Rapport de projet de Calcul Scientifique : Méthodes d'itération de sous-espace (Partie 1 et 2)

FRESCO Alan - HUANG Julien

Département Sciences du Numérique - Première année
2022-2023

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet est d'utiliser une nouvelle méthode, autre que celle vue en CTD (la méthode de la puissance itérée), pour calculer les plus grands couples de vecteurs et valeurs propres d'une matrice, en valeur absolue. Il s'agit de la méthode d'itération de sous-espace, qui s'appuie sur un objet appelé le quotient de Rayleigh. Nous allons ainsi analyser quatre variantes de cette méthode.

2 Les limites de la méthode de la puissance itérée

La méthode de la puissance itérée basique, qui a été introduite en cours de Calcul Scientifique, est rappelée dans l'Algorithme 1 ci-dessous ; il peut être utilisé pour déterminer le vecteur propre associé à la plus grande valeur propre (en valeur absolue).

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
$\beta = v^T.A.v$
repeat
  $y = A.v$
  $v = y/\|y\|$
  $\beta_{old} = \beta$
  $\beta = v^T.A.v$
until $|\beta - \beta_{old}| - |\beta_{old}| < \epsilon$
$\lambda_1 = \beta$ and $v_1 = v$

En ajoutant le processus de déflation, nous sommes en mesure de calculer toutes les paires propres dont nous avons besoin pour atteindre un certain pourcentage de la trace de A.

Une version Matlab de la méthode de puissance de base avec déflation est fournie dans le fichier power_v11.m avec un calcul différent du critère d'arrêt.

On se fixe ici le pourcentage de la trace à atteindre à 40%.

QUESTION 1 :

Taille fixée à 128×128 :

imat	1	2	3	4
eig (10)	9.000e-02	2.000e-02	3.000e-02	9.000e-02
powerMethod (11)	1.460e+01	2.000e-01	5.000e-01	1.588e+01

Taille fixée à 256×256 :

imat	1	2	3	4
eig (10)	7.000e-02	6.000e-02	5.000e-02	8.100e-02
powerMethod (11)	6.560e+01	7.900e-01	1.900e+00	6.950e+01

Taille fixée à 512×512 :

imat	1	2	3	4
eig (10)	3.000e-01	2.800e-01	2.700e-01	2.900e-01
powerMethod (11)	7.360e+02	1.226e+01	1.866e+01	7.338e+02

Taille fixée à 1024×1024 :

imat	1	2	3	4
eig (10)	1.590e+00	1.210e+00	1.290e+00	1.200e+00
powerMethod (11)	3.940e+03	6.339e+01	1.349e+02	3.900e+03

On constate que notre première méthode (power_v11) atteint vite ses limites. En effet, le temps de calcul devient rapidement très élévé quand on augmente la taille des matrices. De plus, cette méthode perd beaucoup en efficacité pour certains types de matrice (1 et 4), en particulier les matrices de grandes tailles.

QUESTION 2 :

On remarque que dans l'algorithme 1, il y a deux produits $A.v$ dans la boucle. Nous avons amélioré le programme de telle sorte à ne laisser qu'un unique produit $A.v$ dans la boucle, comme suit :

Algorithm 1bis Vector power method (upgrade)

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
$z = A.v$
$\beta = v^T.z$
repeat
  $v = z/\|z\|$
  $z = A.v$
  $\beta_{old} = \beta$
  $\beta = v^T.z$
until $|\beta - \beta_{old}| - |\beta_{old}| < \epsilon$
$\lambda_1 = \beta$ and $v_1 = v$

Taille fixée à 128×128 :

imat	1	2	3	4
powerMethod (11)	1.460e+01	2.000e-01	5.000e-01	1.588e+01
powerMethod (12)	7.940e+00	1.700e-01	4.000e-01	8.588e+00

Taille fixée à 256×256 :

imat	1	2	3	4
powerMethod (11)	6.560e+01	7.900e-01	1.900e+00	6.950e+01
powerMethod (12)	3.309e+01	5.500e-01	1.260e+00	3.32e+01

Taille fixée à 512×512 :

imat	1	2	3	4
powerMethod (12)	7.360e+02	1.226e+01	1.866e+01	7.338e+02
powerMethod (12)	3.670e+02	6.420e+00	9.540e+00	3.956e+02

Taille fixée à 1024×1024 :

imat	1	2	3	4
powerMethod (11)	3.940e+03	6.339e+01	1.349e+02	3.900e+03
powerMethod (12)	1.948e+03	3.094e+01	6.955e+01	1.853e+03

QUESTION 3 :

En comparaison à la version précédente, on constate que le temps de calcul a été divisé par deux. Ce résultat se voit tout particulièrement sur les matrices de grande taille.

Mathématiquement, cet algorithme reste identique à la version précédente, cependant, elle fournit de bien meilleurs résultats en termes de complexité de calculs. On préférera donc cette version à la précédente.

Toutefois, cette méthode reste très peu efficace en termes de temps de calcul. En effet, pour une matrice trop grande, ce temps reste beaucoup trop long. En outre, cet algorithme devient même inopérant pour certains types de matrice.

