Projet CS-AD partie 2 : Subspace Iteration Methods

Auteurs: Rogard Diego Souchon Théo Sijelmassi Iliass

Département Science du Numérique : Première année 2020-2021

Sommaire :

I Introduction

Cette partie du projet a pour objectif de comparer l'efficacité de différentes méthodes de calculs de valeurs propres. La méthode sur les puissances itérés n'étant pas la meilleure en terme d'efficacité nous allons nous intéresser à la subspace iteration method et ses variantes.

Réponses aux questions de réflexion :

II Limites de la puissance itérée

Question 1 :

Using the file test_v11.m, compare the running time of the function power_v11 to compute a few eigenpairs with the running time of the function eig of Matlab that compute all the eigenpairs for different sizes and types of matrices. Comment.

Exemples d'exécution de test_v11 :

Matrice 200 x 200 - type 1
******* calcul avec eig ******
Temps eig = 1.000e-02

******* calcul avec la méthode de la puissance itérée ******
puissance itérée : pourcentage 4.000e-01 non atteint avec 20 valeurs propres

Matrice 200 x 200 - type 2
******* calcul avec eig ******
Temps eig = 1.000e-02

******* calcul avec la méthode de la puissance itérée ******
Temps puissance itérée = 1.100e-01

Matrice 200 x 200 - type 3
******* calcul avec eig ******
Temps eig = 1.000e-02

******* calcul avec la méthode de la puissance itérée ******
Temps puissance itérée = 2.300e-01

Matrice 200 x 200 - type 4
******* calcul avec eig ******
Temps eig = 1.000e-02

******* calcul avec la méthode de la puissance itérée ******
puissance itérée : pourcentage 4.000e-01 non atteint avec 20 valeurs propres

On remarque que l'algorithme de la puissance itérée est toujours plus long que la fonction eig de matlab. De plus, la puissance itérée ne calcule que les premiers couples propres et elle peut échouer. Il est donc évident qu'elle est moins efficace et moins rapide que la fonction eig.
La différence de temps d'exécution entre les deux méthodes est en partie dûe au fait que l'algorithme de la puissance itérée utilise deux produits vecteur-matrice ce qui augmente le temps de calcul.

Question 2 :

Algorithme de la puissance itérée amélioré :

```
% mÃ(c)thode de la puissance itÃ(c)rÃ(c)e avec dÃ(c)flation

% DonnÃ(c)es
% A          : matrice dont on cherche des couples propres
% m          : nombre maximum de valeurs propres que l'on veut calculer
% percentage : pourcentage recherchÃ(c) de la trace
% eps        : seuil pour dÃ(c)terminer si un couple propre a convergÃ(c) (mÃ(c)thode de la puissance
itÃ(c)rÃ(c)e)
% maxit      : nombre maximum d'itÃ(c)rations pour calculer une valeur propre (mÃ(c)thode de la puissance
itÃ(c)rÃ(c)e)

% RÃ(c)sultats
% W : vecteur contenant les valeurs propres (ordre dÃ(c)croissant)
% V : matrice des vecteurs propres correspondant
% n_ev : nombre de couples propres calculÃ(c)s
% itv : nombre d'itÃ(c)rations pour chaque couple propre
% flag : indicateur sur la terminaison de l'algorithme
%  flag = 0  : on a convergÃ(c) (on a calculÃ(c) le pourcentage voulu de la trace)
%  flag = 1  : on a atteint le nombre maximum de valeurs propres sans avoir atteint le pourcentage
%  flag = -3 : on n'a pas convergÃ(c) en maxit itÃ(c)rations pour calculer une valeur propre
function [ W, V, n_ev, itv, flag ] = power_v12( A, m, percentage, eps, maxit )

    n = size(A,1);
```

```
% initialisation des rÃ(c)sultats
W = [];
V = [];
it = [];
n_ev = 0;

% trace de A
tA = trace(A);

% somme des valeurs propres
eig_sum = 0.0;

% indicateur de la convergence (pourcentage atteint)
convg = 0;

% numÃ(c)ro du couple propre courant
k = 0;

while (~convg && k < m)

    k = k + 1;

    % mÃ(c)thode de la puissance itÃ(c)rÃ(c)e
    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)
      v = z / norm(z,2);
      z = A*v;
      beta = v'*z;
      norme = norm(beta*v - z, 2)/norm(beta,2);
      nb_it = nb_it + 1;
    end

    % la calcul de ce couple propre a Ã(c)chouÃ(c) => Ã(c)chec global
    if(nb_it == maxit)
      flag = -3;
      % on sort de la fonction en plein milieu
      % ce n'est pas trÃ¨s bien structurÃ(c)
      % pardon aux enseignants de PIM
      return;
    end

    % on sauvegarde le couple propre
    W(k) = beta;
    V(:,k) = v;
    itv(k) = nb_it;
    eig_sum = eig_sum + beta;

    % dÃ(c)flation
    A = A - beta* (v*v');

    % est-ce qu'on a atteint le pourcentage
    convg = eig_sum/tA > percentage;

end

% on a atteint le pourcentage
if (convg)
  n_ev = k;
  flag = 0;
```

```
      W = W';
    else
      % ce n'est pas le cas
      flag = 1;
    end

end
```

