TD3

Calcul de valeurs propres

On cherche dans ce TD à entrer en contacte avec le module `NumPy` (qui proportionne des éléments spécifiques destinés au calcul scientifique) et le module `SciPy` (qui complète et étend ces compétences). D'un point de vue méthodologique, nous nous focalisons sur des aspects liés à l'algèbre linéaire, notamment au calcul des valeurs propres.

1 Éléments de base de l'analyse matricielle

Nous commençons par importer les modules `NumPy` et `SciPy`

```
import numpy as np
from scipy import linalg as LA
```

L'objet fondamental dans `NumPy` est un `ndarray`, un "array" d'objets du même type (par contraste au `List`), avec une taille fixée à sa création. Pour la création d'un ndarray on utilise la fonction `array`. On va illustrer ceci avec la construction de matrices.

1.1 Construction de matrices

Nous pouvons construire une matrice comme un `ndarray`, suivant cet exemple

```
A = np.array([[1,2], [3,4]])
print(A.shape) ##nous donne la taille de la matrice
print(A)
```

Nous pouvons définir une deuxième matrice B et un vecteur b, tous les deux comme des `arrays`, et effectuer le produit matricielle avec la fonction `dot` de `NumPy` :

```
B =  np.array([[5,6], [7,8]])
b = np.array([[1],[2]])
print(b.shape)

AB = np.dot(A,B)
print(AB)
Ab = np.dot(A,b)
print(Ab)
```

Cette approche garde la versatilité des arrays `ndarrays`, mais est moins proche à l'usage mathématique. Une alternative dans NumPy à la construction de matrices, plus adaptée à la notation mathématique, utilise la notion `matrix`. Dans ce cadre, les opérations précédentes s'écrivent

```
A_M = np.matrix([[1,2], [3,4]])
print(A_M.shape)
B_M =  np.matrix([[5,6], [7,8]])
print(B_M)
b_M = np.matrix([[1],[2]])
print(b_M.shape)

AB_M = A_M*B_M
print(AB_M)
Ab_M = A_M*b_M
print(Ab_M)
```

1.2 Operations sur les matrices

On a vu ci-dessus comment faire des produits de matrices. Des autres opérations sur les matrices, autant vues comme des `arrays` que comme des `matrix` sont fournies par `NumPy` et `Scipy` : Par exemple

```
AI = LA.inv(A)
AI_M = LA.inv(A_M)

AT = np.transpose(A)
AT_M = np.transpose(A_M)
```

Exercice.

i) Explorer les fonctions dans python pour calculer le déterminant et la trace de une matrice, et l'appliquer aux cas précédentes.

ii) Résoudre un problème type $A \cdot x = b$, en utilisant la fonction `solve` de `linalg` et vérifier le résultat en utilisant l'inverse et le produit des matrices.

1.3 Visualisation d'une matrice

Dans certaines applications, il peut être utile d'avoir une affichage graphique qualitative de la matrice, surtout s'il s'agit d'une matrice de grand rang. Une exemple d'une telle visualisation est le suivant

```
fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
ax.set_aspect('equal')
plt.imshow(A, interpolation='nearest', cmap=plt.cm.ocean)
plt.colorbar()
plt.show()
```

Exercice. Introduire une fonction `visualisation_matrice(A)` qui affiche la structure qualitative de la matrice. L'appliquer à une matrice aléatoire définie comme

```
A = np.random.rand(ndim,ndim)
```

avec `ndim` la dimension de la matrice.

2 Calcul des valeurs propres

2.1 Diagonalisation

Étant donnée une matrice carrée $A \in M_n(\mathbb{K})$, où $\mathbb{K}$ est un corps donné (ici, notamment $\mathbb{R}$ ot $\mathbb{C}$)), on appelle valeur propre de A tout $\lambda \in \mathbb{K}$ tel qu'il existe $x \in \mathbb{K}^n$, $x \neq 0$, qui satisfait $A\lambda = \lambda x$. L'élément x est appelé vecteur propre de A associé à λ. On dit que A est diagonalisable dans $\mathbb{K}$ s'il existe une base $(v_1, \ldots, v_n)$ avec $v_i \in \mathbb{K}^n$ et des valeurs $\lambda_1, \ldots, \lambda_n \in \mathbb{K}$ telles que $A\lambda_i = \lambda_i v_i$.

Nous pouvons calculer les valeurs et vecteurs propres d'une matrice en utilisant la fonction `eig`. Exemple, pour la matrice

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \tag{1}$$

on peut écrire

```
A =  np.array([[1, 2], [3, 4]])
eigenvalue, eigenvector= LA.eig(A)
```

Exercice. Un utilisant un critère sur la nature de vecteurs propres comme base, construire une fonction en python qui détermine si une matrice est diagonalisable.

Exercice. Étant donnée $A \in M_n(\mathbb{C})$ définie par

$$\begin{pmatrix} 0 & 1 & & & & & \\ & 0 & 1 & & & & \\ & & 0 & 1 & & & \\ & & & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \\ & & & & & 0 & 1 \\ & & & & & & 0 \end{pmatrix},$$

montrer que A n'est pas diagonalisable. Appliquer à A la fonction construite en python pour déterminer la diagonisabilité des matrices.