C'est pourquoi nous allons nous intéresser à d'autres méthodes.

3 Extension de la méthode de la puissance itérée pour calculer l'espace dominant des valeurs propres des vecteurs

3.1 subspace_iter_v0 : une méthode basique pour calculer l'espace dominant des valeurs propres

La version de base de la méthode pour calculer un sous-espace invariant associé aux plus grandes valeurs propres d'une matrice symétrique A est décrite dans l'algorithme 2.

Ce sous-espace est également appelé espace propre dominant. Étant donné un ensemble de m vecteurs orthonormaux V, l'algorithme 2 calcule les vecteurs propres associés aux m plus grandes (en module) valeurs propres.

Algorithm 2 Subspace iteration method v0 : the basic version

Input : Symmetric matrix $A \in \mathbb{R}^{n \times n}$, nulber of required eigenpairs m, tolerance ϵ and $MaxtIter$ (max nb of iterations)
Output : m dominant eigenvectirs 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**
$k = k + 1$
$Y = A.V$
$H = V^T.A.V$ {ou $H = V^T.Y$}
Compute $acc = \|A.V - V.H\| \,/\, \|A\|$
$V \leftarrow$ orthonormalisation of the columns of Y
until ($k > Maxiter$ **or** $acc \leq \epsilon$)
Compute the spectral decomposition $X.\Lambda_{out}.X^T = H$, where the eigenvalues of H ($diag(\Lambda_{out})$) are arranged in descending order of magnitude.
Compute the corresponding eigenspace $V_{out} = V.X$

Cet algorithme est très proche de l'algorithme 1 :

- Le processus est principalement basé sur des produits itératifs entre la matrice A et les colonnes de V.
- À l'intérieur de la boucle, la matrice H joue le même rôle que le scalaire β dans l'algorithme 1.

Il y a cependant quelques différences :

- Une orthogonalisation des colonnes de Y est réalisée à chaque itération,
- La relation entre les deux critères d'arrêt n'est pas triviale,
- À la fin de la boucle, les colonnes de V ne sont pas les vecteurs propres de A. Des opérations supplémentaires doivent être effectuées après la convergence pour obtenir le sous-espace propre dominant réel de A.

3.1.1 Orthonormalisation

La première question que vous pouvez vous poser est : « Pourquoi ne pas simplement appliquer l'Algorithme 1 sur une matrice de m vecteurs initiaux (matrice V) au lieu d'un seul vecteur (v) ? ».

En réalité, si l'on étend l'Algorithme 1 pour itérer sur une telle matrice, alors il ne tendra pas vers le résultat attendu, c'est-à-dire que V ne converge pas vers une matrice dont les colonnes contiennent m vecteurs propres différents de A.

Pour éviter ce phénomène, une orthogonalisation des colonnes de Y est réalisée à la fin de chaque itération dans l'Algorithme 2. Le nouvel ensemble V est le résultat de cette orthogonalisation.

QUESTION 4 :

Si l'on venait à appliquer l'Algorithme 1 sur m vecteurs, la matrice V va converger vers une matrice qui va contenir m vecteurs colinéaires et associés à la plus grande valeur propre. En effet, la méthode de la puissance itérée permet, à partir d'une matrice A, d'estimer sa valeur propre la plus grande en module et le vecteur propre qui lui est associé.
Voici un exemple avec $m = 3$, et où A est une matrice de taille 3×3 (voir le fichier matlab exo4.m) :

$$A = \begin{bmatrix} 2.408 & -0.347 & 0.542 \\ -0.347 & 2.295 & -0.461 \\ 0.542 & -0.461 & 1.297 \end{bmatrix}$$

$$V_{\text{power}} = \begin{bmatrix} 0.703 & -0.703 & -0.703 \\ -0.598 & 0.598 & 0.598 \\ 0.385 & -0.385 & -0.385 \end{bmatrix}$$

$$V_{\text{eig}} = \begin{bmatrix} 0.294 & 0.648 & -0.703 \\ -0.250 & 0.762 & 0.598 \\ -0.923 & 0 & -0.385 \end{bmatrix}$$

$$D = \begin{bmatrix} 1.000 & 0 & 0 \\ 0 & 2.000 & 0 \\ 0 & 0 & 3.000 \end{bmatrix}$$

On notera :

- V_{power} est obtenue avec l'Algorithme 1
- V_{eig} est obtenue avec la fonction eig de Matlab.
- D correspond aux valeurs propres d'une matrice A.

Cet exemple illustre bien le problème : appliquer l'Algorithme 1 sur une matrice de m vecteurs initiaux (matrice V) ne suffit pas.

Ainsi, il ne faut pas oublier la méthode de déflation qui permet d'éviter ce problème.

3.1.2 Critère d'arrêt

Une autre difficulté pour adapter l'algorithme 1 afin de calculer des blocs de paires propres à la fois est le critère d'arrêt, car un ensemble de paires propres doit être testé pour la convergence et non pas seulement un vecteur.

Le critère d'arrêt actuel dans l'algorithme 1 repose sur la stagnation de la valeur propre calculée (il teste le fait que la valeur propre calculée ne change plus « beaucoup »). Ce choix a été fait car il est peu coûteux en termes de calcul. Mais cela ne prend pas en compte l'invariance du vecteur propre qui est numériquement plus significative.

