TP 2 - Subspace Iteration Methods

A. Rivière - Y. Lacroix - V. Mangé

Département Sciences du Numérique - Première année
2019-2020

1 Limitations of the power method

Question 1 : On peut voir sur la figure 1 que le temps de calcul des valeurs propres avec la méthode de la puissance itérée est **bien plus important (d'un facteur qui va de 10 à 100)** que le temps de calcul sur la même matrice en utilisant la méthode de la routine LAPACK.

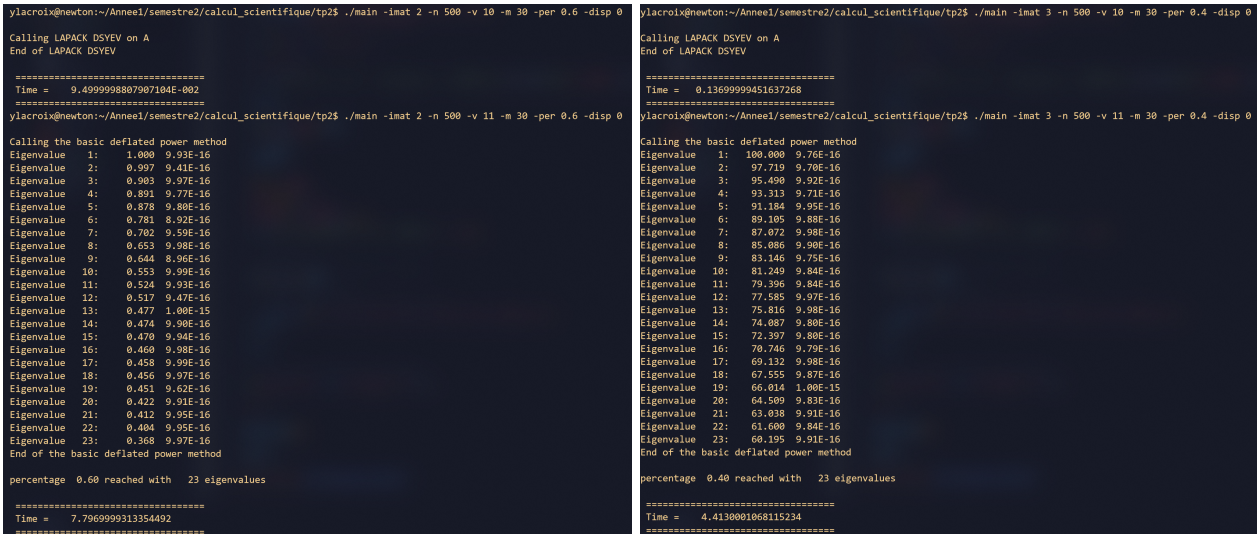

FIGURE 1 – Comparaison entre la méthode de la puissance itérée et la routine LAPACK

Question 2 : Le principal inconvénient de la méthode de la puissance itérée avec déflation est que l'on calcule les valeurs propres les unes après les autres.

2 Extending the power method to compute dominant eigenspace vectors

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

Question 3 : En appliquant la méthode de la puissance itérée à chaque colonne de V, on risque trouver m fois les mêmes éléments propres. En effet, en lançant l'algorithme de `puissance_iteree_ortho.f90`, on s'aperçoit que la première méthode renvoie m fois la même valeur propre et m fois le même vecteur propre correspondant.

Question 4 : Tout d'abord, la dimension de H est $m \times m$, avec m le nombre de vecteurs voulus ($m \leq n$). Ce n'est donc pas un problème de calculer tout le spectre de la matrice H, puisqu'il nous donne seulement le spectre du nombre de valeurs propres voulues, et non pas celui de tout le spectre de A.

Question 5 : Voir le code.

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

Question 6 :

1. lignes 128 à 130 : Répéter, $k = k + 1$
2. ligne 133 : Calculer $Y = AV$
3. ligne 136 : Stocker dans V les colonnes orthonormalisées de Y

4. lignes 145 à 150 : Projection de Rayleigh-Ritz appliquée aux matrices A et V
5. lignes 163 à 200 : Sauvegarde des couples propres qui ont convergé et mise à jour de $PercentReached$
6. ligne 202 : Jusqu'à ($PercentReached > PercentTraceorn_{ev} = mork > MaxIter$)

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

3.1 Block approach (subspace iter v2)

Question 7 : Pour calculer le nombre d'opérations en virgule flottante de $A^p \cdot V$ on calcule :
-le coût du calcul de $A^p \in \mathbb{R}^{n\times n}$. Pour un produit de deux matrices de $\mathbb{R}^{n\times n}$ on a n^2 coefficients à calculer. Pour chaque coefficient, on effectue n produits et $n-1$ sommes donc pour un produit de deux matrices on a $n^2(2n-1)$ opérations. D'où $(p-1)n^2(2n-1)$ opérations pour un produit de p matrices.

