Séance Projet 2 : Subspace Iteration Methods

Grégoire Honvault, Rachida Oussakel

Département Sciences du Numérique - Première année
2021-2022

Table des matières

Table des figures

1 Introduction

In the first part of the project, we saw that only the leading eigenpairs of the variance/covariance matrix are needed for the Principal Component Analysis (PCA) and that they provided enough information about the data. We then implemented the power method with deflation, in order to find these leading eigenpairs.

In this part, we will see why this algorithm is not efficient in terms of performance, and we will study 4 variants of a more efficient alternative called the **subspace iteration method**.

2 Limitation of the power method

The basic *power method* can be used to determine the eigenvectors associated to the largest (in module) eigenvalue.

Question 1 : We find that **eig** is clearly ahead, with an average $\approx 0.20s$ to find all the eigenpairs of the matrix with an extremely good quality (10^{-14}). On the other hand, although *powerv11* also provides eigenpairs of great quality, it's only a few of the total number and 4 times the computing time : $\approx 0.72s$. On top of that, the computing time for the inflated method can get much worse when computing a huge matrix.

Question 2 : See figure 1 for the new algorithm. After implementing this algorithm we see on figure 2 that the new average computing time is $\approx 0.43s$ which is indeed almsot twice as fast as the proposed algorithm.

```
% méthode de la puissance itérée
v = randn(n,1);
z = A*v;
beta = v'*z;

% conv = || beta * v - A*v||/|beta| < eps
% voir section 2.1.2 du sujet
norme = norm(beta*v - z, 2)/norm(beta,2);
nb_it = 1;
while(norme > eps && nb_it < maxit)
  %% changements effectués par rapport à powerv11
  v = z / norm(z,2);
  z=A*v;

  %% retour à powerv11
  beta = v'*z;
  norme = norm(beta*v - z, 2)/norm(beta,2);
  nb_it = nb_it + 1;
end
```

FIGURE 1 – New optimization implemented in $power\_v12$

Question 3 : The deflated power method is less efficient because the eigenpairs are found one by one, and very costly matrix operations have to be repeated each time on a new matrix $A - \beta * (VV^\top)$.

```
******* calcul avec la méthode de la puissance itérée ******

Temps puissance itérée = 7.190e+00
Nombre de valeurs propres pour attendre le pourcentage = 46
******* calcul avec la méthode de la puissance itérée ******

Temps puissance itérée = 4.290e+00
Nombre de valeurs propres pour attendre le pourcentage = 46
```

FIGURE 2 – Old and New processing time for the inflated power method

3 Extending the power method to compute dominant eigenspace vectors

3.1 subspace iter v0 : a basic method to compute a dominant eigenspace

As computing a matrix of m vectors would not tend to the expected result using our current version of the power method algorithm, a few adjustments have to be made. A new method to compute an invariant subspace associated to the largest eigenvalues of a symmetric matrix is proposed. Adjustments are made : new stopping criterion, an orthonormalised initial matrix as well as orthonormalisation of the matrix on each iteration are needed.

Question 4 : The power vector algorithm, when applied to a randomly generated n*m matrix (instead of a n*1 vector), will converge towards a n*m matrix with all identical columns. In fact, these columns are all the same n*1 dominant eigenvector with we would have found computing only the n*1 random vector. The eigenvalues found are also all the same value : the dominant eigenvalue associated to the eigenvector found. As expected, 3 the matrix with all identical columns, and 4 the matrix of all identical eigenvalues.