Il convient de noter que dans l'Algorithme 1, nous avons atteint la convergence une fois que $v = x_1$ et $\beta = \lambda_1$, et donc $A.v = \beta.v$. Ainsi, un critère d'arrêt plus significatif pour l'Algorithme 1 consiste à calculer l'invariance relative de la valeur propre, qui peut être estimée comme $\|A.v - \beta.v\| \,/\, \|\beta\|$, car la norme de v est égale à 1.

Dans l'Algorithme 2, nous supposons que nous avons convergé de telle sorte que $AV = VH$. Ainsi, dans notre contexte, une mesure possible de la convergence pourrait être : $\|AV - VH\| \,/\, \|A\|$. La norme de Frobenius peut être utilisée pour calculer la norme du numérateur et du dénominateur.

3.1.3 Quotient de Rayleigh

Une fois que la convergence est atteinte dans l'Algorithme 2, V ne contient pas de vecteurs propres de A. Mais des informations spectrales sur A peuvent être extraites de H. En effet, la boucle de l'algorithme 2 a convergé une fois que $AV = VH$. Ainsi, certains couples propre d'A peuvent être facilement obtenus à partir de couples propres de H : si (λ, x) est un couple propre de H, alors $(\lambda, y = Vx)$ est un couple propre de A^2. Pour une matrice symétrique A et une matrice V dont les colonnes sont orthonormales, une matrice $H = V^T AV$ est appelée quotient de Rayleigh. Il jouera un rôle crucial dans les derniers algorithmes.

QUESTION 5 :

Dans cette nouvelle méthode, au lieu de calculer toute la décomposition spectrale de A, on se limite à calculer uniquement celle de H. En effet, H est une matrice de taille $m \times m$, donc le calcul de sa décomposition spectrale sera moins coûteux que celui de A, qui est une matrice de plus grande dimension (A est de taille $n \times n$, où $n > m$).

De plus, nous pouvons facilement obtenir les vecteurs propres et les valeurs propres correspondants pour A en utilisant une simple transformation linéaire.

QUESTION 6 :

Voir les fichiers Matlab suivants :

- subspace_iter_v0.m
- test_v0v1.m

Pour les tests, on obtient :

Taille fixée à 128×128 :

imat	1	2	3	4
v0	4.410e+00 m = 29	2.000e-02 m = 3	1.400e-01 m = 6	5.350e+00 m = 30
v1	1.100e-01 m = 128	4.000e-02 m = 30	1.000e-01 m = 50	1.100e-01 m = 128

Taille fixée à 512×512 :

imat	1	2	3	4
v0	1.243e+02 m = 116	2.400e+00 m = 11	5.480e+00 m = 23	1.340e+02 m = 117
v1	2.053e+00 m = 512	3.300e-01 m = 100	6.500e-01 m = 100	2.280e+00 m = 512

Taille fixée à 1024×1024 :

imat	1	2	3	4
v0	1.230e+03 m = 231	1.533e+01 m = 23	2.634e+01 m = 46	1.300e+03 m = 232
v1	1.097e+01 m = 1024	2.580e+00 m = 231	5.010e+00 m = 231	1.200e+01 m = 1024

Pour la méthode subspace_iter_v0, la valeur m (nombre de valeurs propres cherchées) dépend du pourcentage de la trace qu'on cherche (tout au long du rapport on fixe le pourcentage de la trace à déterminer à 40%) .

Et pour la méthode subspace_iter_v1, la valeur m est la taille du sous-espace considéré, il faut qu'elle soit plus grande que n_ev le nombre de vecteurs propres à calculer, et elle dépend de la forme de la matrice A (si ses valeurs propres sont uniformément réparties ou non).

Conclusion : la méthode subspace_iter_v1 est meilleure que subspace_iter_v0.
Regardons si l'on peut faire mieux.

3.2 subspace_iter_v1 : une version améliorée utilisant la projection de Rayleigh-Ritz

Plusieurs modifications sont nécessaires pour transformer l'itération de sous-espace de base en un code efficace.

3.2.1 Premières améliorations

Algorithm 3 Rayleigh-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 $H = V^T.A.V$.
Compute the spectral decomposition $X.\Lambda_{out}.X^T = H$, where the eigenvalues of H ($diag(\Lambda_{out})$) are arranged in descending order of magnitude.
Compute $V_{out} = V.X$.

L'algorithme de subspace_iter_v1 est donc :

Algorithm 4 Subspace iteration method v1 with Rayleigh-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 (lignes 48 et 49)
$V \in \mathbb{R}^{n \times m}$;
$k = 0$; (ligne 38)
$PercentReached = 0$ (ligne 40)
repeat
  $k = k + 1$ (ligne 54)
  Compute Y such that $Y = A.V$ (ligne 56)
  $V \leftarrow$ orthonormalisation of the columns of Y (ligne 58)
  Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V (ligne 61)
  Convergence analysis step (lignes 70 à 109) : save eigenpairs that have converged (ligne 92) and update $PercentReached$ (ligne 97)
until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$) (ligne 52)

3.2.2 Étape de l'analyse de convergence

Une étape d'analyse de convergence est effectuée immédiatement après l'étape de projection de Rayleigh-Ritz ; son objectif est de déterminer quelles vecteurs propres ont convergé à l'itération courante k.
Nous considérons que le vecteur propre j, stocké dans la j^{me} colonne de V, a convergé lorsque :

$$\|r_j\| = \|A.V_j - \Lambda_j.V_j\| \,/\, \|A\| \leq \epsilon$$

.