Nous avons supprimé dans la deuxième boucle while le y=A*v car nous l'avons déjà calculé dans le précédent parcours de la boucle. Nous n'utilisons plus que z qui réduit de 1 le nombre de calculs matriciels dans la boucle.

Question 3 :

What do you think to be the main drawback of the deflated power method in terms of computing time?

L'opération de déflation consiste à diminuer le rang de la matrice de 1 en conservant les mêmes couples propres.
Si on pose $(\lambda_1 ; W_1)$ le couple propre obtenu par la puissance itérée, on modifie A la matrice de départ par le calcul :

$B = A - \lambda_1 * W_1 * W_1^T$.

Cela implique qu'à chaque nouveau calcul d'un couple propre, il faut effectuer ce produit matriciel et donc cela augmente significativement le temps de calcul à chaque nouveau couple.

III Extension de la méthode de déflation pour calculer les couples propres dominants

Question 4 :

Towards which matrix does V converge the power method algorithm if we apply it to a set of m vectors?

A chaque itération de la boucle while, V est composé de m vecteurs et est modifié. Par ailleurs, ici les m vecteurs ne s'orthonormalisent pas à chaque itération. Alors de cette façon la matrice V va converger vers le vecteur propre associé à la valeur propre la plus élevée en module.

Question 5 :

We are looking at variants of the power method in order to avoid computing the whole spectral decomposition of the matrix A. But in Algorithm 2, a computation of the whole spectral decomposition of the matrix H is performed. Explain why it is not a problem by investigating the dimensions of H.

On sait que $H = V^T * A * V$.
Or $V \in \mathbb{R}^{n \times m}$ et $A \in \mathbb{R}^{n \times n}$
On a donc $H \in \mathbb{R}^{n \times n}$. Or, par définition, m est bien plus petit que n car on ne cherche que les m plus grands couples propres de la matrice A.
Ainsi, calculer la décomposition spectrale de H n'est pas un problème dans le sens où cela sera relativement bien moins long que calculer celle de A.

Question 7 :

Algorithm 4 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 PercenTrace the target percentage of the trace of A

Output: n ev dominant eigenvectors V out and the corresponding eigenvalues Λ_{out} .

l38 : k = 0;

l42 : PercentReached = 0

l48-49 : Generate an initial set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$;

repeat

l52 k = k +1

l56 Compute Y such that Y = A · V

l58 V ← orthonormalization of the columns of Y

l61 Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V

l63-109 Convergence analysis step: save eigenpairs that have converged and update PercentReached

until (PercentReached > PercentTrace or $n_{ev} = m$ or k > MaxIter)

IV Subspace_iter_v2 et Subspace_iter_v3: Vers un solveur efficace

Question 8 :

What is the cost in term of flops of the computation of A^p , then $A^p \cdot V$? How organize differently this computation to reduce this cost?

Calculer A^p revient à faire p-1 produits matriciels de dimensions nxn. On aura donc une complexité d'environ$(p-1) * n^3$ donc de l'ordre du n^3. Pour $A^p * V$ on aura réalisé $(p-1) * n^3 + m * n^2$ opérations. On a ici aussi une complexité en n^3. Ainsi, nous allons d'abord réaliser le calcul $A * V$ puis multiplier p-1 fois par A à gauche pour réaliser le moins d'opérations possibles : on fera $p * m * n^2$ opérations soit une complexité en O(n^2) au lieu de O(n^3).