0.0422	0.0424	0.0423
-0.0042	-0.0042	-0.0042
0.0256	0.0262	0.0259
-0.0195	-0.0199	-0.0197
-0.0679	-0.0681	-0.0680
0.0184	0.0182	0.0183
-0.0092	-0.0090	-0.0091
-0.0261	-0.0256	-0.0259
0.0656	0.0650	0.0653
0.0218	0.0221	0.0219
-0.0040	-0.0040	-0.0040
-0.0460	-0.0463	-0.0462
-0.0250	-0.0244	-0.0247
0.0041	0.0039	0.0040
0.0115	0.0117	0.0116
-0.0636	-0.0637	-0.0636
-0.0011	-0.0013	-0.0012
-0.0443	-0.0450	-0.0446
-0.1091	-0.1093	-0.1092
-0.0158	-0.0165	-0.0161

FIGURE 3 – Resulting m vectors of the power method algorithm

9.1651e+04	9.1648e+04	9.1650e+04
9.1648e+04	9.1651e+04	9.1650e+04
9.1650e+04	9.1650e+04	9.1651e+04

FIGURE 4 – Resulting eigenvalues of the power method algorithm

Question 5 : In algorithm 2 there is indeed a computation of the whole spectral decomposition of H, using eig. However, this computation is done on a noticably smaller-sized matrix H (m*m) compared to A (n*n). In fact m (=20 in our tests), the number of largest eigenvalues we aim for, is 10 times smaller than n (=200), which makes eig way more time efficient.

Question 6 : *See matlab file subspace-iter-v0.m*

3.2 subspace iter v1 : improved version making use of Raleigh-Ritz projection

In this version of the subspace iteration method, improvements are made to optimize its efficiency.

Question 7 : The algorithm and the corresponding lines :

Generate an initial set of m orthonormal vectors $V \in R^{n \times m}$ *(lines **48, 49**)*
We have $k = 0$, *(line **38**)*
$PercentReached = 0$ *(line **40**)*
repeat
  $k = k + 1$ *(line **54**)*
  Compute Y such that $Y = AV$ *(line **56**)*
  $V \leftarrow$ orthonormalisation of the columns of Y *(line **58**)*
  Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V *(line **61**)*
  Convergence analysis step : save eigenpairs that have converged *(lines **92, 94**)*
  and update $PercentReached$ *(line **97**)*
until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$) *(lines **52, 115**)*

---

4 subspace iter v2 and subspace iter v3 : toward an efficient solver

New optimizations are hereby proposed. The idea is to combine both the block approach and the deflation method to speed up the convergence of the solver.

4.1 Block approach (subspace iter v2)

Orthonormalisation is performed at each iteration and is quite costly. One simple way to accelerate the approach is to perform p products at each iteration (replace $V = AV$ (first step of the iteration) by $V = A^pV$).

Question 8 : We have a matrix $A \in R^{n\times n}$ and we are computing $A \times A$ so the cost of this operation is $O(2n^3)$. In order to compute A^p the multiplication is done $p-1$ times, so the cost is $O(2(p-1)n^3)$.

As far as V is concerned we have $V \in R^{n\times m}$ with $m << n$. Processing A^pV comes down to a $R^{n\times n} \times R^{n\times m}$ multiplication, so the cost is $O(2pn^3 + 2mn^2) \sim O(2pn^3)$.

In order to reduce the cost, A^p was processed trough p $A \times V$ operations (as we see in figure 5). The complexity is then $O(2pmn^2)$ with $m << n$ so it is much more efficient than $O(2pn^3)$.

```
for i=1:p
    Y=A*Vr;
    Vr=Y;
end
```

FIGURE 5 – The implemented A^p operation

Question 9 : *See matlab file subspace-iter-v2.m*

Question 10 : Figure 6 (log. scale) highlights 2 behaviours.

Firstly, the number of iterations needed to compute matrices of types 1 and 4 is in general much higher than for types 2 and 3, because the algorithm stops upon reaching a certain percentage of the matrix's trace.
This means that for types 2 and 3, which have a few high dominant values and a lot of eigenvalues close to 0 (see figure 8), once these few dominant values are obtained, the needed percentage of the trace is reached very fast, and the loop stops.
For types 1 and 4 however, eigenvalues slowly decrease on a linear curve, which means a lot more eigenvalues have to be obtained to reach the needed trace percentage.

