Projet de Calcul Scientifique

Frenois Etan & Rouot Emile

Mai 2024

Département Sciences du Numérique

Première année

Table of contents

1 Introduction

In this project, we will see that this specific algorithm is not efficient in terms of performance. Then we will present a more efficient method called subspace iteration method, based on a mathematical object called *Rayleigh* quotient.
We will study four variants of this method.

2 Subspace Iteration Methods Application

2.1 Limitations of the power method

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

Algorithm 1 Vector power method

Input: Matrix $A \in \mathbb{R}^{n \times n}$
Output: (λ_1, v_1) eigenpair associated to the largest (in module) eigenvalue.
$v \in \mathbb{R}^n$ given
$\beta = v^T \cdot A \cdot v$
repeat
  $y = A \cdot v$
  $v = y / \|y\|$
  $\beta_{old} = \beta$
  $\beta = v^T \cdot A \cdot v$
until $|\beta - \beta_{old}| / |\beta_{old}| < \varepsilon$
$\lambda_1 = \beta$ and $v_1 = v$

By adding the deflation process, we are able to compute the eigenpairs we need, for instance, to reach a certain percentage of the trace of A.

Question 1 :

Here are the results we observed using Matlab's *eig* and *power_v11* functions :

Matrix size (n) & type ($imat$)	***eig* function**	***power_v11* function**
$n = 200$ & $imat = 1$	8.000e-02	1.680e+00
$n = 200$ & $imat = 2$	5.000e-02	3.000e-02
$n = 200$ & $imat = 3$	2.000e-02	5.000e-02
$n = 200$ & $imat = 4$	2.000e-02	1.600e+00
$n = 1000$ & $imat = 1$	3.900e-01	convergence not achieved
$n = 1000$ & $imat = 2$	3.200e-01	3.970e+00
$n = 1000$ & $imat = 3$	3.100e-01	2.139e+01
$n = 1000$ & $imat = 4$	3.500e-01	convergence not achieved
$n = 50$ & $imat = 1$	1.000e-02	2.000e-02
$n = 50$ & $imat = 2$	1.000e-02	1.000e-02
$n = 50$ & $imat = 3$	4.000e-02	3.000e-02
$n = 50$ & $imat = 4$	1.000e-02	2.000e-02

Figure 1 - Calculation time between *eig function* and *power_v11* function

We can see that for a matrix of size 200x200, the eig method is faster for matrices of type 1

and 3, while the iterated power method is faster for matrices of type 2 and 4. For a matrix of size 1000x1000, the eig method is always faster. We also note that for matrices of type 1 and 4, convergence is not achieved with the iterated power algorithm. Finally, for small matrices (size 50x50), the two methods are more or less equivalent.
Overall, we can say that the *eig* function is much faster than the *power_v11* function and also more accurate. Moreover, it computes all eigenvector/eigenvalue pairs, which is not the case for the power method.
This can be explained by the fact that the *power_v11* function is a "basic" method, whereas the *eig* function takes advantage of the structure and properties of the matrix to improve efficiency and accuracy.

Question 2 :

Looking closely at the proposed algorithm, two matrix $\times$ vector products, $A \cdot V$, are identified inside the loop.
We have proposed an algorithm below, which will only have one matrix $\times$ vector product in the loop, making the basic algorithm more efficient :

Algorithm 2 Vector power method improved

Input: Matrix $A \in \mathbb{R}^{n \times n}$
Output: (λ_1, v_1) eigenpair associated to the largest (in module) eigenvalue.
$v \in \mathbb{R}^n$ given and $z = A \cdot v$
$\beta = v^T \cdot z$
repeat
  $v = z / \|z\|$
  $z = A \cdot v$
  $\beta_{old} = \beta$
  $\beta = v^T \cdot z$
until $|\beta - \beta_{old}| \,/\, |\beta_{old}| < \varepsilon$
$\lambda_1 = \beta$ and $v_1 = v$

Using this algorithm, we obtain the following results:

Matrix size (n) & type ($imat$)	***power_v12* function**	***power_v11* function**
$n = 200$ & $imat = 1$	6.500e-01	1.680e+00
$n = 200$ & $imat = 2$	2.000e-02	3.000e-02
$n = 200$ & $imat = 3$	3.000e-02	5.000e-02
$n = 200$ & $imat = 4$	7.300e-01	1.600e+00
$n = 1000$ & $imat = 1$	convergence not achieved	convergence not achieved
$n = 1000$ & $imat = 2$	3.300e-01	3.970e+00
$n = 1000$ & $imat = 3$	1.720e+00	2.139e+01
$n = 1000$ & $imat = 4$	convergence not achieved	convergence not achieved
$n = 50$ & $imat = 1$	1.000e-02	2.000e-02
$n = 50$ & $imat = 2$	0.000e+00	1.000e-02
$n = 50$ & $imat = 3$	1.000e-02	3.000e-02
$n = 50$ & $imat = 4$	1.000e-02	2.000e-02

