 5.
# RESULTS

# SOME NOTES

We ran MATLAB and Eigen scripts both on Windows and Linux, while Scikit-Sparse was only tested on Linux, because of the impossibility of installing it on Windows. Specifically, we had problem obtaining SuiteSparse library and compiling cholmod.h to run it with Python.

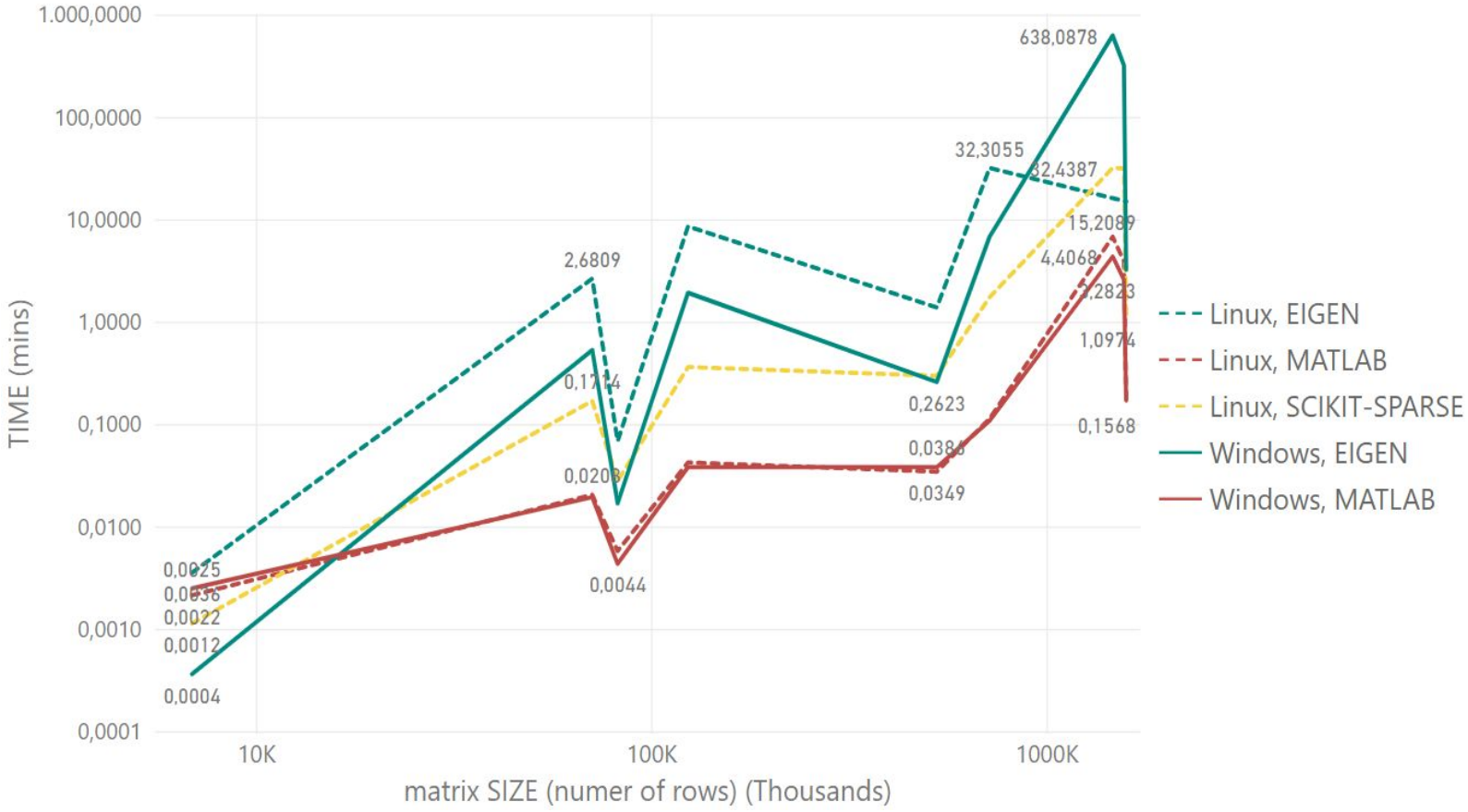
Also, you will notice that some value are missing for the two largest matrices with C++ Eigen library.