Secondly, we see that increasing p reduces the number of iterations and the computing time of this algorithm for every matrix type. However, at around $p = 15$ (resp. $p = 25$), increasing p increases greatly the computing time for a matrix of type 2 (resp. type 3) while it keeps decreasing for types 1 and 4.

4.2 Deflation method (subspace iter v3)

Because the columns of V converge in order, we can freeze the converged columns of V . This freezing results in significant savings in the matrix-vector ($V = AV$), the orthonormalisation and Rayleigh-Ritz Projection steps.

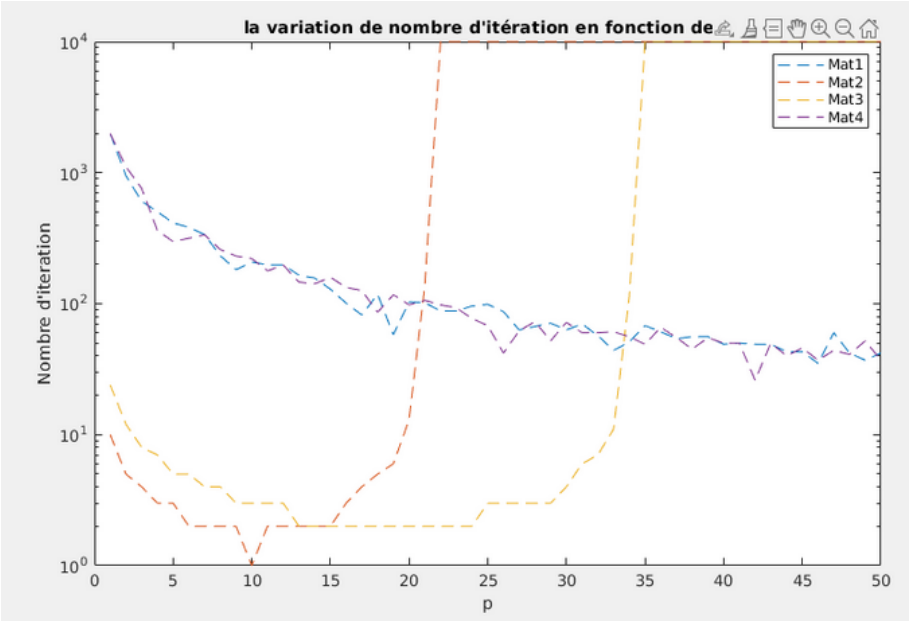

FIGURE 6 – Number of iterations as a function of p for the 4 matrix types (log scale)

Question 11 : On figure 7 *subspace_iter_v*1 we see the accuracy of the dominant eigenpair is 10^6 times better than the least dominant that was calculated.

This is because the more dominant the eigenpair is, the faster it converges, and in *subspace_iter_v*1, the eigenpairs that have already converged (with a sufficient accuracy for our criteria), are still being processed in the next loops, together with the eigenpairs that have not yet converged. And each "bonus" loops makes the eigenpairs that have already converged more accurate even if it is not needed, while for example the least dominant eigenpair is less accurate because the loop stops upon reaching just the needed accuracy (the given criteria).

5.8764e-14
1.1057e-13
2.0013e-12
6.9341e-12
1.4856e-11
2.4215e-10
1.4643e-10
8.2451e-10
1.5135e-09
1.4531e-08
8.5341e-08

FIGURE 7 – Accuracy of each eigenpair for *subspace_iter_v*1

Question 12 : The number of iterations needed to converge will stay the same. However, the number of operations will be greatly reduced since the the vectors that have converged will not be processed any longer. Furthermore, this means the accuracy will be much closer between all the eigenpairs since this no "bonus" loops anymore, that would make the most dominant eigenpairs more accurate than the given criteria.

