Calcul Scientifique :
Subspace Iteration Methods
Application to image compression

ROURE Mathéo
BROCHETON Damien

Département Sciences du Numérique - Première année
2022-2023

Table des matières

1 Introduction

Ce projet a pour but de nous faire découvrir les méthodes d'itération des sous-espaces (subspace iteration methods) afin de calculer les valeurs propres et les vecteurs propres des matrices carrées. Ensuite, nous utiliserons ces algorithmes pour reconstituer des images.

2 Partie 1 : Subspace Iteration Methods

2.1 Limitations of the power method

Question 1 : Comparaison du temps d'exécution des fonctions power_v11 et eig pour calculer des paires propres de différentes tailles et pour différents types de matrices (en utilisant le fichier test_v11.m).

Taille de la matrice	Type de la matrice	Temps de calcul Eig (en secondes)	Temps de calcul power_v11 (en secondes)
200x200	1	1e-2	7
	2	1,2e-1	1,7
	3	2e-2	2,9e-1
	4	3e-2	7,1
150x150	1	2e-2	3,9
	2	1e-2	6,4e-1
	3	6e-2	4e-1
	4	1e-2	4,5
125x125	1	1e-2	2,3

Le temps d'exécution est beaucoup plus petit avec la méthode eig plutôt que la méthode power_v11, peu importe le type et la taille de matrice. En plus, la méthode eig est plus précise.
Remarque : Pour certains types et tailles de matrice, le temps de calcul des fonctions est comparable (ex : taille 200x200 et type 3), mais en général la méthode eig a un temps de calcul beaucoup plus court (d'ordre x10 ou x100).

Question 2 :
Vector power method
Require: $A \in R^{n \times n}$, $v \in R^n$ (given)
Ensure: (λ_1, v_1) eigenpair associated to the largest (in module) eigenvalue.
1: $z = A \cdot v$
2: $\beta = v \cdot z$
3: **repeat**
4: $v = \frac{z}{|z|}$
5: $y = A \cdot v$
6: $\beta_{old} = \beta$
7: $\beta = vz$
8: **until** $|\beta - \beta_{old}| \,/\, |\beta_{old}| < \varepsilon$
9: $\lambda_1 = \beta$ and $v_1 = v$

Dans le code précédent, on a retiré le calcul de la variable y=A×v à la première ligne de la boucle while. En effet, la variable z est initialisé avec la même formule à l'extérieur de la boucle while et est recalculée à chaque itération de la boucle while. Ainsi, il n'est pas nécessaire de recalculer y dans la première ligne de la boucle while, car z est suffisant pour la suite des calculs. Cela permet d'économiser du temps de calcul et d'améliorer l'efficacité de l'algorithme.

Comparaison du temps d'exécution de la fonction power_v12, qui est une amélioration de power_v11 deux fois plus rapide, pour calculer quelques paires propres pour différentes tailles et différents types de matrices

Taille matrice	Type matrice	power_v11	power_v12
200	1	8,65	2.22
200	2	$3{,}2\times10^{-1}$	$3{,}2\times10^{-1}$
200	3	$2{,}9\times10^{-1}$	1.2×10^{-1}
200	4	9.62	2.73
150	1	4.18	1.39
150	2	6×10^{-2}	4×10^{-2}
150	3	1.9×10^{-1}	1×10^{-1}
150	4	3.78	1.55

Question 3 : La méthode de la puissance déflatée à un principal inconvénient, le temps de calcul. En faisant des produits matrices vecteurs à chaque fois, dans le calcul de A × V, ce qui ralentit considérablement la méthode déflatée.

2.2 Extending the power method to compute dominant eigenspace vectors

L'objectif de cette partie est d'étendre la méthode de la puissance pour calculer un bloc de paires propres dominantes.

2.2.1 Subspace_iter_v0 : a basic method to compute a dominant eigenspace

Question 4 : En appliquant l'algorithme de la méthode des puissances à un ensemble de m vecteurs, il converge vers la matrice dont les vecteurs propres correspondent aux valeurs propres dominantes.

Question 5 : Calculer la décomposition spectrale entière de H en étudiant ses dimensions ne pose pas de problème car la matrice H est de taille m x m, où m est le nombre de vecteurs propres que nous voulons calculer et m est inférieur à n. Ainsi, on cherche à trouver m valeurs propres et les m vecteurs propres correspondants. Ce calcul est moins coûteux que la recherche de la décomposition spectrale complète de A, qui implique la recherche de n valeurs propres et de leurs vecteurs propres correspondants ($\dim(A) = n \times n$).