Question 9:

Modify the file subspace_iter_v2.m to implement this acceleration (note that the initial code of this subprogram is the v1 version of the method, the only difference is the input p).

Voir le fichier subspace_iter_v2.m

Question 10:

Observe the behaviour of this approach when increasing p. Explain your results

Plus p augmente, plus le temps de calcul de subspace_iter_v2.m diminue. Cela est dû au fait qu'on effectue des calculs nécessitant moins de complexités : on calcule les couples propres seulement les p itération. Attention cependant, il ne faut pas mettre un p trop grand sinon le calcul de A^p prend trop de temps et ne contrebalance pas le temps gagné.
Exemples :
p = 5 : Temps subspace iteration v2 = 5.000e-02
p = 10 : Temps subspace iteration v2 = 4.000e-02
p = 15 : Temps subspace iteration v2 = 4.000e-02
p = 20 : Temps subspace iteration v2 = 3.000e-02
p = 25 : Temps subspace iteration v2 = 4.000e-02
p = 30 : Temps subspace iteration v2 = 6.000e-02
p = 35 : Temps subspace iteration v2 = 1.700e-01

Question 11:

For each method, after the computation of the eigenpairs, a file is saved with different values, especially the accuracy of each pair. For the subspace iter v1, it is the iter v1.mat file and the accuracies are in the vector qv1. Explain why, with subspace iter v1 method, this accuracy differs for some of the vectors.

En utilisant subspace iter v1, on remarque que le nombre d'itérations nécessaires pour converger est plus important pour les premières eigenvalues tandis que les dernière converge plus lentement. Il y a aussi un impact sur la précision qui est de moins en moins bonne pour les dernières eigenvalues.

Il est normal que la précision soit moins bonne car il y a plus d'itération donc plus d'approximation dans les valeurs ce qui implique des erreurs numériques.

Question 12:

Try to anticipate what will occur, in term of convergence of the eigenvectors, with the subspace iter v3 method

Nous avons une précision plus élevée car les colonnes ne sont plus perturbées par les erreurs de calcul. Ces erreurs sont induites par le fait de recalculer et ré-orthonormaliser les premières colonnes.

Question 13 :

Copy the file subspace_iter_v2.m into a file subspace_iter_v3.m to implement this deflation.

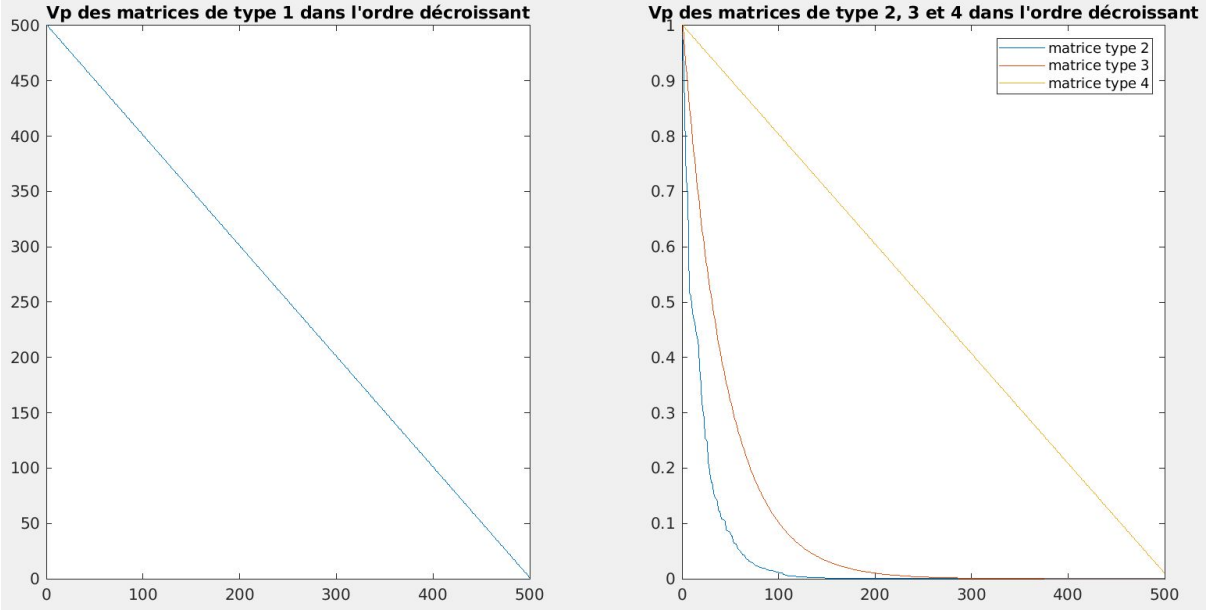
Voir le fichier subspace_iter_v3.m

V Expérimentation Numérique

Question 14 :