Question 13 : *See matlab file subspace-iter-v3.m*

5 Numerical Experiments

Question 14 : The matrices are all full and symmetric matrices of the same size ($n \times n$). The difference between them is the distribution of their eigenvalues. On figure 8 we see the curve of the eigenvalues of the matrices of type 1 and 4 are linear while those of types 2 and 3 are of a $\frac{1}{x}$ shape. It should be noted that type 1 has eigenvalues varying from 0 to 200, while for the 3 other types eigenvalues stay between 0 and 1.

These 4 types of matrices are used as different specific examples of matrices that could be processed by the algorithm, so we can test them to see which algorithm, with which parameters, is best fit for each type of matrix. (as we saw for the value of p in Q.10)

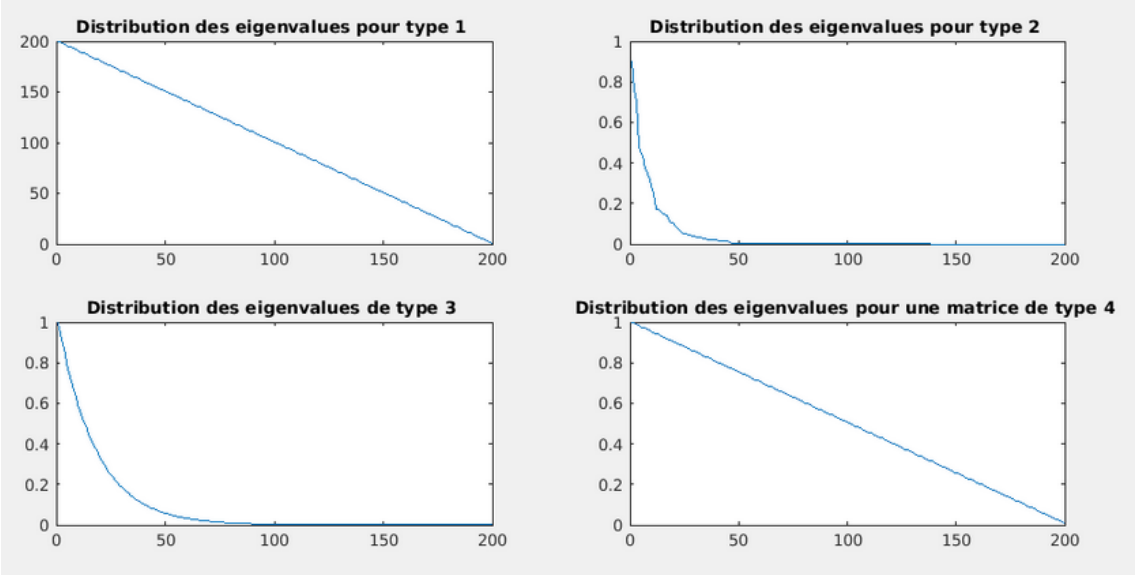

FIGURE 8 – Eigenvalues distribution for each type of matrix

Question 15 : The figures show that overall, as we already knew, bigger matrices and matrices of type 1 and 4 are much more time consuming (almost 4 times higher in figure 11). We also It is also obvious now that the power method can never beat the native eig of matlab which shows extremely low computing times.