La théorie de convergence stipule que les vecteurs propres correspondant aux plus grandes valeurs propres convergent plus rapidement que ceux correspondant aux plus petites valeurs propres. Pour cette raison, nous devrions tester la convergence des vecteurs propres dans l'ordre $j = 1, 2, ...$:

- nous considérons que nous ne convergions pas à cette itération avec le premier vecteur à ne pas passer le test,
- il n'est pas non plus utile de tester à nouveau les vecteurs qui ont convergé lors de l'itération précédente.

QUESTION 7 :

Dans l'algorithme 4, nous avons indiqué les lignes de code de subspace_iter_v1.m correspondant à chacune des étapes de l'algorithme.

4 subspace_iter_v2 et subspace_iter_v3 : vers un solveur efficace

Deux moyens d'améliorer l'efficacité du solveur sont proposés. Notre objectif est de construire un algorithme qui combine à la fois l'approche par blocs et la méthode de déflection afin d'accélérer la convergence du solveur.

4.1 L'approche par bloc (subspace_iter_v2)

L'orthonormalisation est effectuée à chaque itération et est assez coûteuse. Une façon simple d'accélérer l'approche consiste à effectuer p produits à chaque itération (remplacer $V = AV$ (première étape de l'itération) par $V = ApV$). Notez que cette méthode d'accélération très simple est applicable à toutes les versions de l'algorithme.

QUESTION 8 :

Pour calculer A^p, il faut p multiplications matricielles. Or, une multiplication matricielle demande n^3 multiplications scalaires. Donc, A^p en demande $O(n^3 p)$, et $A^p V$ en demande quant à lui $O(n^3 pm)$, car V est de taille $n \times m$.
On peut diminuer le nombre de calculs avec l'Algorithme "exponentiation by squaring" comme illustré ci-dessous :

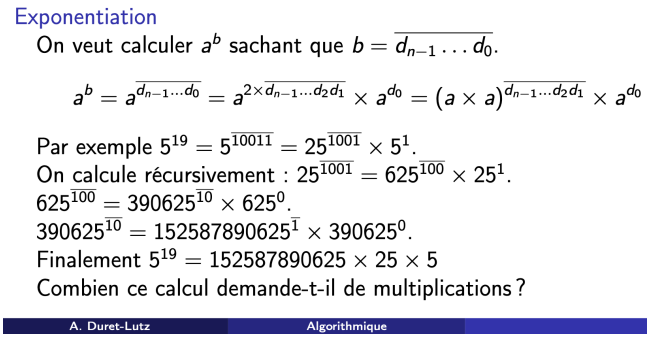

FIGURE 1 – Illustration de la méthode

Le coût des calculs pour A^p s'élève donc à $O(n^3 log(p))$ opérations élémentaires.

QUESTION 9 :

Voir les fichiers Matlab suivants :

- subspace_iter_v2.m
- test_v0v1v2.m

QUESTION 10 :

Cependant, on observe que lorsque l'on augmente p (la puissance), le temps de calcul de la méthode $v2$ devient alors meilleure que celle de la méthode $v1$. Toutefois, si l'on augmente un peu trop p la méthode $v2$ devient alors moins efficace que la méthode $v1$.

Cela s'explique dans un premier temps : en augmentant p, on diminue le nombre d'itérations, donc on diminue le nombre d'orthonormalisations, ce qui nous permet de gagner un temps considérable. Mais si l'on augmente trop p, les puissances de matrice deviennent trop « lourdes »comme elles ont un coût de $O(n^3 log(p))$ opérations élémentaires, et le calcul de A^p perd de ce fait en rentabilité.

Il faut donc une valeur de p suffisamment grande pour accélérer la convergence de l'algorithme, mais pas trop grande pour éviter d'augmenter considérablement le coût de calcul à chaque itération. Nous devons donc trouver un entre-deux.

Observation des résultats avec des puissances optimales :

On fixe un pourcentage de 40%.