Figure 2 - Calculation time between *power_v12* and *power_v11* functions

As can be seen, the second algorithm is almost twice faster than the first algorithm for each case. Indeed the multiplication between the matrice and the vector was a long operation in time

so if we compute just once instead of twice, we reduce the calculation time by a factor of two.

Question 3 :

The main drawback of the deflated power method in terms of computing time is the fact that this method computes the whole spectral decomposition of the matrix A, which is a square n by n matrix.
In addition, another problem may be that if the matrix A has several eigenvalues close to each other, deflation may be inefficient and convergence to the remaining eigenvalues will be slow.

2.2 Extending the power method to compute dominant eigenspace vectors

Now, our objective is to extend the power method to compute a block of dominant eigenpairs.

2.2.1 subspace_iter_v0: A basic method to compute a dominant eigenspace

The basic version of the method to compute an invariant subspace associated to the largest eigenvalues of a symmetric matrix A is described in Algorithm 3. This subspace is also called dominant eigenspace.
Given a set of m orthonormal vectors V , the Algorithm 3 computes the eigenvectors associated with the m largest (in module) eigenvalues.

Algorithm 3 Vector power method improved

Input: Symmetric matrix $A \in \mathbb{R}^{n \times n}$, number of required eigenpairs m, tolerance ϵ and *MaxtIter* (max nb of iterations)
Output: m dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}.

Generate a set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$; $k = 0$.
repeat
  $k = k + 1$
  $Y = A \cdot V$
  $H = V^T \cdot A \cdot V$ {ou $H = V^T \cdot Y$}
  Compute $acc = \|A \cdot V - V \cdot H\| / \|A\|$
  $V \leftarrow$ orthonormalization of the columns of Y
until $k >$ MaxIter or $acc \leq \epsilon$
Compute the spectral decomposition $X \cdot \Lambda_{\text{out}} \cdot X^T = H$, where the eigenvalues of H ($\text{diag}(\Lambda_{\text{out}})$) are arranged in descending order of magnitude.
Compute the corresponding eigenspace $V_{\text{out}} = V \cdot X$

This algorithm is very close to Algorithm 1:

- The process is mainly based on iterative products between the matrix A and the columns of V.
- Inside the loop, the matrix H plays the same role as the scalar β in Algorithm

There are however some differences:

- An orthonormalization of the columns of Y is realized at each iteration.
- The relationship between both stopping criteria is not trivial.
- At the end of the loop, the columns of V are not the eigenvectors of A. Additional operations have to be done after convergence to obtain the actual dominant eigenspace of A.

Question 4 :

If we apply the power method algorithm to a set of m vectors instead of matrix we obtain a matrix whose columns are the same and which are equal to the eigenvector associated with the largest eigenvalue. Moreover, we obtain a matrix whose components all converge to the same eigenvalue.

Question 5 :

It is not a problem to compute the whole spectral decomposition of H because, in Algorithm 2, the matrix H is of dimension m by m, while the matrix A is of dimension n by n, which makes the matrix H much smaller than the matrix A. As a result, calculating the spectral decomposition of H is much cheaper in terms of computation time than calculating the spectral decomposition of A.

Question 6 :

View files :

- subspace_iter_v0.m
- test_v0v1.m

2.2.2 subspace_iter_v1: Improved version making use of Raleigh-Ritz projection

Several modifications are needed to make the basic subspace iteration an efficient code.

Algorithm 4 Raleigh-Ritz projection

Input: Matrix $A \in \mathbb{R}^{n \times n}$ and an orthonormal set of vectors V.
Output: The approximate eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}.
Compute the Rayleigh quotient $H = V^T \cdot A \cdot V$.
Compute the spectral decomposition $X \cdot \Lambda_{out} \cdot X^T = H$, where the eigenvalues of H ($diag(\Lambda_{out})$) are arranged in descending order of magnitude.
Compute $V_{out} = V \cdot X$.