# SOME NOTES

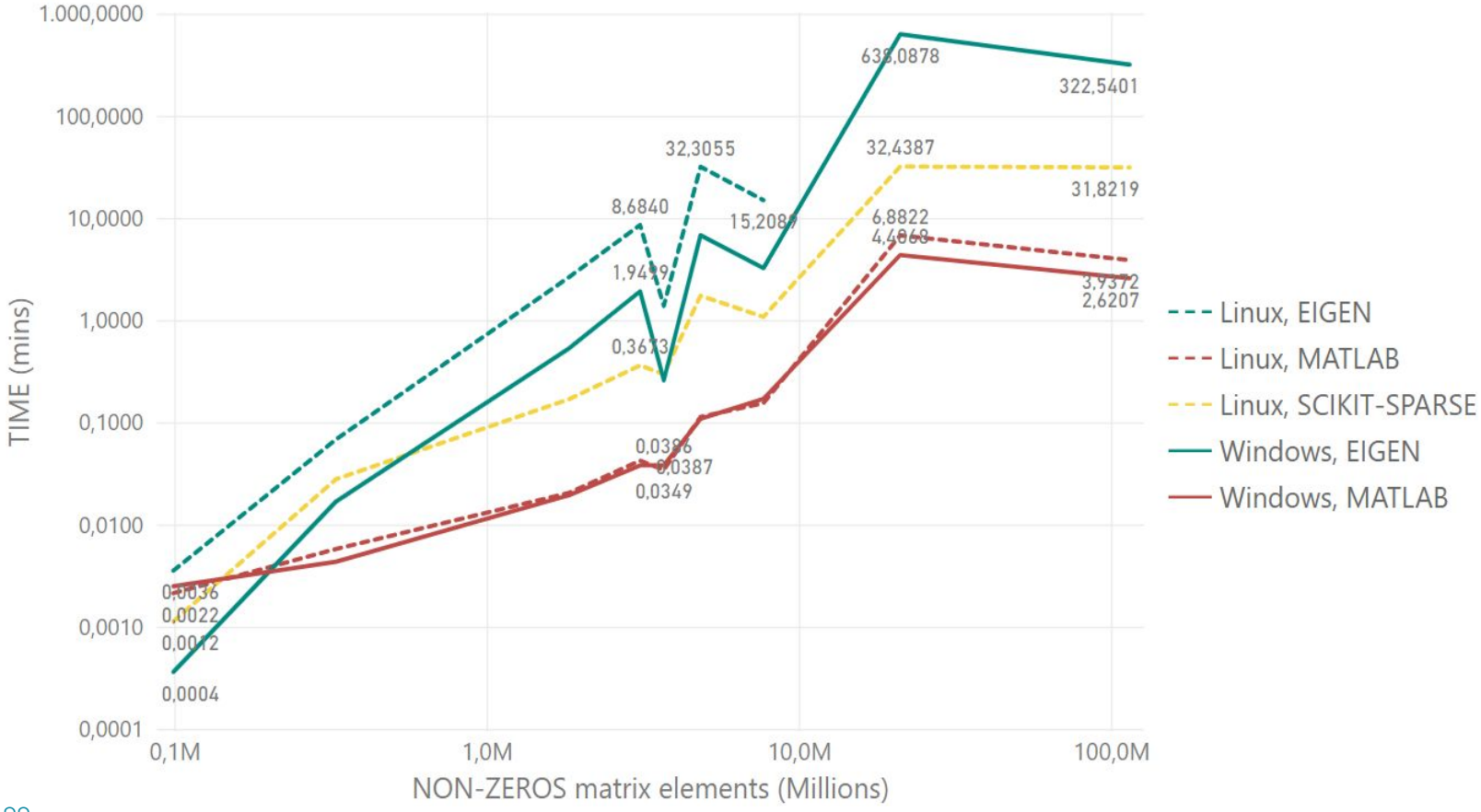
Although the first idea of plotting some matrices resolution result charts could be to put the matrix size on the X-axis, the most natural way to present this type of data for **sparse matrices** is to consider, for the abscissa, the number of non-zero elements that appear in each matrix.

However, we'll show you both size and non-zeros based charts.