Pour les tests, on obtient :

Taille fixée à 128×128 :

imat	1	2	3	4
v0	4.410e+00 m = 29	2.000e-02 m = 3	1.400e-01 m = 6	5.350e+00 m = 30
v1	1.100e-01 m = 128	4.000e-02 m = 30	1.000e-01 m = 50	1.100e-01 m = 128
v2	1.100e-01 m = 128 p = 1	6.000e-02 m = 30 p = 2	7.000e-02 m = 50 p = 3	1.100e-01 m = 128 p = 1

Taille fixée à 512×512 :

imat	1	2	3	4
v0	1.243e+02 m = 116	2.400e+00 m = 11	5.480e+00 m = 23	1.340e+02 m = 117
v1	2.053e+00 m = 512	3.300e-01 m = 100	6.500e-01 m = 100	2.280e+00 m = 512
v2	2.090e+00 m = 512 p = 1	5.700e-01 m = 100 p = 5	8.600e-01 m = 100 p = 6	2.360e+00 m = 512 p = 1

Taille fixée à 1024×1024 :

imat	1	2	3	4
v0	1.230e+03 m = 231	1.533e+01 m = 23	2.634e+01 m = 46	1.300e+03 m = 232
v1	1.097e+01 m = 1024	2.580e+00 m = 231	5.010e+00 m = 231	1.200e+01 m = 1024
v2	1.163e+01 m = 1024 p = 1	3.800e+00 m = 231 p = 3	4.700e+00 m = 231 p = 5	1.113e+01 m = 1024 p = 1

On remarque qu'outre la détermination de m (déjà expliquée dans la question 6), la détermination de p dépend, quant à elle, du nombre d'itérations à faire. Par exemple, on observe dans la méthode $v1$, pour $imat = 1$ ou 4, qu'il ne fallait qu'une itération pour trouver les couples propres. On doit donc fixer $p = 1$, sinon cela pourrait engendrer des calculs inutiles.

4.2 La méthode de déflation (subspace_iter_v3)

Comme les colonnes de V convergent dans l'ordre, nous pouvons figer les colonnes de V qui ont convergé. Ce gel permet de réaliser des économies significatives dans les étapes de multiplication matrice-vecteur ($V = AV$), d'orthonormalisation et de projection Rayleigh-Ritz. Plus précisément, supposons que les premières nbc^3 colonnes de V ont convergé, et partitionnons $V = [Vc, Vnc]$ où Vc a nbc colonnes et Vnc a $m - nbc$ colonnes. Ensuite, nous pouvons former la matrice $[Vc, AVnc]$, qui est identique à celle obtenue en multipliant V par A. Cependant, nous devons encore orthogonaliser Vnc par rapport aux vecteurs figés de Vc en l'orthogonalisant d'abord contre Vc puis contre elle-même. Enfin, l'étape de projection Rayleigh-Ritz peut également être limitée aux colonnes Vnc de V.

QUESTION 11 :

Avec la méthode subspace_iter_v1 après chaque itération, il y a une orthogonalisation complète de la matrice contenant les vecteurs propres. Cela a pour effet d'impacter les vecteurs ayant déjà convergé. Il en va de même pour la projection de Rayleigh-Ritz et l'opération $Y \leftarrow A^p * V$, les vecteurs sont recalculés et deviennent donc plus précis (mais cela rajoute du temps de calcul).

QUESTION 12 :

Avec la méthode subspace_iter_v3, après chaque itération, les vecteurs propres ayant déjà convergé ne sont plus modifiés. Donc, après chaque itération, les vecteurs ne convergent pas plus, ils sont figés. Ainsi, la méthode subspace_iter_v3 est moins précise que la méthode subspace_iter_v1 ; elle est cependant plus rapide.

QUESTION 13 :

Voir les fichiers Matlab suivants :

- subspace_iter_v3.m
- test_v0v1v2v3.m

Pour les tests, on obtient :

Taille fixée à 128×128 :

imat	1	2	3	4
v0	4.410e+00 m = 29	2.000e-02 m = 3	1.400e-01 m = 6	5.350e+00 m = 30
v1	1.100e-01 m = 128	4.000e-02 m = 30	1.000e-01 m = 50	1.100e-01 m = 128
v2	1.100e-01 m = 128 p = 1	6.000e-02 m = 30 p = 2	7.000e-02 m = 50 p = 3	1.100e-01 m = 128 p = 1
v3	9.000e-02 m = 128 p = 1	5.000e-02 m = 30 p = 2	6.000e-02 m = 50 p = 3	1.000e-01 m = 128 p = 1

Taille fixée à 512×512 :

imat	1	2	3	4
v0	1.243e+02 m = 116	2.400e+00 m = 11	5.480e+00 m = 23	1.340e+02 m = 117
v1	2.053e+00 m = 512	3.300e-01 m = 100	6.500e-01 m = 100	2.280e+00 m = 512
v2	2.090e+00 m = 512 p = 1	5.700e-01 m = 100 p = 5	8.600e-01 m = 100 p = 6	2.360e+00 m = 512 p = 1
v3	1.950e+00 m = 512 p = 1	5.400e-01 m = 100 p = 5	7.100e-01 m = 100 p = 6	2.330e+00 m = 512 p = 1

Taille fixée à 1024×1024 :

imat	1	2	3	4
v0	1.230e+03 m = 231	1.533e+01 m = 23	2.634e+01 m = 46	1.300e+03 m = 232
v1	1.097e+01 m = 1024	2.580e+00 m = 231	5.010e+00 m = 231	1.200e+01 m = 1024
v2	1.163e+01 m = 1024 p = 1	3.800e+00 m = 231 p = 3	4.700e+00 m = 231 p = 5	1.113e+01 m = 1024 p = 1
v3	1.096e+01 m = 1024 p = 1	3.640e+00 m = 231 p = 3	4.440e+00 m = 231 p = 5	1.098e+01 m = 1024 p = 1

On remarque que le gain de temps entre la méthode v2 et v3 n'est pas significatif.