The algorithm of the subspace_iter_v1 is then :

Algorithm 5 Subspace iteration method v1 with Raleigh-Ritz projection

Input: Symmetric matrix $A \in \mathbb{R}^{n \times n}$, tolerance ε, $MaxIter$ (max nb of iterations) and $PercentTrace$ the target percentage of the trace of A
Output: n_{ev} dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}.

Generate an initial set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$; $k = 0$; $PercentReached = 0$
repeat
  $k = k + 1$
  Compute Y such that $Y = A \cdot V$
  $V \longleftarrow$ orthonormalisation of the columns of Y
  Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V
  Convergence analysis step: save eigenpairs that have converged and update $PercentReached$
until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$)

Question 7 :

In the file subspace_iter_v1.m, the lines which correspond to the steps of the Subspace iteration method with Rayleigh-Ritz projection above, are :

- Generate an initial set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$; $k = 0$; $PercentReached = 0$
  – *lines* 38, 40, 48 *and* 49
- $k = k + 1$
  – *line* 54
- Compute Y such that $Y = A \cdot V$
  – *line* 56
- $V \longleftarrow$ orthonormalisation of the columns of Y
  – *line* 58
- *Rayleigh-Ritz projection* applied on matrix A and orthonormal vectors V
  – *lines* 58
- *Convergence analysis step*: save eigenpairs that have converged and update $PercentReached$
  – *lines* 87 *to* 100

2.3 subspace_iter_v2 and subspace_iter_v3: Toward an efficient solver

Two ways of improving the efficiency of the solver are proposed. Our aim is to build an algorithm that combines both the block approach and the deflation method in order to speed-up the convergence of the solver.

Question 8 :

Given that A is a square matrix of dimension n by n, the cost in terms of flops of the computation of A^p if we using the classic matrix product would be $O(n^3p)$ because the cost of one matrix multiplication n by n by an other is $O(n^3)$ and we would need to realise $p-1$ matrix multiplications.
Now, if we multiply the matrix A which is a matrix of dimension n by n with a matrix V which is a matrix of dimension n by m the cost in terms of flops would be $O(n^2m)$. Then, if we have to compute A^p using the classic matrix product, the cost in terms of flops for $A^p \cdot V$ would be $O(n^3p + n^2m)$ which in the case where $m << n$ would be $O(n^3p)$.
To reduce this cost, we propose first of all to multiply the matrix A (dimension n by n) with the matrix V (dimension n by m) and then multiply $p-1$ times by A on the left side. In the case where $m << n$, this would reduce the cost in terms of flops since we will have $p-1$ n by n matrix product by n by m matrix, which would give a cost in terms of flops of $O(pn^2m)$ which is inferior to $O(n^3p)$.

Question 9 :

View files :

- subspace_iter_v1.m
- test_v0v1v2.m

Question 10 :

As p increases, this leads to faster convergence of the subspace iteration method. Indeed, increasing p provides a better approximation of the dominant eigenvalues and therefore increases the speed of convergence.
However, if p is too large the algorithm does not converge anymore. This can be explained by the fact that increasing p also increases the computational cost of each iteration, since $A^p \cdot V$ requires p matrix multiplications, which can be very costly for large matrices.
We must therefore choose p in such a way that it is efficient in terms of computation efficiency but also in terms of convergence speed.

Question 11:

The quality of each eigenpairs differs for some of the vectors because of initial conditions. Indeed, if the initial set of vectors is well-chosen and represents a significant portion of the eigenspace, then the quality of the eigenpairs will be high and conversely, if the initial set of vectors does not represent a significant portion of the eigenspace, the convergence will be slower and the quality of eigenpairs will be less important

Question 12:

We suppose that the accuracy of the eigenpairs will be lower compared to the previous methods. This is due to the fact that deflation leads to a loss of information on the previously calculated eigenvectors and therefore reduce this accuracy.