What are the differences between the 4 types of matrices ? Create some figures that show the eigenvalue distribution of these different types (the matrix and its spectrum are store in a file when you create the matrix).

Voici le tracé des valeurs propres des différentes matrices.



On remarque que leur différence réside dans la distribution de leurs valeurs propres.

Les matrices de type 1 auront des valeurs propres de module largement supérieur aux autres. C'est pourquoi nous avons séparé ce type des autres lors du tracé. On rappelle que D(i) = i

Pour les matrices de type 2 nous avons D(i) = random(1/cond, 1) avec leur logarithmes uniformément répartie, cond = 1e10. donc les valeurs des D(i) seront réparties entre 0 et 1 ce qui explique la forme de la courbe.

Pour les matrices de type 3 nous avons $D(i) = cond^{-(i-1)/(n-1)}$ avec cond = 1e5 ce qui démontre la similitude avec la fonction de type 1/x.

Pour les matrices de type 4 nous avons D(i) = 1 - ((i-1)/(n-1))*(1 - 1/cond) avec cond = 1e2. Ceci est bien une fonction affine décroissante.

Question 15 :

Compare the performances of the algorithms you have implemented and those provided (including eig) for different types and sizes of matrices.

Voici les temps d'exécution (en secondes) des différents algorithmes pour des tailles de matrices variables.

Matrice de type 1 :

Temps d'exécution:

n algorithme	50	100	150	200	250	300	350	400	450	500
iter_v0	9e-02	CNA	0.17e1	0.83e1	1.55e1	1.56e1	2.37e1	2.63e1	3.85e1	3.74e1
iter_v1	4e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
iter_v2	1e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
iter_v3	1e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
eig	2e-02	2e-02	1e-02	2e-02	2e-02	2e-02	2e-02	4e-02	5e-02	6e-02
power_v11	5e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
power_v12	4e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA

CNA : convergence non atteinte.

Ordre de grandeur de la qualité des résultats :

algorithme	Qualité des valeurs propres (par rapport au spectre de la matrice)	Qualité des couple propres
iter_v0	[e-16 ,e-13]	[e-9 , e-07]
iter_v1	[e-16 ,e-15]	[e-14 , e-08]
iter_v2	[e0, e-15]	[e-14 ,e-08]
iter_v3	[e-16, e-15]	[e-12 ,e-08]
eig	[e0 ,e-15]	[e-16 ,e-14]
power_v11	[e-16 ,e-15]	[e-9 ,e-8]
power_v12	[e-16 ,e-15]	[e-9 ,e-8]

Matrice de type 2 :

Temps d'exécution:

n algorithme	50	100	150	200	250	300	350	400	450	500
iter_v0	1e-02	5e-02	4.6e-1	4.0e-1	3.6	3.71	6.3e-1	1.63e1	3.96	1.99
iter_v1	1e-02	4e-02	5e-02	6e-02	6e-02	8e-02	9e-02	1.1e-1	1.3e-1	14e-1
iter_v2	CNA	CNA	CNA	CNA	CNA	6e-02	3e-02	3e-02	4e-02	4e-02
iter_v3	CNA	CNA	CNA	CNA	CNA	6e-02	3e-02	3e-02	4e-02	4e-02
eig	0e-02	2e-02	1e-02	1e-02	1e-02	2e-02	2e-02	4e-02	4e-02	5e-02
power_v11	1e-02	1e-02	7e-02	1.2e-1	1.7e-1	3.3e-1	1.57	1.62	2.54	2.09
power_v12	1e-02	1e-02	5e-02	1.1e-1	1.4e-1	2.1e-1	8.9e-1	1.10	1.30	1.36

CNA : convergence non atteinte.

