TP5 – Approximation de valeurs propres

Exercice 1. Méthode de la puissance

Soient $A \in \mathrm{M}_n(\mathbb{C})$ 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 $A \in \mathrm{M}_n(\mathbb{C})$ une matrice admettant une valeur propre dominante : $\mathrm{spec}(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_p\}$ avec $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_p|$. On note D_1 l'intersection de $\mathrm{Ker}(A - \lambda_1 I_n)$ 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 λ_1, la méthode converge au sens où*

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

De plus on a les comportements asymptotiques suivants.

- ⋆ *Si λ_1 et toutes les valeurs propres de même module que λ_2 sont non-défectives, alors*
$$\mathrm{d}(\mathbf{x}_k, D_1) = O\left(\left|\tfrac{\lambda_2}{\lambda_1}\right|^k\right).$$
- ⋆ *Si λ_1 n'est pas défective mais qu'au moins une valeur propre de module égal à $|\lambda_2|$ est défective, alors en notant $r \in [\![2, n]\!]$ la taille du plus grand bloc de Jordan associé, on a*
$$\mathrm{d}(\mathbf{x}_k, D_1) = O\left(k^{r-1}\left|\tfrac{\lambda_2}{\lambda_1}\right|^k\right).$$
- ⋆ *Si λ_1 est défective, alors*
$$\mathrm{d}(\mathbf{x}_k, D_1) = O\left(\tfrac{1}{k}\right).$$

- 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 $\mathbf{y}_1, \ldots, \mathbf{y}_m$ de $\mathbb{C}^n$, on souhaite déterminer une famille orthonormée $\mathbf{z}_1, \ldots, \mathbf{z}_m$ telle que

$$\forall j \in [\![1, m]\!], \ \mathrm{Vect}(\mathbf{z}_1, \ldots, \mathbf{z}_j) = \mathrm{Vect}(\mathbf{y}_1, \ldots, \mathbf{y}_j). \tag{Flag}$$

- 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 $\mathrm{Vect}(\mathbf{z}_1, \ldots, \mathbf{z}_j)$ s'obtient simplement par `Z*Z'`.

  · On vérifiera ensuite la propriété (Flag) en utilisant la commande `rank` et la caractère orthonormé de la famille $(\mathbf{z}_1, \ldots, \mathbf{z}_m)$ par le calcul de `Z'*Z`.
- On se place dans $\mathbb{C}^4$. Ecrire une fonction `householder(a)` et tester les commandes :

```
--> import numpy as np
--> A=np.random.rand(4,4)
--> u=householder(A[0:,0]), u.shape=(4,1), H1=np.eye(4,4)-2*np.dot(u,u.T)
--> B=np.dot(H1,A)
--> u=householder(B[1:,1]), 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)
```

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

- 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`.
- Vérifier que les deux algorithmes coïcident, 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])
```

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

Soient $A \in \mathsf{H}_n(\mathbb{C})$ et $m \in [\![1, n]\!]$. 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.tril(np.ones((n,n))); J=(J-J.T)/2; U=expm(J,100)
--> A=np.dot(np.dot(U,np.diag(100*0.9**(np.arange(1,n+1)))),U.T)
```

où `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 $\mathsf{H}_m(\mathbb{C})$ 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 :

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

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

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 $m = 1$.

- 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.
- Estimer la vitesse de convergence de la suite (N_k) pour l'exemple proposé.
- Tester d'autres matrices hermitiennes ou non-hermitiennes. Trouvez-vous la méthode robuste ?
- 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 $A \in \mathsf{GL}_n(\mathbb{C})$, 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 $\mathsf{GL}_n(\mathbb{C})$ par la récurrence :

$$\left|\begin{array}{l} A_0 = A \\ \forall k \geq 0 \quad A_k = Q_k R_k \quad \text{(décomposition } QR \text{ de } A_n\text{)} \\ A_{k+1} = R_k Q_k. \end{array}\right.$$

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.

- À l'aide de l'exercice 1, programmer la méthode QR.
- Tester sur différentes matrices (non triangulaires !), hermitiennes ou non-hermitiennes.
- Examiner la convergence.