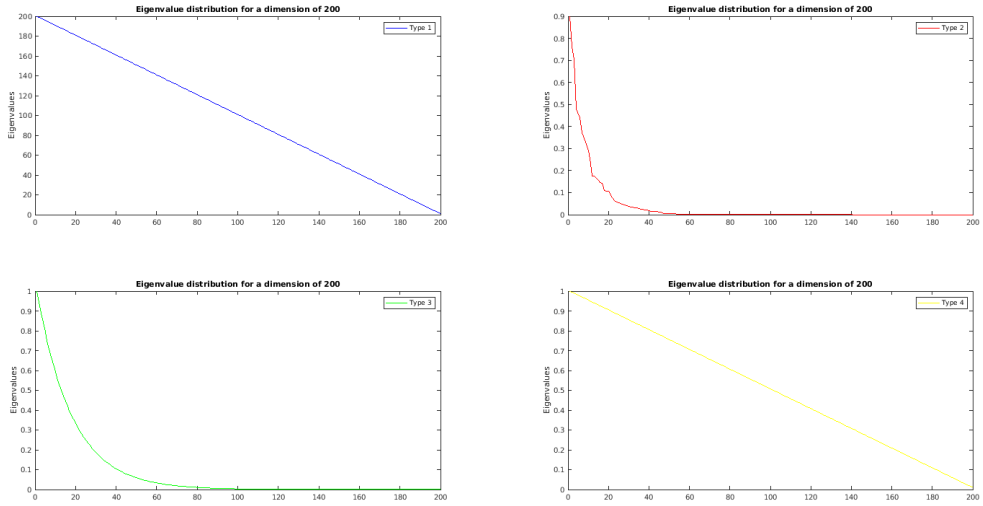
Question 13:

View files :

- subspace_iter_v3.m
- test_v0v1v2v3.m

2.4 Numerical experiments

Question 14:



Here are the characteristics of the 4 types of matrix :

- Type 1 : these matrices have specific eigenvalues : $\lambda_i = i$

- Type 2 : these matrices have random eigenvalues of high conditioning. Their logarithmic value is uniformly distributed : $\lambda_i = random(\frac{1}{\text{cond}}, 1)$ and cond $= 1e^{10}$

- Type 3 : these matrices have random eigenvalues with medium conditioning. Their logarithmic value is uniformly distributed : $\lambda_i = cond^{(\frac{1-i}{n-1})}$ and cond $= 1e^{5}$

- Type 4 : these matrices have random eigenvalues of low conditioning :
$\lambda_i = (n-i) * \frac{1}{n-1} * (1 - \frac{1}{cond}) + \frac{1}{cond}$ avec cond $= 1e^{2}$

Question 15:

Matrix size (n) & type ($imat$)	***subspace_iter_v0* function**
$n = 200$ & $imat = 1$	1.940e+00
$n = 200$ & $imat = 2$	7.000e-02
$n = 200$ & $imat = 3$	1.600e-01
$n = 200$ & $imat = 4$	1.310e+00
$n = 1000$ & $imat = 1$	2.293e+01
$n = 1000$ & $imat = 2$	1.460e+00
$n = 1000$ & $imat = 3$	2.790e+00
$n = 1000$ & $imat = 4$	2.331e+01
$n = 50$ & $imat = 1$	2.000e-02
$n = 50$ & $imat = 2$	0.000e+00
$n = 50$ & $imat = 3$	1.000e-02
$n = 50$ & $imat = 4$	2.000e-02

Figure 3 - Calculation time of *subspace_iter_v0* function

Matrix size (n) & type ($imat$)	***subspace_iter_v1* function**
$n = 200$ & $imat = 1$	3.400e-01
$n = 200$ & $imat = 2$	4.000e-02
$n = 200$ & $imat = 3$	2.000e-02
$n = 200$ & $imat = 4$	2.500e-01
$n = 1000$ & $imat = 1$	7.670e+00
$n = 1000$ & $imat = 2$	2.500e-01
$n = 1000$ & $imat = 3$	4.600e-01
$n = 1000$ & $imat = 4$	7.560e+00
$n = 50$ & $imat = 1$	1.000e-02
$n = 50$ & $imat = 2$	0.000e+00
$n = 50$ & $imat = 3$	0.000e+00
$n = 50$ & $imat = 4$	1.000e-02

Figure 4 - Calculation time of *subspace_iter_v1* function

Matrix size (n) & type ($imat$)	***subspace_iter_v2* function**
$n = 200$ & $imat = 1$	2.500e-01
$n = 200$ & $imat = 2$	4.000e-02
$n = 200$ & $imat = 3$	4.000e-02
$n = 200$ & $imat = 4$	2.400e-01
$n = 1000$ & $imat = 1$	7.400e+00
$n = 1000$ & $imat = 2$	2.500e-01
$n = 1000$ & $imat = 3$	5.000e-01
$n = 1000$ & $imat = 4$	7.620e+00
$n = 50$ & $imat = 1$	2.000e-02
$n = 50$ & $imat = 2$	1.000e-02
$n = 50$ & $imat = 3$	1.000e-02
$n = 50$ & $imat = 4$	2.000e-02