Ordre de grandeur de la qualité des résultats :

algorithme	Qualité des valeurs propres (par rapport au spectre de la matrice)	Qualité des couple propres
iter_v0	[e0 ,e-12]	[e-16 , e-06]
iter_v1	[e0 ,e-15]	[e-12 , e-08]
iter_v2	[e-0, e-15]	[e-16 ,e-9]
iter_v3	[e0, e-15]	[e-16 ,e-9]
eig	[e0 ,e-8]	[e-16 ,e-6]
power_v11	[e-16 ,e-15]	[e-9 ,e-8]
power_v12	[e-16 ,e-15]	[e-9 ,e-8]

Matrice de type 3 :

Temps d'exécution:

n algorithme	50	100	150	200	250	300	350	400	450	500
iter_v0	2e-02	2.4e-1	5.3e-1	1.04	1.55	1.78	2.57	3.46	4.5	15.24
iter_v1	1e-02	5e-02	6e-02	8e-02	1.2e-1	1.5e-1	2.6e-1	4.4e-1	1.46	CNA
iter_v2	CNA	CNA	3e-02	3e-02	3e-02	3e-02	6e-02	8e-02	3.4e-1	CNA
iter_v3	CNA	CNA	2e-02	2e-02	3e-02	3e-02	5e-02	7e-02	1.9e-1	CNA
eig	0e-02	2e-02	1e-02	1e-02	1e-02	2e-02	3e-02	4e-02	5e-02	5e-02
power_v11	0e-02	1e-02	1.7e-1	2.9e-1	5.6e-1	7.1e-1	1.21	1.50	2.14	CNA
power_v12	0e-02	1e-02	1.6e-1	1.8e-1	2.8e-1	4.1e-1	7.3e-1	9.1e-1	1.22	CNA

CNA : convergence non atteinte.

Ordre de grandeur de la qualité des résultats :

algorithme	Qualité des valeurs propres (par rapport au spectre de la matrice)	Qualité des couple propres
iter_v0	[e-16 ,e-13]	[e-10 , e-07]
iter_v1	[e-16 ,e-13]	[e-16--e-12 , e-08]
iter_v2	[e-16, e-15-- e-14]	[e-16 ,e-10--e-08]
iter_v3	[e-16, e-15-- e-14]	[e-16 ,e-10--e-08]
eig	[e0 ,e-12]	[e-16 ,e-11]
power_v11	[e0 ,e-15]	[e-9 ,e-8]
power_v12	[e0 ,e-15]	[e-9 ,e-8]

Matrice de type 4 :

Temps d'exécution:

n algorithme	50	100	150	200	250	300	350	400	450	500
iter_v0	9e-02	1.56	1.81	9.22	1.17e1	1.58e1	2.49e1	2.72e1	3.11e1	4.21e1
iter_v1	4e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
iter_v2	1e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
iter_v3	1e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
eig	0e-02	0e-02	1e-02	2e-02	2e-02	2e-02	4e-02	4e-02	5e-02	5e-02
power_v11	3e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA
power_v12	4e-02	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA	CNA

CNA : convergence non atteinte.

Ordre de grandeur de la qualité des résultats :

algorithme	Qualité des valeurs propres (par rapport au spectre de la matrice)	Qualité des couple propres
iter_v0	[e0 ,e-13]	[e-9 , e-07]
iter_v1	[e-16 ,e-15]	[e-14 , e-08]
iter_v2	[e0, e-15]	[e-14 ,e-08]
iter_v3	[e-16, e-15]	[e-12 ,e-08]
eig	[e0 ,e-14]	[e-16 ,e-14]
power_v11	[e0 ,e-15]	[e-9 ,e-8]
power_v12	[e0 ,e-15]	[e-9 ,e-8]

Notons que les valeurs de qualités des résultats et des temps d'exécution n'ont pas été réalisées à l'aide d'une moyenne sur plusieurs réalisations et ne sont donc pas à prendre comme tel. Cependant, elles donnent une bonne idée de ce à quoi on peut s'attendre.
On remarque que les améliorations apportées au subspace_iter_v0 améliorent à la fois le temps d'exécution et la qualité des résultats mais on perd la convergence certaine.
De plus, les algorithmes de la puissance itérée sont bien en général moins efficaces que ceux utilisant le subspace iteration.