5 Expérimentations numériques

QUESTION 14 :

Les différences entre les quatre types de matrices sont les suivantes :

— $imat = 1$: La matrice voit ses valeurs propres uniformément réparties entre 1 et n. (voir figure 2)

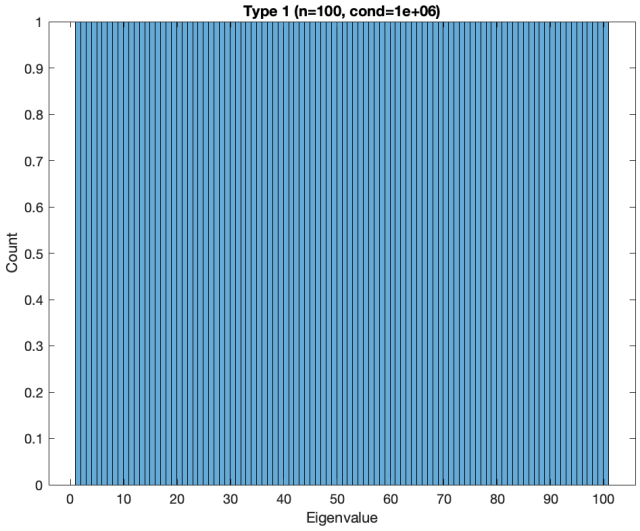

FIGURE 2 – Distribution des valeurs propres de la matrice type 1

— $imat = 2$: La matrice possède des valeurs propres réparties de manière aléatoire entre 0 et 1, principalement centrées en 0. (voir figure 3)

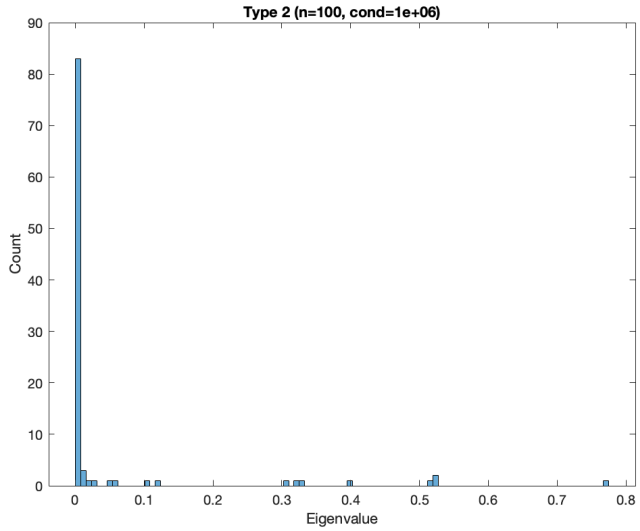

FIGURE 3 – Distribution des valeurs propres de la matrice type 2

— $imat = 3$: Quasi identique à $imat = 2$, mais la répartition n'est pas aléatoire. (voir figure 4)

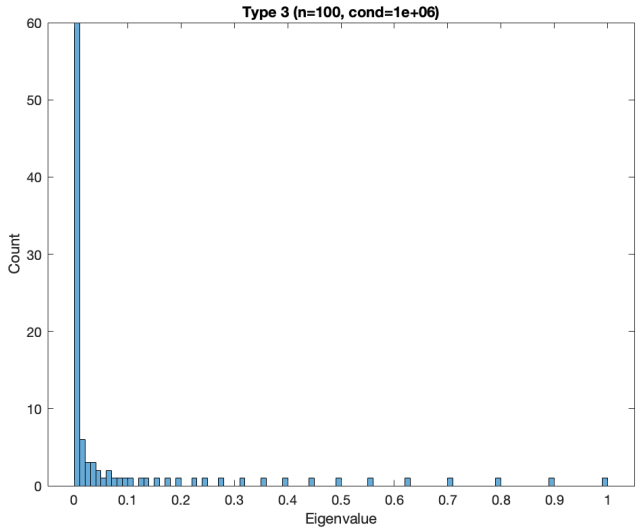

FIGURE 4 – Distribution des valeurs propres de la matrice type 3

— $imat = 4$: Répartition uniforme des valeurs propres entre 0 et 1. (voir figure 5)

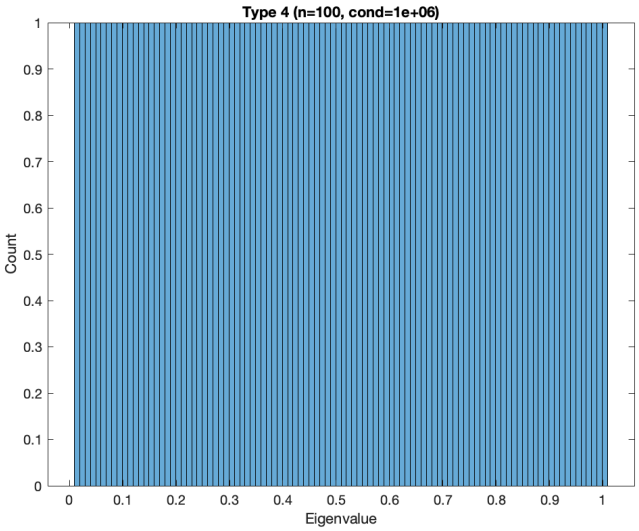

FIGURE 5 – Distribution des valeurs propres de la matrice type 4

QUESTION 15 :