Question 6 : Fait dans matlab (dans le fichier subspace_iter_v0.m, remplir la fonction pour obtenir l'algorithme 2.)

2.2.2 Subspace_iter_v1 : improved version making use of Raleigh-Ritz projection

Nous allons faire plusieurs modifications sont nécessaires pour faire de l'itération du sous-espace de base un code efficace.

Question 7 : L'identification des étapes de l'algorithme 4 dans subspace_iter_v1 est en annexe car le code est trop long (voir annexe 1).

2.3 Subspace_iter_v2 and subspace_iter_v3 : toward an efficient solver

Nous allons d'améliorer l'efficacité du solveur de deux faons. Nous allons construire un algorithme qui combinant à la fois l'approche par blocs et la méthode de déflation afin d'accélérer la convergence du solveur.

2.3.1 Subspace_iter_v2 : Block approach

Question 8 :
Calcul du coût en opérations de A^p : le calcul de A^p consiste à faire p-1 produits matriciel, ce qui vaut $(p-1)n^3$. La complexité du calcul de A^p est $O(n^3)$.

Calcul du coût en opérations de $A^p \times V$: le calcul de $A^p \times V$ est le calcul de A^p multiplié par la matrice V de taille $n \times m$, ce qui ajoute n^2m opérations. Le coût total est $(p-1)n^3 + n^2m$. La complexité du calcul de A^p est $O(n^3)$.

Pour réduire le coût, nous calculons AV que nous multiplions par A p fois à gauche. Nous avons donc des produits matrice/vecteur au lieu d'avoir des produits matrice/matrice, ce qui est moins coûteux. Le coût est de mpn^2, donc la complexité est $O(n^2)$.

Question 9 : Fait dans matlab (Modification du fichier subspace_iter_v2.m pour implémenter cette accélération).

Question 10 : En augmentant la valeur de p, nous pouvons réduire le nombre d'itérations nécessaires à la convergence. Cependant, l'augmentation de p entraîne une augmentation du coût de calcul par itération. Il faut donc trouver un compromis entre le coût de calcul et le taux de convergence.

Valeur de P	Nombre d'itérations pour converger	Temps (seconde)	Précision
2	339	1.1e0	6.930e-14
4	170	8.1e-1	7.156e-14
6	113	5.2e-1	4.666e-14
8	85	4.0e-1	4.831e-14
10	68	3.7e-1	4.882e-14
12	57	4.2e-1	3.089e-14
14	49	3.1e-1	4.204e-14
16	43	3.7e-1	1.544e-14
18	38	3.9e-1	1.419e-14
20	34	3.8e-1	2.359e-14

2.3.2 Subspace_iter_v3 : Deflation method

Nous voyons bien sur ce tableau qu'en augmentant p, le nombre d'itérations diminue mais le temps de calcul par itération augmente. Nous pouvons aussi voir que la précision s'améliore en augmentant p.

Question 11 : Avec la méthode subspace_iter_v1, la précision diffère pour certaines paires propres car le processus d'orthonormalisation est effectué après plusieurs itérations. Pendant ces itérations, les erreurs d'arrondis s'ajoutent et se reprennent, cela peut entraîner une perte de précision dans les paires propres calculées.

Question 12 : Avec la méthode subspace_iter_v3, la précision des paires propres est satisfaisante pour toutes les paires car les colonnes de V convergent dans l'ordre, et les colonnes déjà convergentes sont gelées. Cela évite les problèmes d'ajouter des erreurs d'imprécision qui affecte la précision des paires propres, et permet d'obtenir une précision satisfaisante pour toutes les paires calculées.

Question 13 : Fait dans matlab (Copie du fichier subspace_iter_v2.m dans le fichier subspace_iter_v3.m pour mettre en œuvre cette déflation).

2.4 Numerical experiments

Question 14 :