Exercice. Construire, pour chaque n, la matrice Δ_n

$$\Delta_2 = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \quad , \quad \Delta_3 = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \quad , \quad \Delta_4 = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix},$$

c'est à dire :

$$(\Delta_n)_{ij} = \begin{cases} 2 & , \ i = j \\ -1 & , \ |i-j| = 1 \\ 0 & , \ |i-j| > 1 \end{cases}$$

i) Définir une fonction en python qui construit la matrice Δ_n.
ii) Pour une matrice Δ_n, avec un choix de n arbitraire, déterminer avec la fonction construite en python si elle est diagonalisable.
iii) Afficher la structure qualitative de la matrice Δ_n avec la fonction `visualisation_matrice(A)`.
iv) Calculer et afficher les valeurs propres de Δ_n.
v) Montrer graphiquement a distribution de valeurs propres.
vi) Explorer l'effet sur la distribution de valeurs propre de l'addition des matrices ϵB, avec ϵ petit.

3 Calcul du valeur propre avec la valeur absolue la plus grande

Si on se intéresse que à la valeur propre de plus grande valeur absolue λ_1, on peut utiliser la *méthode la puissance* définie par l'algorithme suivant :

1. On choisit un vecteur initial $z_o \in \mathbb{R}^n$ (arbitraire) et on définit $x^{(0)} = \frac{z_o}{||z_o||_2}$, $\lambda^{(0)} = (x^{(0)})^t \cdot A \cdot x^{(0)}$ et $\epsilon^{(0)} = 2\epsilon$ (avec ϵ choisi positif).
2. On définit $z^{(k+1)} = Ax^{(k)}$.
3. On normalise $x^{(k+1)} = \frac{z^{(k+1)}}{||z^{k+1}||_2}$ et on calcule $\lambda^{(k+1)} = (x^{(k+1)})^t \cdot A \cdot x^{(k+1)}$
4. On calcule $\epsilon^{(k+1)} = |\lambda^{(k+1)} - \lambda^{(k)}|$.
5. Si $\epsilon^{(k+1)} < \epsilon$ on arrête l'itération. Si $\epsilon^{(k+1)} \geq \epsilon$ on itère en k.

Pour des choix de x_o assez génériques [notamment telles que $z_o \notin \mathrm{Ker}(A - \lambda_a \mathrm{I}_n)$], la suite $\lambda^{(k)}$ définie par l'algorithme converge vers la valeur λ_1.

Exercice.
i) Construire une fonction en python qui implémente la méthode de la puissance.
ii) Comparer le résultat obtenue avec celui de la fonction `eig` pour la matrice A in Eq. (1) et avec une matrice aléatoire de taille 25×25.

4 Calcul de valeurs propres avec la méthode QR

Un matrice Q est dite orthogonale si elle satisfait $Q^tQ = QQ^t = \mathrm{I}_n$ ou, d'une manière équivalente, $Q^{-1} = Q^t$. Toute matrice $A \in M_n(\mathrm{R})$ admet une décomposition

$$A = QR \,, \tag{2}$$

où Q est orthogonale et R est une matrice triangulaire supérieure à coefficients diagonaux positifs. Si A est inversible, la décomposition est unique. En python, on peut calculer la décomposition QR d'une matrice A en utilisant le module `linalg` de `SciPy`, notamment

```
Q, R =  LA.qr(A)
```

Exercice. Déterminer avec python la décomposition QR de la matrice Δ_7. Montrer (numériquement) que les vecteurs colonne de la matrice Q sont orthogonaux. Raisonner si les vecteurs lignes doivent être orthogonaux.

4.1 Algorithme QR pour la détermination des valeurs propres

L'algorithme QR de détermination des valeurs propres est une méthode itérative qui construit une suite de matrices $A^{(k)}$ dont la diagonale converge vers les valeurs propres de A :

1. On pose $k = 0$ et $A^{(0)} = A$ et on fixe un seuil de convergence ϵ.
2. Si $\max_{i>j} |A^{(k)}_{ij}| \geq \epsilon$:
   i) On calcule $Q^{(k)}$ et $R^{(k)}$ telles que
   $$A^{(k)} = Q^{(k)} R^{(k)} \tag{3}$$
   ii) On définit
   $$A^{(k+1)} = R^{(k)} Q^{(k)} \tag{4}$$
3. On itère en k.

L'algorithme s'arrête après un nombre des pas k_o. Pour définir la reste des éléments de la suite $(A^{(k)})_{k\in\mathbb{N}}$ on considère la suite stationnaire déterminée par $A^{(k)} = A^{(k_o)}, k \geq k_o$. Sous des conditions assez génériques (mais aussi assez techniques), les éléments diagonaux de A convergent vers les valeurs propres : $\lim_{\epsilon\to 0} \lim_{k\to\infty} (A^{(k)})_{ii} = \lambda_i$, avec λ_i valeurs propres de A. Si A est symétrique, $A^{(k)}$ converge vers une matrice diagonale.

Exercice.
i) Construire une fonction `diagonale_superieure_max(A)` qui calcule $\max_{i>j} |A^{(k)}_{ij}|$.
ii) Implémenter la méthode QR à la matrice A in Eq. (1) et comparer avec les résultats obtenus avec la fonction `eig`.
iii) Implémenter l'algorithme QR pour une matrice aléatoire avec 25 lignes et colonnes, en affichant la matrice à chaque pas de l'itération avec la fonction `visualisation_matrice(A)` construite ci-dessus. Exporter le résultat en pdf en utilisant une instruction type

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.set_aspect('equal')
plt.imshow(A, interpolation='nearest', cmap=plt.cm.ocean)
plt.colorbar()
fig.savefig("matrix_QR.pdf")
```

iv) Répéter pour une matrice aléatoire symétrique et vérifier que l'algorithme QR converge vers une matrice diagonale.