Pour les matrices de petite taille les méthodes se valent. Regardons pour les matrices de moyenne taille et de grande taille quelle est la meilleure méthode entre eig, $v12$, $v0$, $v1$, $v2$, $v3$ pour les quatre types de matrice. (voir la fonction Matlab testall.m)

Théoriquement, subspace_iter_v1 est meilleure que subspace_iter_v0 (grâce à la projection de Rayleigh-Ritz), subspace_iter_v2 est plus rapide que subspace_iter_v1 (avec le calcul de A^p), et subspace_iter_v3 est plus rapide que subspace_iter_v2, mais moins précise car elle gèle les vecteurs ayant déjà convergé, donc il y aura moins de calculs.

— imat 1 :
  — matrice de petite taille : eig, v3, v2
  — matrice de moyenne taille : eig
  — matrice de grande taille : eig
— imat 2 :
  — matrice de petite taille : eig, v0
  — matrice de moyenne taille : eig, v1
  — matrice de grande taille : eig, v2, v3
— imat 3 :

- matrice de petite taille : eig, v2, v3
- matrice de moyenne taille : eig, v1
- matrice de grande taille : eig

— imat 4 :
- matrice de petite taille : eig, v3,v2
- matrice de moyenne taille : eig
- matrice de grande taille : eig

Attention ces résultats sont à prendre avec des pincettes, il y a une marge d'incertitudes dans les temps de calcul.

Nous avons rassemblé ci-dessous les tableaux récapitulant les performances de tous les algorithmes utilisés pour différents types et pour différentes tailles de matrices :

On fixe le pourcentage à 40%.

Taille fixée à 128×128 :

imat	1	2	3	4
eig	9.000e-02	2.000e-02	3.000e-02	9.000e-02
v11	1.460e+01	2.000e-01	5.000e-01	1.588e+01
v12	7.940e+00	1.700e-01	4.000e-01	8.588e+00
v0	4.410e+00 m = 29	2.000e-02 m = 3	1.400e-01 m = 6	5.350e+00 m = 30
v1	1.100e-01 m = 128	4.000e-02 m = 30	1.000e-01 m = 50	1.100e-01 m = 128
v2	1.100e-01 m = 128 p = 1	6.000e-02 m = 30 p = 2	7.000e-02 m = 50 p = 3	1.100e-01 m = 128 p = 1
v3	9.000e-02 m = 128 p = 1	5.000e-02 m = 30 p = 2	6.000e-02 m = 50 p = 3	1.000e-01 m = 128 p = 1

Taille fixée à 512×512 :

imat	1	2	3	4
eig	3.000e-01	2.800e-01	2.700e-01	2.900e-01
v11	7.360e+02	1.226e+01	1.866e+01	7.338e+02
v12	3.670e+02	6.420e+00	9.540e+00	3.956e+02
v0	1.243e+02 m = 116	2.400e+00 m = 11	5.480e+00 m = 23	1.340e+02 m = 117
v1	2.053e+00 m = 512	3.300e-01 m = 100	6.500e-01 m = 100	2.280e+00 m = 512
v2	2.090e+00 m = 512 p = 1	5.700e-01 m = 100 p = 5	8.600e-01 m = 100 p = 6	2.360e+00 m = 512 p = 1
v3	1.950e+00 m = 512 p = 1	5.400e-01 m = 100 p = 5	7.100e-01 m = 100 p = 6	2.330e+00 m = 512 p = 1

Taille fixée à 1024×1024 :

imat	1	2	3	4
eig	1.590e+00	1.210e+00	1.290e+00	1.200e+00
v11	3.940e+03	6.339e+01	1.349e+02	3.900e+03
v12	1.948e+03	3.094e+01	6.955e+01	1.853e+03
v0	1.230e+03 m = 231	1.533e+01 m = 23	2.634e+01 m = 46	1.300e+03 m = 232
v1	1.097e+01 m = 1024	2.580e+00 m = 231	5.010e+00 m = 231	1.200e+01 m = 1024
v2	1.163e+01 m = 1024 p = 1	3.800e+00 m = 231 p = 3	4.700e+00 m = 231 p = 5	1.113e+01 m = 1024 p = 1
v3	1.096e+01 m = 1024 p = 1	3.640e+00 m = 231 p = 3	4.440e+00 m = 231 p = 5	1.098e+01 m = 1024 p = 1

6 Application à la compression d'images

Théorème : Décomposition en valeurs singulières (SVD)

Soit $A \in \mathbb{R}^{q\times p}(p \leq q)$ de rang complet, $rg(A) = p$, alors nous pouvons décomposer A comme $A = U\Sigma V^T$ avec :

- $U \in \mathbb{R}^{q\times q}$ est formée de q vecteurs propres orthogonaux associés à q valeurs propres de AA^T.
- $V \in \mathbb{R}^{p\times p}$ est formée de p vecteurs propres orthogonaux associés à p valeurs propres de A^TA.
- $\Sigma \in \mathbb{R}^{q\times p}$ est une matrice rectangulaire dont les éléments non nuls sur la « diagonale »sont les valeurs singulières σ_i, $i = 1, .., q$ de A en ordre décroissant (elles sont les racines carrées des valeurs propres de A^TA et AA^T).