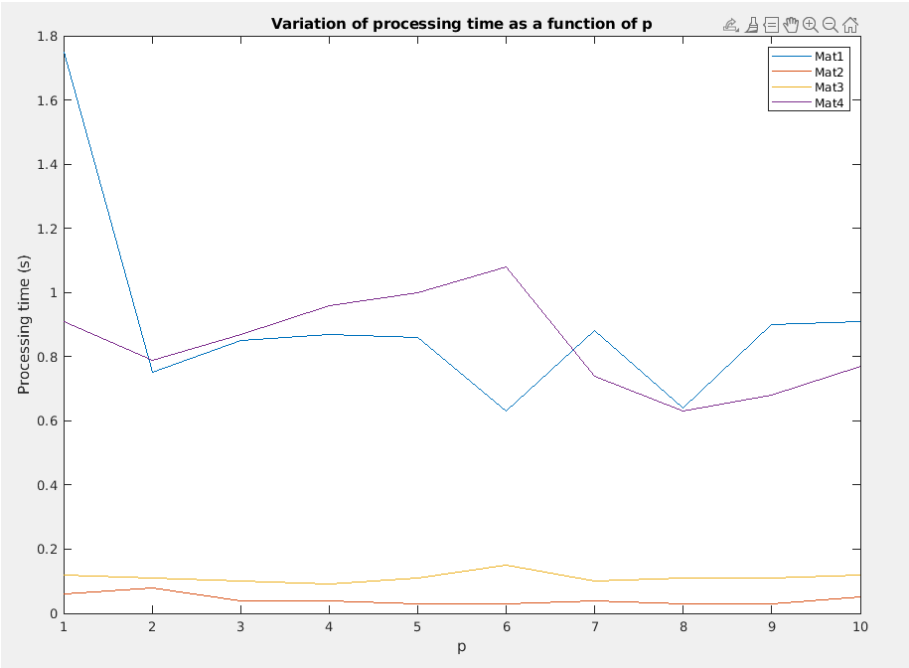

FIGURE 9 – Processing time for *subspace_iter_v*0 for each value of p (n=200)

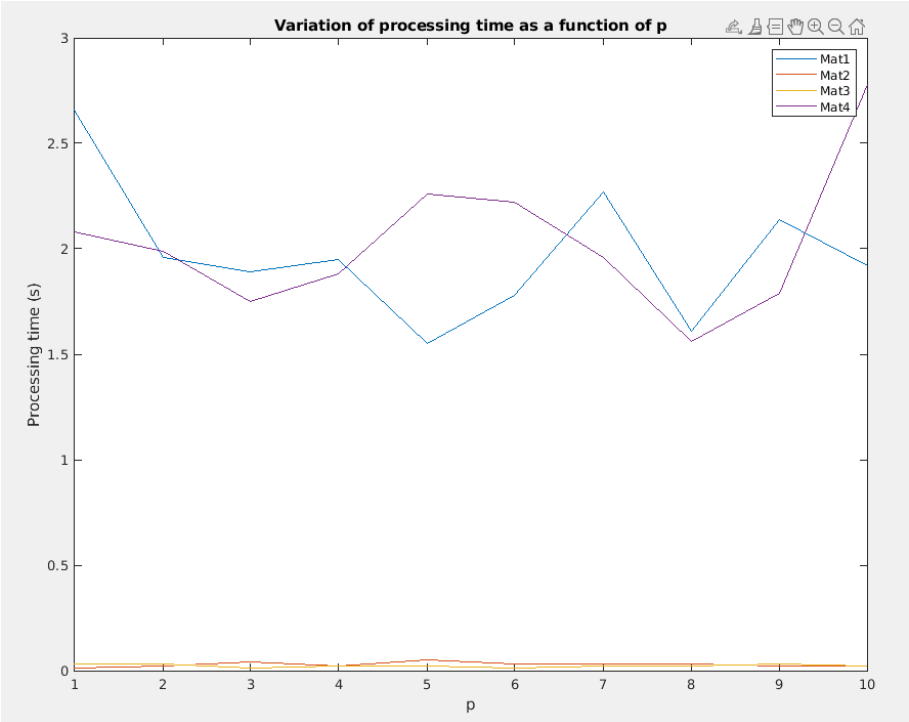

FIGURE 10 – Processing time for *subspace_iter_v*1 for each value of p (n=200)

