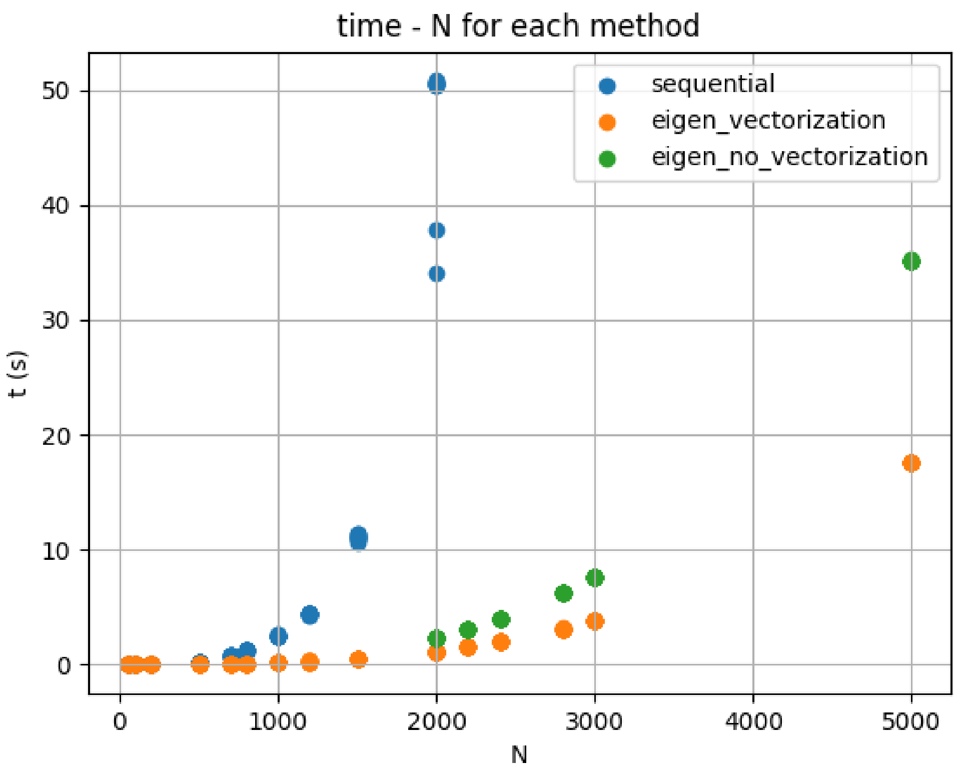
Néstor Monzón - 735418

Cristina Oriol - 755922

Lab 1 PACS

To check the performance of the different versions developed we have made some different tests. For different sizes (different N) Matrix our test makes ten executions of each method. We have initially chosen as N, N=(50, 100, 200, 500, 700, 800, 1000, 1200, 1500, 2000) for the three methods. As we realized that the Eigen methods were much faster, we also tested them with up to N=5000.

We have measured the user time returned by the time command (as we only care about the time in which the program is actually running, without the system time). With these time results we have elaborated the following plot:



The first conclusion we get is the significant difference between the sequential and the two versions of eigen. The time on the sequential mode corresponds to the nested loops made to calculate the multiplication, following the given scheme. For this reason it is logical that the time increases as a function of N, since it requires a greater number of iterations in the loops. As there are three nested loops of size N, the algorithm is $O(N^3)$. This explains the quick raise in cost we see in the graph.

With the Eigen library we see a huge improvement, allowing larger values of N to be tested, thanks to their more efficient implementation. The cost seems to just grow with $O(N^2)$ instead of cube.

Finally, disabling vectorization by using the *EIGEN_DONT_VECTORIZE* directive when compiling, we see a 2x increase in computational time for each value of N.

This is because the vectorization takes advantage of the processor architecture which enables instruction parallelism, meaning performing the same instruction on multiple data at the same time

(SSE2[1] in the case of the computers in the lab). According to the recorded time, we can assume in this case two operations are performed at the same time, effectively halving the computational time.

[1] https://en.wikipedia.org/wiki/SSE2