Nous avons les relations suivantes entre u_i et v_i :

$$\text{pour } i \leq p,\ v_i = \tfrac{1}{\sigma_i}A^Tu_i \text{ et } u_i = \tfrac{1}{\sigma_i}Av_i$$

En traitement d'image, les valeurs singulières représentent l'énergie de l'image. En effet, l'énergie totale de l'image $I \in \mathbb{R}^{q\times p}$ est représentée par la formule suivante :

$$\|I\| = Tr(I^TI) = \sum_{i=1}^{q}\sum_{j=1}^{p} I_{ij}^2 = \sum_{i=1}^{q}\sigma_i^2$$

Théorème : Meilleure approximation de rang faible

Soit $A \in \mathbb{R}^{q\times p}$ dont la SVD est $A = \sum_{i=1}^{p}\sigma_i u_i v_i^T$ avec $p = rg(A)$. Si $k < p$ et $Ak = \sum_{i=1}^{k}\sigma_i u_i v_i^T$ alors A_k est la meilleure approximation de rang faible de A de rang $k < p$, c'est-à-dire :

$$\min_{rg(D)=k} \|A - D\|_F = \|A - A_k\|_F$$

Soit I de taille $q \times p$ la matrice décrivant l'image.
Soit $k < p < q$, nous pouvons reconstruire l'image I en utilisant son approximation de rang k, I_k.
Pour calculer les vecteurs et les valeurs singulières nécessaires pour atteindre cette reconstruction, nous pouvons soit :

1. effectuer la décomposition SVD de I
2. ou, construire $M = II^T$ (ou $M = I^TI$ selon les tailles q et p) et calculer les k paires d'auto-valeurs dominantes de M, puis en utilisant les relations (1), calculer l'autre ensemble de vecteurs (encore une fois selon la taille p et q).

Une fois que nous avons les k valeurs singulières et les premiers k vecteurs de U et V, nous pouvons calculer I_k.
On parle de compression d'image car au lieu d'avoir l'image représentée par un tableau de taille $q \times p$, elle est maintenant un triplet (Σ_k, U_k, V_k) avec moins de valeurs lorsque $k < q < p$.

Remarque : k peut correspondre à un nombre de valeurs propres pour atteindre un pourcentage de la trace.

QUESTION 1 :

Σ_k est de taille $k \times k$, U_k est de taille $q \times k$ et V_k est de taille $p \times k$.

QUESTION 2 :

En analysant la matrice M = $I.I^T$, on remarque que la valeur propre la plus grande (en module) représente 95.94% de la trace. Il faudra donc prendre un pourcentage assez élevé pour pouvoir calculer plus d'une valeur propre (par exemple la méthode power_v11 détermine en premier la valeur propre la plus grande (en module)). Si nous voulons calculer 200 valeurs propres, il faudra prendre un pourcentage au moins égal à 99.55%.

On a également fixé arbitrairement *maxit* comme étant assez grand pour que les vecteurs propres aient le temps de converger.

On a fixé la tolérance **eps** de sorte qu'elle ne soit pas trop petite, sinon les couples propres aurait pris beaucoup de temps a être calculés (si l'on veut plus de précision il faudra augmenter *maxit*).

nb_vp (le nombre de valeurs propre à calculer) dans les méthodes power_v11, power_v12 doit juste être plus grand que 200, on peut même prendre nb_vp = 100000, en effet, cela n'a pas d'importance comme on cherche au moins 200 couples propres.

search_space de la méthode subspace_iter_v0 est le nombre de couples propres à déterminer (ici 200 suffisent).

search_space = 350, dans les méthodes v1, v2, v3. Déterminé par tâtonnement.

	tps de calcul	eps	maxit	search_space	nb_vp (v11, v12)	%	puiss
SVD	1.601775	X	X	X	X	X	X
eig	0.581048	X	X	X	X	X	X
v11	212.929050	1e-8	90000	X	300	99.55	X
v12	101.249649	1e-8	90000	X	300	99.55	X
v0	48.735964	1e-8	90000	200	X	99.55	X
v1	3.840503	1e-8	90000	350	X	99.55	X
v2	3.077127	1e-8	90000	350	X	99.55	2
v3	3.139700	1e-8	90000	350	X	99.55	2

7 Conclusion

Tout le long de ce travail, nous avons vu différentes méthodes de calcul pour déterminer les couples propres d'une matrice. Mais la méthode eig de matlab reste tout de même la plus efficace.

8 Références

- Cours de Calcul Scientifique
- TD de Calcul Scientifique
- TP de Calcul Scientifique