# RESOLUTION TIME BY MATRIX SIZE



# RESOLUTION TIME BY MATRIX NON-ZEROS ELEMENTS

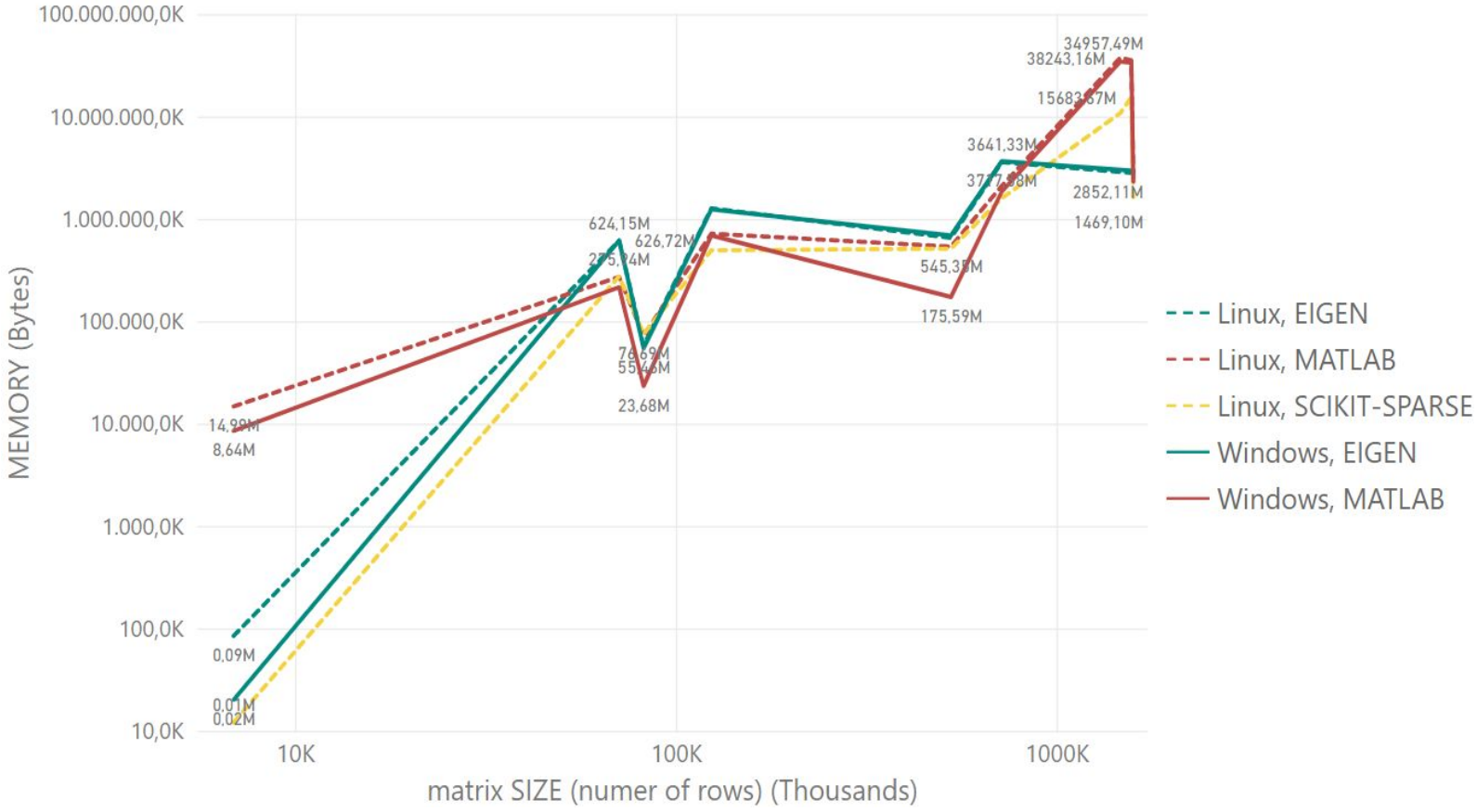


# RESOLUTION TIME CONSIDERATIONS

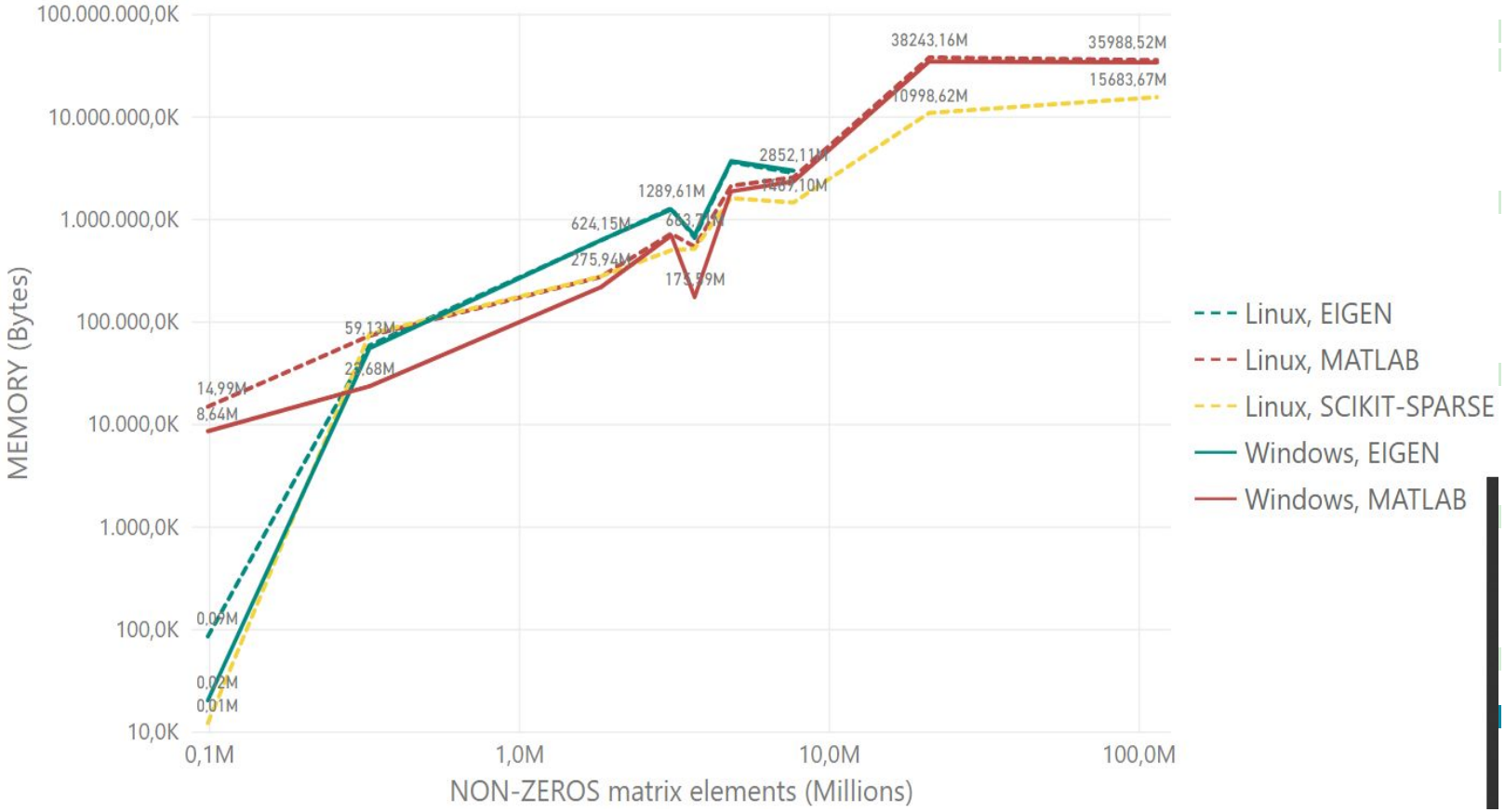
As previously mentioned, you can easily see that non-zeros based chart is more significant than the size based one.

Speaking about time, we notice that **MATLAB is significantly faster** than the others. It manages the most complex matrix in just 6 minutes, versus 32 minutes for Python Scikit and more than 10 hours for C++ Eigen (even missing on Windows due to computational troubles). Instead, we don't find very much difference between Windows and Linux.