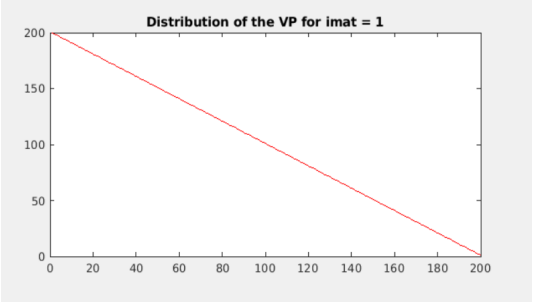

FIGURE 1 – Distribution des valeurs propres pour les matrices de type 1

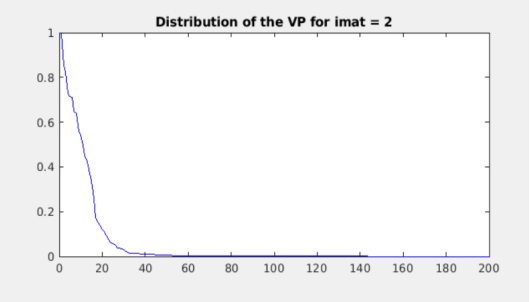

FIGURE 2 – Distribution des valeurs propres pour les matrices de type 2

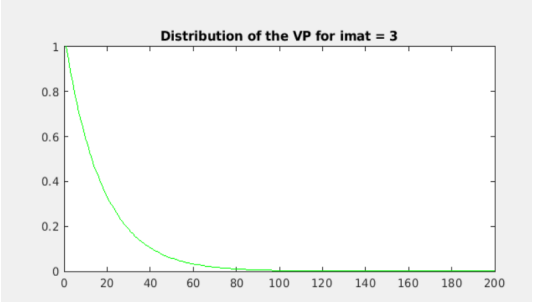

FIGURE 3 – Distribution des valeurs propres pour les matrices de type 3

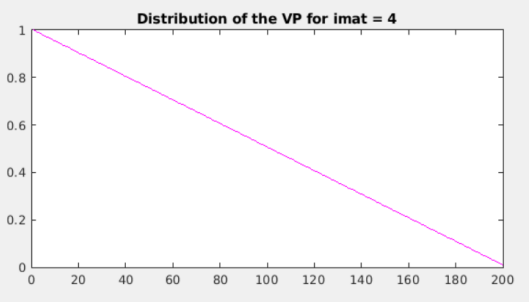

FIGURE 4 – Distribution des valeurs propres pour les matrices de type 4

Les matrices diffèrent principalement par leur spectre. Les matrices de type 1 et 4 ont un spectre uniformément distribué, tandis que les matrices de type 3 ont un spectre plus espacé. Les matrices de type 2 ont quant à elles un spectre distribué de manière aléatoire.

Question 15 :
Le tableau ci-dessous représente les performances en secondes de chaque algorithme pour des matrices de taille et de type différents.

	Type de matrice	Taille de la matrice	
		100	200
Subspace_ iter_v0	1	1.34	3.38
	2	3e-2	1.4e-1
	3	2.1e-1	5.4e-1
	4	1.34	3.14
Subspace_ iter_v1	1	2.3e-1	6.3e-1
	2	2e-2	3e-2
	3	2e-2	5e-2
	4	1.4e-1	6.2e-1
Subspace_ iter_v2	1	2e-2	7e-2
	2	1e-2	1e-2
	3	1e-2	1e-2
	4	1e-2	7e-2
Subspace_ iter_v3	1	2e-2	6e-2
	2	1e-2	1e-2
	3	1e-2	1e-2
	4	2e-2	6e-2
Eig	1	1e-2	2e-2
	2	1e-2	1e-2
	3	0	1e-2
	4	0	2e-2

En regardant ce tableau, nous pouvons conclure que :
- eig est la méthode la plus performante, peu importe le type et la taille de la matrice.
- subspace_iter_v0 est la méthode la plus lente, elle est faite pour être fiable.
- subspace_iter_v1 garde la fiabilité de la v0, mais elle est plus rapide. C'est une amélioration de subspace_iter_v0.
- subspace_iter_v2 est beaucoup plus performante que v0 et que la v1 mais elle est moins fiable. Quand la valeur de p est élevée, la convergence risque de ne pas être atteinte.
- subspace_iter_v3 est une amélioration de v2. Elle garde la rapidité de la v2 mais elle est plus précise.

3 Partie 2 : Application to image compression

Question 1 : La taille des éléments du triplet (Σ_k, U_k, V_k) est
$\Sigma_k = R^{k \cdot k}$
$U_k = R^{p \cdot k}$
$V_k = R^{k \cdot q}$

Question 2 :

La modification des paramètres eps, search_space et percentage peut affecter la précision et la vitesse de convergence des méthodes d'itération du sous-espace, ainsi que le nombre de valeurs singulières et de vecteurs qui sont approximés.

eps : Ce paramètre contrôle la tolérance de convergence des méthodes d'itération du sous-espace. Une valeur plus faible de eps permet d'obtenir une plus grande précision, mais peut entraîner une convergence plus lente. L'augmentation de la valeur de eps peut accélérer la convergence, mais peut conduire à une précision moindre. Comme on le voit, dans le premier tableau, eps = 1^{-8} alors que dans le second eps = 1^{-7}.