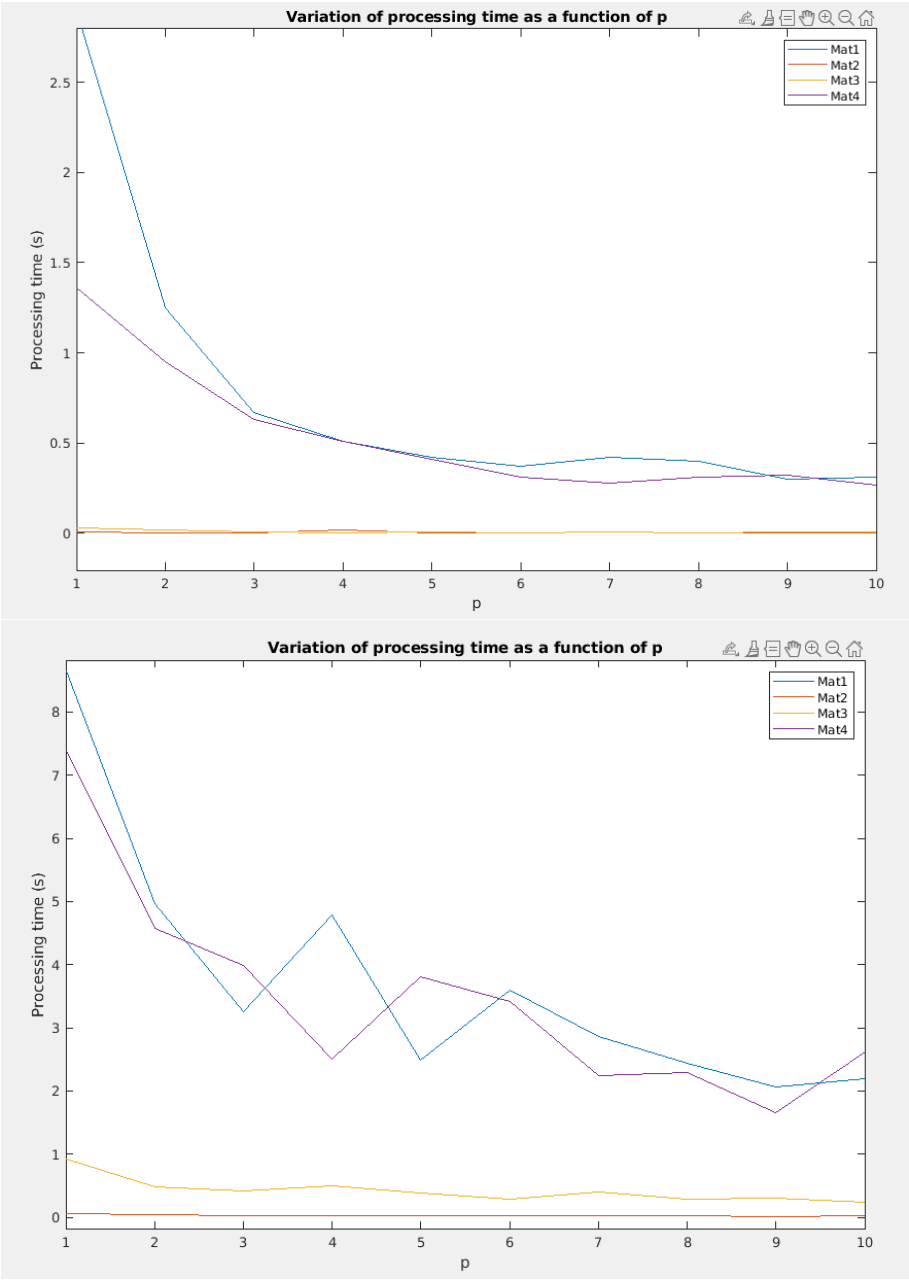

FIGURE 11 – Processing time for *subspace_iter_v2* for each value of p (n=200 then n=1000)