# MEMORY BY MATRIX SIZE



# MEMORY BY MATRIX NON-ZEROS ELEMENTS



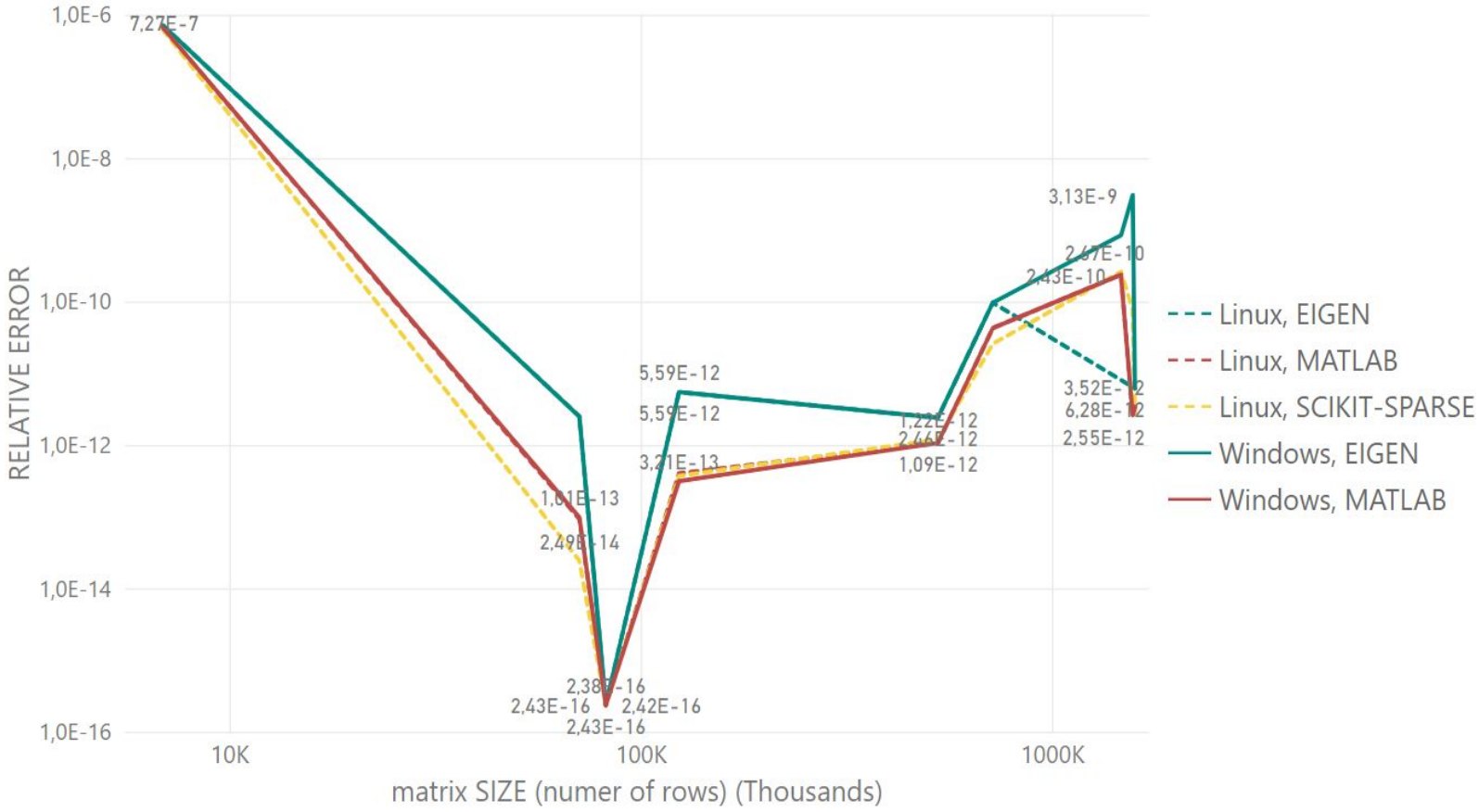
# MEMORY CONSIDERATIONS

Memory usage is pretty similar among all tested solutions until you compute matrix with less than 1 million non-zeros elements.