-le coût du produit de $A^p \in \mathbb{R}^{n\times n}$ et de $V \in \mathbb{R}^n$. On a n coefficients à calculer. Pour chaque coefficient, on calcule n produits et $n-1$ sommes. On fait donc $n(2n-1)$ opérations au total.

D'où un nombre d'opérations total de : $\boxed{(np-n+1)n(2n-1)}$
Afin de réduire le coût de calcul de A^p on peut utiliser l'algorithme de **l'exponentiation rapide**.

Question 8 : Voir code Fortran.

3.2 Deflation method (subspace iter v3)

Question 9 : Dans le fichier `subspace_iter_v1.f90`, sans la déflation, les premiers vecteurs vont continuer à converger même après avoir atteint une certaine précision. Donc on s'attend à une précision décroissante.

Question 10 : Dans `subspace_iter_v2.f90`, faire $Y = A^pV$ modifie aussi les premiers vecteurs donc on s'attend au même résultat que la question 9.

Question 11 : Nous avons développé le code de la v3 sur Matlab. *Voir code Matlab.*

4 TO DO : Numerical experiments

Question 12 : On a testé l'algorithme `subspace_iter_v2.f90` pour différentes valeurs de p :

Valeur de p	temps
1	2.46
2	1.79
3	1.47
4	1.25
5	1.33
10	1.21
20	1.15

(Avec matrice de type 2, taille 500 et 20 valeurs propres.)

On voit que l'impact de p est important puisqu'on réduit par **deux** le temps de calcul pour $p = 10$ en comparaison avec `subspace_iter_v1.f90` où il n'y a pas de paramètre p (et donc que $p = 1$).

Question 13 : Pour des matrices de taille 100 on voit sur la figure 4 que :

Type 1 : Les valeurs propres sont uniques avec un écart de 1 entre deux valeurs propres consécutives, leurs valeurs vont de 1 à 100.

Type 2 : Les valeurs propres peuvent être très proches voir identiques, leurs valeurs sont assez petites, entre 0 et 1. Les valeurs sont **bornées entre 0 et 1** quelque que soit la taille de la matrice.

Type 3 : Les valeurs propres ont l'air de suivre une loi exponentielle décroissante et prennent des valeurs entre 0 et la taille de la matrice. Les valeurs sont **bornées entre 0 et 100** quelque que soit la taille de la matrice.

Type 4 : Sembable au type 1, mais avec des valeurs propres beaucoup plus grandes. Les valeurs vont de 10 à 1000. Les valeurs sont **bornées entre 10 et 1000** quelque que soit la taille de la matrice.