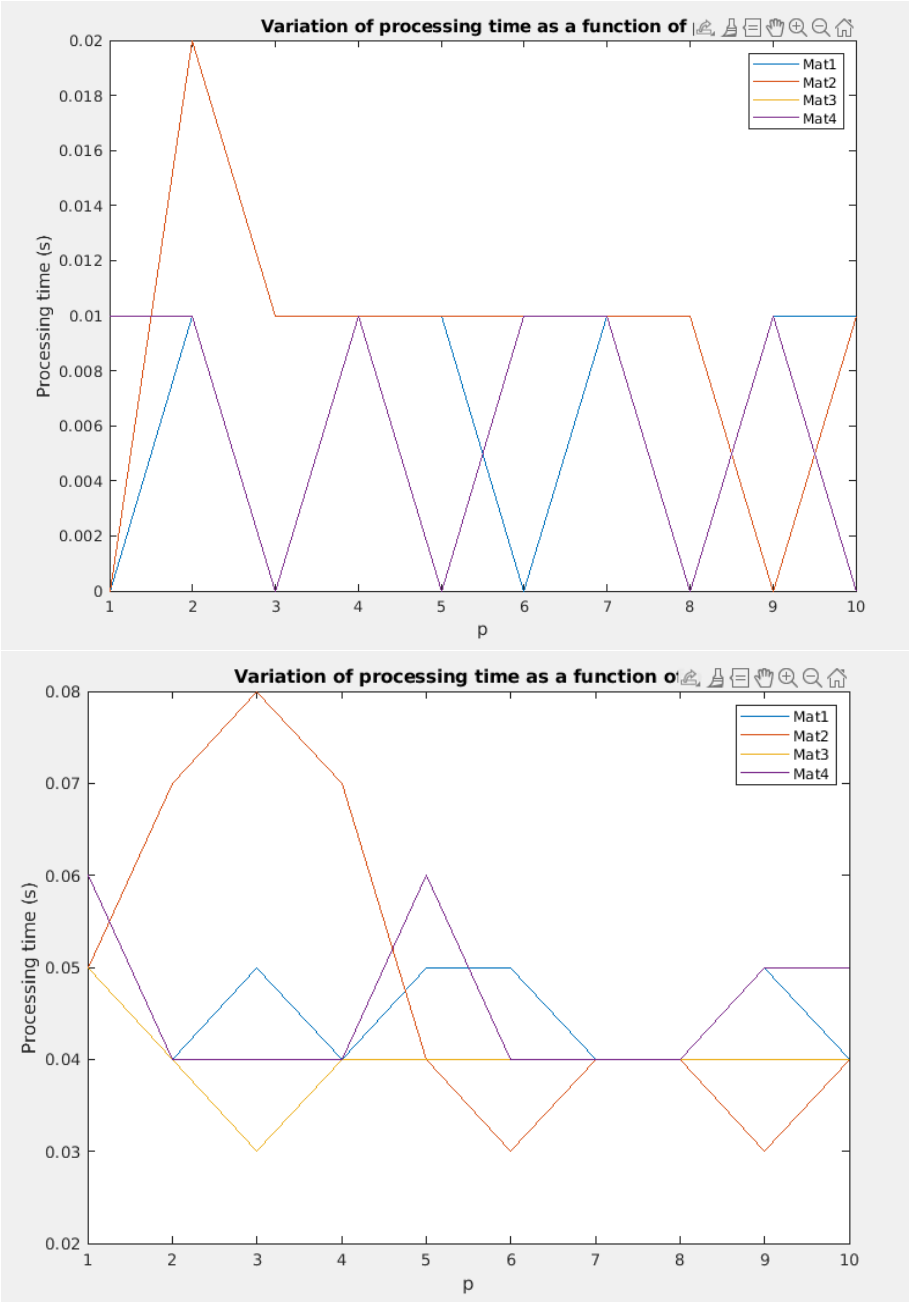

FIGURE 12 – Processing time for *eig* for each value of p (n=200 then n=1000)