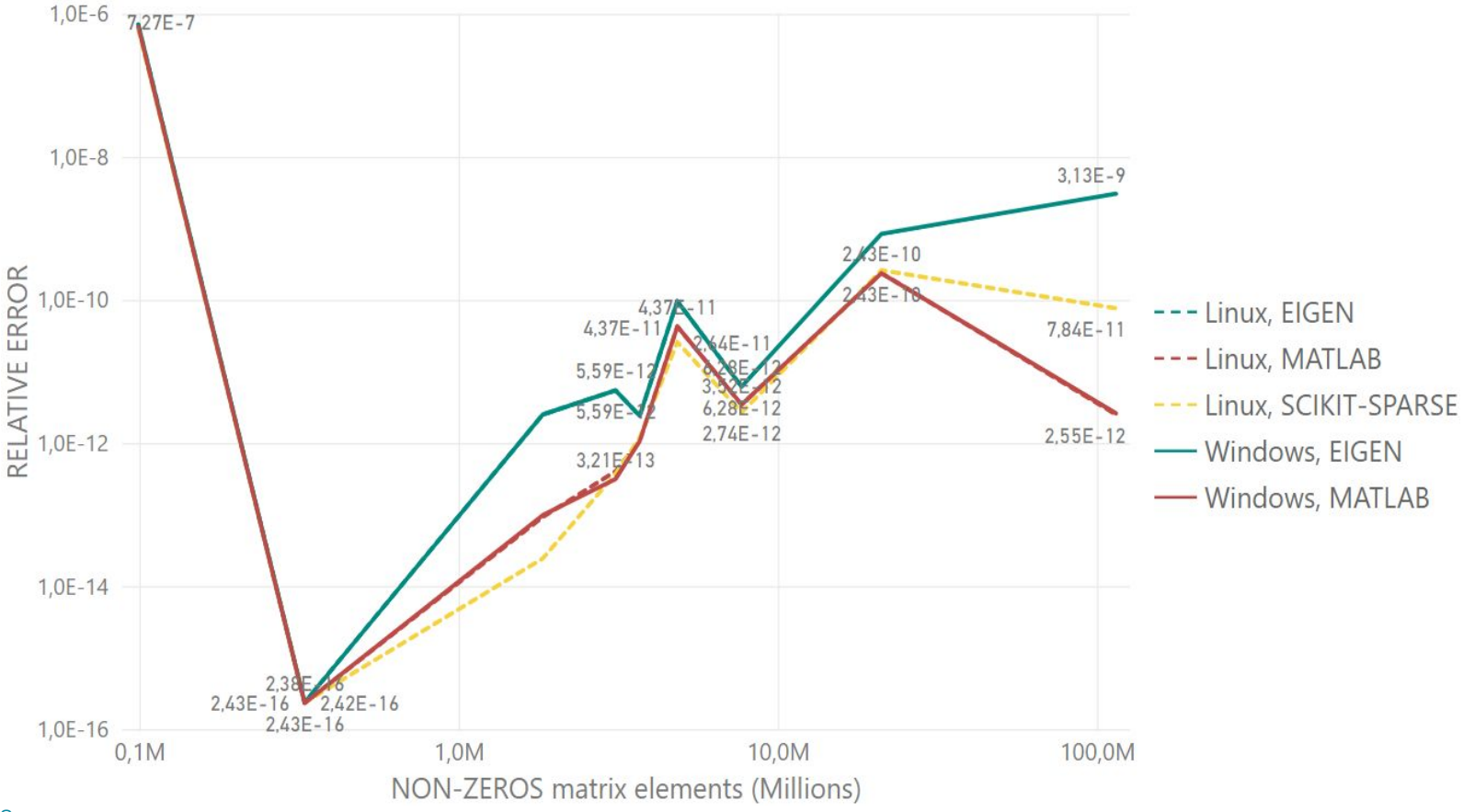
Once you overpass that threshold, **Python Scikit is almost four times better than MATLAB**, since it doesn't require too much RAM to find the solution.

Unfortunately, you can't say anything about C++ Eigen, due to technical problems and too long computation over time.

# RELATIVE ERROR BY MATRIX SIZE



# RELATIVE ERROR BY MATRIX NON-ZEROS ELEMENTS



# RELATIVE ERROR CONSIDERATIONS

Regarding relative error, all algorithms that come to an end have almost the same outcome.

You can only notice a relevant difference in the last matrix, where C++ Eigen and MATLAB relative errors differ for three order of magnitude; the latter is overall the most precise among tested libraries.

# 6. CONCLUSIONS

# CONCLUSIONS

As we seen in the previous slides, **MATLAB seems to be the fastest tool** for solving linear sparse systems with Cholesky Decomposition. However, on large matrices it needs a significant amount of memory to work properly.

It also produces the minimum relative error compared with the other ones.

# CONCLUSIONS

Python represents a good open source alternative to MATLAB with Scikit-Sparse. Although it is five times slower, it uses just one third of the memory that MATLAB needs.

Then, **Scikit-Sparse is a nice pick for Linux users with few RAM** and no particular time urgency. Anyway, MATLAB could still be the best one if a very fast hard drive is mounted on the machine, dedicating it to memory swap.

# CONCLUSIONS

C++ **Eigen is absolutely the worst choice** both in terms of time, memory and relative error.

It could be interesting investigating more with Eigen enabling the support of Intel Math Library, useful to significantly speed up the computation on Intel CPUs for Linear Algebra, Fast Fourier Transforms and more.

https://software.intel.com/en-us/mkl-developer-reference-c-overview

# THANKS!

## Any questions?

You can find us at:

m.ferri17@campus.unimib.it

n.habbash@campus.unimib.it

*https://github.com/dodicin/slss-comparison-test*