Figure 5 - Calculation time of *subspace_iter_v2* function

Matrix size (n) & type ($imat$)	***subspace_iter_v3* function**
$n = 200$ & $imat = 1$	2.800e-01
$n = 200$ & $imat = 2$	8.000e-02
$n = 200$ & $imat = 3$	5.000e-02
$n = 200$ & $imat = 4$	2.600e-01
$n = 1000$ & $imat = 1$	3.840e+00
$n = 1000$ & $imat = 2$	1.400e-01
$n = 1000$ & $imat = 3$	3.200e-01
$n = 1000$ & $imat = 4$	4.060e+00
$n = 50$ & $imat = 1$	0.000e+00
$n = 50$ & $imat = 2$	0.000e+00
$n = 50$ & $imat = 3$	0.000e+00
$n = 50$ & $imat = 4$	1.000e-02

Figure 6 - Calculation time of *subspace_iter_v3* function

For the *eig* function, the *power_v11* function and the *power_v12* function :
$\rightarrow$ *see figure 1 and 2*

3 Image Compression

Question 1 :

The size of the elements of the triplet (Σ_k, U_k, V_k) is k *by* k for Σ_k, q *by* k for U_k and p *by* k for V_k.

Question 2 :

Here are the different calculation times for the different methods :

Method	***eps = 1e-8,maxit = 10000, search_space = 400, percentage = 0.995, puiss = 1***
SVD	0.19
eig	0.07
subspace_iter_v0	26.338420
subspace_iter_v1	1.02
subspace_iter_v2	1.07
subspace_iter_v3	1.02

Figure 7 - Calculation time (seconds) of the image reconstruction for *Asterix 0*

Method	***eps = 1e-8,maxit = 10000, search_space = 400, percentage = 0.995, puiss = 1***
SVD	0.082
eig	0.044
subspace_iter_v0	14,2
subspace_iter_v1	0.55
subspace_iter_v2	0.53
subspace_iter_v3	0.59

Figure 8 - Calculation time (seconds) of the image reconstruction for *Asterix 1*

Method	***eps = 1e-8,maxit = 10000, search_space = 400, percentage = 0.995, puiss = 1***
SVD	0.04
eig	0.01
subspace_iter_v0	x
subspace_iter_v1	x
subspace_iter_v2	x
subspace_iter_v3	x

Figure 9 - Calculation time (seconds) of the image reconstruction for *Asterix colored*

Method	***eps = 1e-8,maxit = 10000, search_space = 400, percentage = 0.995, puiss = 1***
SVD	3,9
eig	0.98
subspace_iter_v0	x
subspace_iter_v1	9.93
subspace_iter_v2	9.96
subspace_iter_v3	10.02

Figure 10 - Calculation time (seconds) of the image reconstruction for *Spirou*

Method	***eps = 1e-8,maxit = 10000, search_space = 400, percentage = 0.995, puiss = 1***
SVD	17.9
eig	6.36
subspace_iter_v0	x
subspace_iter_v1	x
subspace_iter_v2	x
subspace_iter_v3	x

Figure 11 - Calculation time (seconds) of the image reconstruction for *Thorgal*

Parameters	**subspace_iter_v3**
eps = 1e-8, maxit = 10000, search space = 400, percentage = 0.995, puiss = 1	1.02
eps = 1e-8, maxit = 10000, search space = 800, percentage = 0.995, puiss = 1	1.94
eps = 1e-8, maxit = 10000, search space = 400, percentage = 0.995, puiss = 2	0.66
eps = 1e-8, maxit = 10000, search space = 400, percentage = 0.8, puiss = 1	0.38

Figure 12 - Calculation time (seconds) of the image reconstruction for *Asterix 0*

We have not included the results of the power_v11 and power_v12 functions because they were not powerful enough for image reconstruction. In addition, we can see that the subspace_iter_v0 method is not efficient because all the calculation times are very long (within 20 seconds) so the improvements on versions 1 to 3 are efficient. Next, the subspace iteration methods are successful for black and white images but not for colour images. Indeed, they do not converge, or converge too slowly, for coloured Asterix and Thorgal. This is because the matrices whose eigenvalues we are looking for are less regular than those in black and white and because their elements take on more distant values.

When we doubled the search space for proper pairs, the time almost doubled, which is logical given that searching is what takes the longest. Furthermore, when the power is increased by a factor of 2, the time is reduced because the algorithm converges faster with the squared matrix. However, we couldn't increase the power too much, otherwise the calculation would no

longer converge. Finally, with a lower percentage, the algorithm converges faster because we are less demanding on precision.