

## TP4 – Approximation de valeurs propres

### Exercice 1. Méthode de la puissance

Soient  $A \in M_n(\mathbb{C})$  et  $\mathbf{x}_0 \in \mathbb{C}^n$ . La méthode de la puissance consiste à calculer les suites  $\nu \in \mathbb{C}^{\mathbb{N}}$  et  $\mathbf{x} \in (\mathbb{C}^n)^{\mathbb{N}}$  suivantes :

$$\nu_k = \langle \mathbf{x}_k, A\mathbf{x}_k \rangle, \quad \mathbf{x}_{k+1} = \frac{A\mathbf{x}_k}{\|A\mathbf{x}_k\|_2}, \quad k \geq 0.$$

**Théorème.** Soit  $A \in M_n(\mathbb{C})$  une matrice admettant une valeur propre dominante :  $\text{spec}(A) = \{\lambda_1, \lambda_2, \dots, \lambda_p\}$  avec  $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_p|$ . On note  $D_1$  l'intersection de  $\text{Ker}(A - \lambda_1 I_n)$  avec la sphère unité de  $\mathbb{C}^n$  et  $d$  la distance en norme hermitienne sur  $\mathbb{C}^n$ . Alors pour tout  $\mathbf{x}_0 \in \mathbb{C}^n$  admettant une composante non-nulle dans le sous-espace caractéristique associé à  $\lambda_1$ , la méthode converge au sens où

$$\lim_{k \rightarrow \infty} \nu_k = \lambda_1, \quad \lim_{k \rightarrow \infty} d(\mathbf{x}_k, D_1) = 0.$$

De plus on a les comportements asymptotiques suivants.

- ★ Si  $\lambda_1$  et toutes les valeurs propres de même module que  $\lambda_2$  sont non-défectives, alors

$$d(\mathbf{x}_k, D_1) = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

- ★ Si  $\lambda_1$  n'est pas défective mais qu'au moins une valeur propre de module égal à  $|\lambda_2|$  est défective, alors en notant  $r \in \llbracket 2, n \rrbracket$  la taille du plus grand bloc de Jordan associé, on a

$$d(\mathbf{x}_k, D_1) = O\left(k^{r-1} \left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

- ★ Si  $\lambda_1$  est défective, alors

$$d(\mathbf{x}_k, D_1) = O\left(\frac{1}{k}\right).$$

- Sans écrire de fonction, programmer cet algorithme et illustrer numériquement le théorème avec les (contre-)exemples suivants (partagez-vous le travail !):

$$A_0 = \begin{pmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_1 = \begin{pmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} -2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_4 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

On initialisera la méthode avec un vecteur  $\mathbf{x}_0 \in \mathbb{R}^3$  aléatoire suivant la loi uniforme sur  $[-1, 1]^3$ .

### Exercice 2. Projections orthogonales

Soient  $n \in \mathbb{N}$ . Étant donnée une famille libre  $\mathbf{y}_1, \dots, \mathbf{y}_m$  de  $\mathbb{C}^n$ , on souhaite déterminer une famille orthonormée  $\mathbf{z}_1, \dots, \mathbf{z}_m$  telle que

$$\forall j \in \llbracket 1, m \rrbracket, \text{Vect}(\mathbf{z}_1, \dots, \mathbf{z}_j) = \text{Vect}(\mathbf{y}_1, \dots, \mathbf{y}_j). \quad (\star)$$

1. Définir une fonction `orthonorm(Y)`, `return Z`, qui procède à l'orthonormalisation de Gram-Schmidt pour répondre à cette question. La matrice  $Y$  aura pour colonnes les vecteurs  $\mathbf{y}_j$  et la matrice  $Z$  aura pour colonnes les vecteurs  $\mathbf{z}_j$ . On pourra utiliser le fait que la matrice de la projection orthogonale sur  $\text{Vect}(\mathbf{z}_1, \dots, \mathbf{z}_j)$  s'obtient simplement par  $Z \cdot Z^T$ . On vérifiera ensuite la propriété  $(\star)$  en utilisant la commande `numpy.linalg.matrix_rank` et le caractère orthonormé de la famille  $(\mathbf{z}_1, \dots, \mathbf{z}_m)$  par le calcul de  $Z^T \cdot Z$ .
2. (a) On se place dans  $\mathbb{C}^4$ . Tester les commandes :

```
--> import numpy as np
--> A=np.random.rand(4,4), A1=A.copy()
--> x=A1[:,0], x.shape=(4,1), q1,r1=np.linalg.qr(x), x[0]=x[0]+r1
--> u=x/np.linalg.norm(x), u.shape=(4,1), H1=np.eye(4,4)-2*np.dot(u,u.T)
--> B=np.dot(H1,A), B1=B.copy(), x=B1[1:,1], x.shape=(3,1)
```

--> q2,r2=np.linalg.qr(x), u=x/np.linalg.norm(x), u.shape=(3,1),H2=np.eye(4,4)

--> H2[1:,1:]=np.eye(3,3)-2\*np.dot(u,u.T),C=np.dot(H2,B)

(b) Compléter cet algorithme par une troisième itération, afin d'obtenir la factorisation QR de la matrice A. Vérifier alors les différentes propriétés attendues.