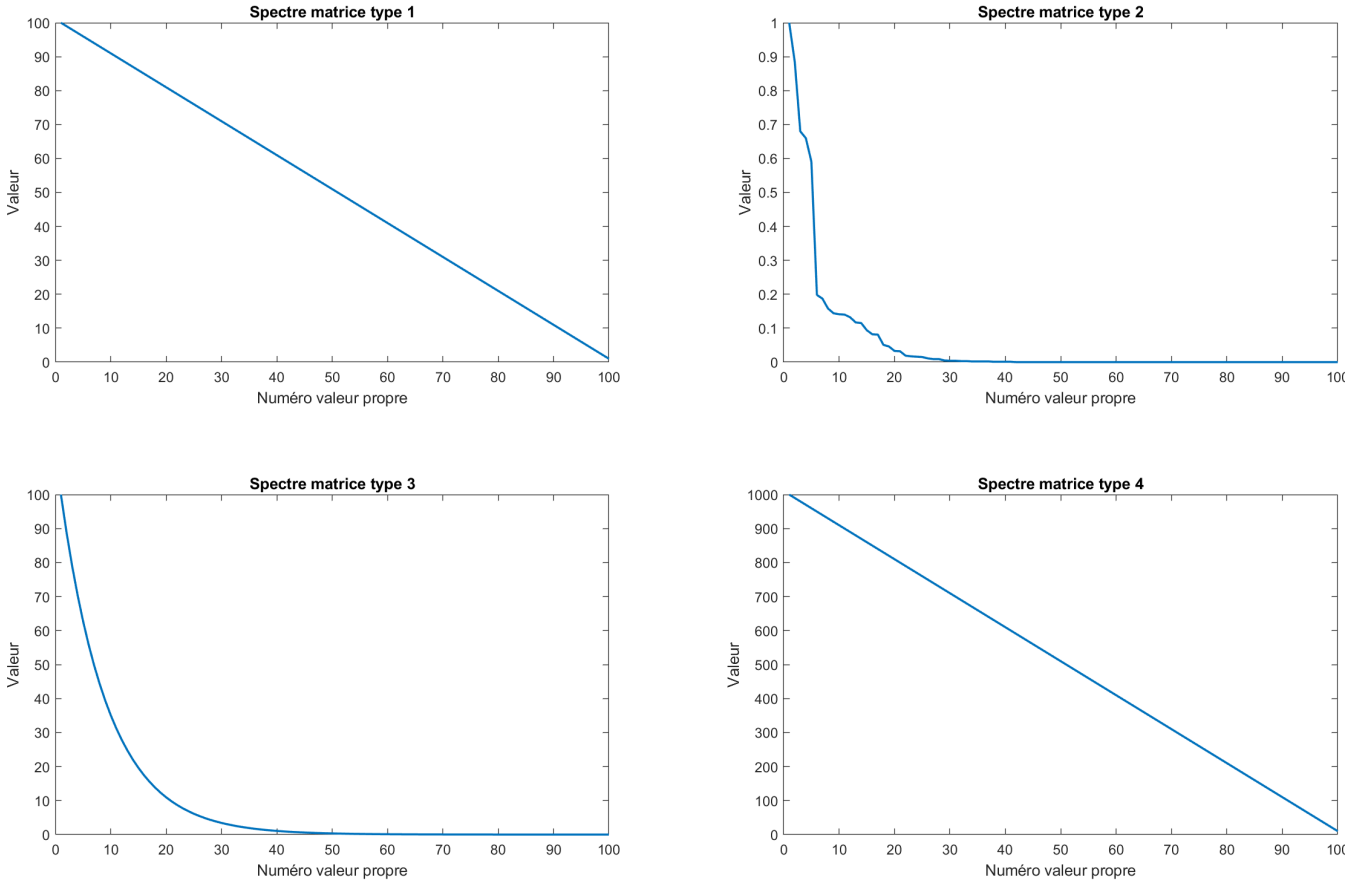

FIGURE 2 – Comparaison des spectres des 4 types de matrices

Question 14 : Avec $p = 20$, on demande 70% de la trace et au maximum 20 valeurs propres/un sous espace de taille 20 et 30.000 itérations maximum avec une précision inchangée de 10^{-15}, on recueille les données suivantes (en secondes) :
(*maxIter* signifie que l'algorithme n'a pas pu arriver à la précision désirée au bout des 30.000 itérations.)

Pour le **type 1** :

Taille	v0	v1	v2	Lapack	Puissance itérée avec déflation
50	3.4	0.2	0.02	0.003	0.4
100	11	1	0.2	0.01	1
250	45	16	3.5	0.07	23
500	131	99	66	0.4	168
1000	*maxIter*	*maxIter*	142	3	*maxIter*

Pour le **type 2** :

Taille	v0	v1	v2	Lapack	Puissance itérée avec déflation
50	*maxIter*	0.002	*maxIter*	0.001	0.02
100	*maxIter*	0.008	*maxIter*	0.04	0.05
250	*maxIter*	0.09	0.05	0.03	1.7
500	*maxIter*	4	2	0.2	19
1000	*maxIter*	70	70	1.3	*maxIter*

Pour le **type 3** :

Taille	v0	v1	v2	Lapack	Puissance itérée avec déflation
50	*maxIter*	0.01	*maxIter*	0.006	0.01
100	*maxIter*	0.01	*maxIter*	0.004	0.08
250	*maxIter*	0.8	0.3	0.04	1.3
500	*maxIter*	6	2.5	0.2	10
1000	*maxIter*	38	17	2	88

Pour le **type 4** :

Taille	v0	v1	v2	Lapack	Puissance itérée avec déflation
50	3.5	0.1	0.02	0.001	0.3
100	7	0.8	0.2	0.005	1.5
250	26	7	3	0.05	13
500	76	56	23	0.3	86
1000	*maxIter*	*maxIter*	142	2.5	*maxIter*

Analyse et conclusion :

1. De manière générale, Lapack est la meilleure méthode pour calculer les valeurs propres peu importe le type et la taille de la matrice.
2. *Cas de maxIter pour de grandes matrices :* On observe que les versions v1 et v2 fonctionnent mieux que la v0 et la méthode de la puissance itérée. En effet, ces dernières ne convergent pas assez vite pour les types 2 et 3, et on n'obtient pas la précision voulue au bout des 30.000 itérations pour une matrice de taille 1000. De même, pour les versions v0, v1 et la méthode de la puissance itérée pour les types 1 et 4.
3. *Cas de maxIter pour de petites matrices :* On observe pour les types 2 et 3, avec les méthodes v0 et v2, la précision des valeurs propres n'atteint jamais 10^{-15} mais cette dernière oscille entre 10^{-10} et 10^{-11}. Notre hypothèse : les valeurs propres étant bornées (entre 0 et 1 pour le type 2, entre 0 et 100 pour le type 3), plus on augmente la taille de la matrice, plus les valeurs propres sont proches deux à deux. Ainsi, il faut moins d'itérations pour calculer une valeur propre d'où la chute en temps pour la v2 pour les types 2 et 3. Si le temps augmente après, cela doit être parce que le nombre d'opérations devient très élevé à chaque itération. Si cette hypothèse n'est pas valide pour le type 4 dont les valeurs propres sont entre 0 et 1000, c'est parce que même pour une matrice de taille 1000, deux valeurs propres consécutives ont au moins un écart de 1 ce qui est élevé.