Versions	Temps (sec)
0	145.87
1	5.41
2	5.3
3	3.02

FIGURE 5 – eps = 1^{-8}

Versions	Temps (sec)
0	49.4
1	3.47
2	3.66
3	2.88

FIGURE 6 – eps = 1^{-7}

On a bien une diminution du temps pour une valeur plus grande de eps.

pourcentage : Ce paramètre est utilisé dans la méthode d'itération par la puissance et dans les méthodes d'itération par le sous-espace (versions 1 à 3) pour contrôler le nombre de valeurs singulières et de vecteurs qui sont approximés. Il représente le pourcentage de l'énergie totale des valeurs singulières qui sont capturées par l'approximation. Une valeur plus élevée du pourcentage conduit à une plus grande précision, mais peut nécessiter davantage de coûts de calcul et de mémoire. De même, dans le tableau 1, *percentage* = 0.995 et dans le deuxième *percentage* = 0.997.

Versions	Temps (sec)
0	145.87
1	5.41
2	5.3
3	3.02

FIGURE 7 – *percentage* = 0.995

Versions	Temps (sec)
0	118.4
1	7.14
2	6.4
3	3.11

FIGURE 8 – *percentage* = 0.997

Pour une valeur plus élevée de *percentage* on a un temps plus élever pour les versions 1 à 3 (la 0 n'étant pas impacté par ce changement).

search_space : Ce paramètre contrôle la taille de l'espace de recherche pour l'approximation des valeurs et vecteurs singuliers dominants. L'augmentation de la valeur de search_space peut améliorer la précision de l'approximation, mais elle entraîne également une augmentation des coûts de calcul et des besoins en mémoire.

Dans l'ensemble, la modification de ces paramètres nécessite un compromis entre le coût de calcul, les besoins en mémoire, la précision et la vitesse de l'approximation. Il est important de choisir les paramètres appropriés en fonction de l'application spécifique et des ressources informatiques disponibles. L'ensemble des versions semble bien suivre ces règles et la plus efficace et complète semble être la V3 (en ne prenant pas en compte eig).

4 Conclusion

Tout au long de ce projet, nous avons amélioré un algorithme de base en mettant l'accent sur son optimisation en termes de coût de calcul et de mémoire. Nous avons mis en pratique ces améliorations dans la partie 2, où nous avons décompressé une image en considérant l'importance des différents paramètres de nos implémentations.