3. À partir de l'exemple précédent, définir une fonction  $[Q,R]=QR(A)$  qui prend en argument une matrice A de  $\mathcal{M}_{n,m}(\mathbb{C})$  et renvoie une matrice Q unitaire d'ordre n et une matrice R de  $\mathcal{M}_{n,m}(\mathbb{C})$  triangulaire supérieure à diagonale réelle positive ou nulle, telles que  $A=Q*R$ .

4. Tester si les deux algorithmes coïncident, en examinant ce que renvoie

--> A = np.random.rand(20,15); Z = orthonorm(A); Q,R = QR(A)

--> np.linalg.norm(Z-Q[0:,0:15])

Comparer avec Q,R = np.linalg.qr(A).

### Exercice 3. Adaptation de la méthode de la puissance pour le calcul de plusieurs valeurs propres

Soient  $A \in H_n(\mathbb{C})$  et  $m \in \llbracket 1, n \rrbracket$ . L'entier  $n$  peut être relativement grand et  $m$  raisonnablement petit devant  $n$ . On souhaite déterminer une approximation des  $m$  « plus dominantes » valeurs propres (réelles) de A et des vecteurs propres associés, qui peuvent être choisis formant une famille libre orthonormée de  $\mathbb{C}^n$ . Pour les expériences numériques, on choisira en premier lieu la matrice fixée suivante :

--> n=40; J=np.random.rand(n,n); U=orthonorm(J)

--> A=np.dot(np.dot(U,np.diag(100\*0.9\*\*(np.arange(1,n+1))))),U.T)

où  $\text{expm}(J,m)$  est la fonction (à écrire) qui renvoie les  $m$  premiers termes de la série exponentielle de J.

Considérons une famille libre  $\mathbf{x}_0^{(1)}, \dots, \mathbf{x}_0^{(m)}$  de  $\mathbb{C}^n$  et définissons la suite  $(N_k)_{k \geq 0}$  à valeurs dans  $H_m(\mathbb{C})$  ainsi que les suites  $\mathbf{x}_k^{(1)}, \dots, \mathbf{x}_k^{(m)}$  indexées par  $k \in \mathbb{N}$ , à valeurs dans  $\mathbb{C}^n$ , de la manière suivante :

$$\forall k \geq 0, \quad N_k = \left( \langle \mathbf{x}_k^{(i)}, A \mathbf{x}_k^{(j)} \rangle \right)_{1 \leq i, j \leq m}$$

$$\forall k \geq 0, \quad \forall j \in \llbracket 1, m \rrbracket, \quad \mathbf{y}_{k+1}^{(j)} = A \mathbf{x}_k^{(j)}$$

et les vecteurs  $\mathbf{x}_{k+1}^{(1)}, \dots, \mathbf{x}_{k+1}^{(m)}$  sont alors obtenus par orthonormalisation de la famille  $\mathbf{y}_{k+1}^{(1)}, \dots, \mathbf{y}_{k+1}^{(m)}$ , comme on l'a fait dans l'exercice 2, dans le cas particulier  $m = 1$ .

1. En adaptant l'algorithme de la méthode de la puissance, programmer l'algorithme en question.

*Indication* : on pourra structurer les vecteurs  $(\mathbf{x}_k^{(j)})_{1 \leq j \leq m}$  dans une matrice rectangulaire X et transcrire toutes les opérations d'algèbre linéaire utilisées en produits matriciels bien choisis.

2. Estimer la vitesse de convergence de la suite  $(N_k)$  pour l'exemple proposé.

3. Reprendre l'exemple de 1. avec le choix suivant de la matrice orthogonale U :

--> n=40; J=np.tril(np.ones((n,n))); J=(J-J.T)/2; U=expm(J,1e-7)

où  $\text{expm}(J, \text{eps})$  est la fonction (à écrire) qui renvoie la série exponentielle de J calculée à eps près.

4. Tester d'autres matrices hermitiennes ou non-hermitiennes. Trouvez-vous la méthode robuste ?

5. Quelles seraient selon vous les hypothèses typiques garantissant la convergence de la méthode ?

### Exercice 4. Algorithme QR de calcul de valeurs propres

Pour approcher toutes les valeurs propres de  $A \in GL_n(\mathbb{C})$ , on peut utiliser la méthode QR, fortement apparentée à l'algorithme précédent, qui consiste à construire la suite de matrice  $(A_k)$  à valeurs dans  $GL_n(\mathbb{C})$  par la récurrence :

$$\begin{cases} A_0 = A \\ \forall k \geq 0 & A_k = Q_k R_k \quad (\text{décomposition QR de } A_k) \\ A_{k+1} = R_k Q_k. \end{cases}$$

On a vu dans le cours que  $A_k$  devient asymptotiquement triangulaire supérieure (mais est, en général, non convergente au sens usuel), sa diagonale convergeant vers les valeurs propres de  $A$  (éventuellement par blocs si  $A$  n'est pas hermitienne) ordonnées par module décroissant. Pour obtenir les vecteurs propres associés, on pourrait utiliser une méthode de la puissance inverse avec translation ou utiliser à bon escient les matrices  $Q_k$  successives.

1. À l'aide de la décomposition  $QR$ , programmer le schéma.
2. Tester sur différentes matrices (non triangulaires!), hermitiennes ou non-hermitiennes.
3. Examiner la convergence.