5 Annexe

```
% version améliorée de la méthode de l'espace invariant (v1)
% avec utilisation de la projection de Raleigh-Ritz

% Données
% A          : matrice dont on cherche des couples propres
% m          : taille maximale de l'espace invariant que l'on va utiliser
% percentage : pourcentage recherché de la trace
% eps        : seuil pour déterminer si un vecteur de l'espace invariant a convergé
% maxit      : nombre maximum d'itérations de la méthode

% Résultats
% V : matrice des vecteurs propres
% D : matrice diagonale contenant les valeurs propres (ordre décroissant)
% n_ev : nombre de couples propres calculées
% it : nombre d'itérations de la méthode
% itv  : nombre d'itérations pour chaque couple propre
% flag : indicateur sur la terminaison de l'algorithme
    %  flag = 0  : on converge en ayant atteint le pourcentage de la trace recherché
    %  flag = 1  : on converge en ayant atteint la taille maximale de l'espace
    %  flag = -3 : on n'a pas convergé en maxit itérations

function [ V, D, n_ev, it, itv, flag ] = subspace_iter_v1( A, m, percentage, eps, maxit )

    % calcul de la norme de A (pour le critère de convergence d'un vecteur (gamma))
    normA = norm(A, 'fro');

    % trace de A
    traceA = trace(A);

    % valeur correspondnat au pourcentage de la trace à atteindre
    vtrace = percentage*traceA;

    n = size(A,1);
    W = zeros(m,1);
    itv = zeros(m,1);

    % numéro de l'itération courante
    k = 0;
```

```
     % somme courante des valeurs propres
     eigsum = 0.0;
     % nombre de vecteurs ayant convergés
     nb_c = 0;

     % indicateur de la convergence
     conv = 0;

     % ---------- ALGORITHME 4 - Debut ----------
     % 1er de l'algorithme 4 : Generate an initial set of m orthonormal vectors
     % on génère un ensemble initial de m vecteurs orthogonaux
     Vr = randn(n, m);
     Vr = mgs(Vr);

     % rappel : conv = (eigsum >= trace) | (nb_c == m)

     % 2eme pas de l'algorithme 4 : repeat
     %  et
     % 8eme pas de l'algorithme 4 : until ( P ercentReached > P ercentT race or nev = m or k > M axIter )
     while (~conv && k < maxit)

         % 3eme pas de l'algorithme 4 : k = k + 1
         k = k+1;

         % 4eme pas de l'algorithme 4 : Compute Y such that Y = A*V
         Y = A*Vr;

         % 5eme pas de l'algorithme 4 : V <- orthonormalisation of the columns of Y
         Vr = mgs(Y);

         % 6eme pas de l'algorithme 4 : Rayleigh-Ritz projection applied on matrix A and orthonormal vectors
         [Wr, Vr] = rayleigh_ritz_projection(A, Vr);

         %% Quels vecteurs ont convergé à cette itération
         analyse_cvg_finie = 0;
         % nombre de vecteurs ayant convergé à cette itération
         nbc_k = 0;
         % nb_c est le dernier vecteur à avoir convergé à l'itération précédente
```

```
        % nombre de vecteurs ayant convergé à cette itération
        nbc_k = 0;
        % nb_c est le dernier vecteur à avoir convergé à l'itération précédente
        i = nb_c + 1;

        % 7eme pas de l'algorithme 4 : Convergence analysis step: save eigenpairs that have converged
        while(~analyse_cvg_finie)
            % tous les vecteurs de notre sous-espace ont convergé
            % on a fini (sans avoir obtenu le pourcentage)
            if(i > m)
                analyse_cvg_finie = 1;
            else
                % est-ce que le vecteur i a convergé

                % calcul de la norme du résidu
                aux = A*Vr(:,i) - Wr(i)*Vr(:,i);
                res = sqrt(aux'*aux);

                if(res >= eps*normA)
                    % le vecteur i n'a pas convergé,
                    % on sait que les vecteurs suivants n'auront pas convergé non plus
                    % => itération finie
                    analyse_cvg_finie = 1;
                else
                    % le_vecteur i a convergé
                    % un de plus
                    nbc_k = nbc_k + 1;
                    % on le stocke ainsi que sa valeur propre
                    W(i) = Wr(i);

                    itv(i) = k;

                    % on met à jour la somme des valeurs propres
                    eigsum = eigsum + W(i);

                    % si cette valeur propre permet d'atteindre le pourcentage
                    % on a fini
                    if(eigsum >= vtrace)
```

```
                    % si cette valeur propre permet d'atteindre le pourcentage
                    % on a fini
                    if(eigsum >= vtrace)
                        analyse_cvg_finie = 1;
                    else
                        % on passe au vecteur suivant
                        i = i + 1;
                    end
                end
            end
        end

        % 7eme pas de l'algorithme 4 suite : update P ercentReached

        % on met à jour le nombre de vecteurs ayant convergés
        nb_c = nb_c + nbc_k;

        % on a convergé dans l'un de ces deux cas
        conv = (nb_c == m) | (eigsum >= vtrace);

    end
    % ---------- ALGORITHME 4 - Fin ----------

    if(conv)
        % mise à jour des résultats
        n_ev = nb_c;
        V = Vr(:, 1:n_ev);
        W = W(1:n_ev);
        D = diag(W);
        it = k;
    else
        % on n'a pas convergé
        D = zeros(1,1);
        V = zeros(1,1);
        n_ev = 0;
        it = k;
    end
```

```
120
121             % 7eme pas de l'algorithme 4 suite : update P ercentReached
122
123             % on met à jour le nombre de vecteurs ayant convergés
124             nb_c = nb_c + nbc_k;
125
126             % on a convergé dans l'un de ces deux cas
127             conv = (nb_c == m) | (eigsum >= vtrace);
128
129         end
130         % ---------- ALGORITHME 4 - Fin ----------
131
132         if(conv)
133             % mise à jour des résultats
134             n_ev = nb_c;
135             V = Vr(:, 1:n_ev);
136             W = W(1:n_ev);
137             D = diag(W);
138             it = k;
139         else
140             % on n'a pas convergé
141             D = zeros(1,1);
142             V = zeros(1,1);
143             n_ev = 0;
144             it = k;
145         end
146
147         % on indique comment on a fini
148         if(eigsum >= vtrace)
149             flag = 0;
150         else if (n_ev == m)
151                 flag = 1;
152             else
153                 flag = -3;
154             end
155         end
156     end
157
```