

TD3

Calcul de valeurs propres

On cherche dans ce TD à entrer en contact avec le module NumPy (qui fournit 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 "tableau" (ou *array* en anglais) 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 matriciel 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)
```

NumPy propose aussi un cas particulier de `ndarrays` pour lesquelles il n'est pas nécessaire de faire appel à la fonction `dot`, de sorte que l'on des notations plus proche des notations mathématiques usuelles : on crée des objets de type `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 Opérations 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 1.

- i) Explorer les fonctions dans python pour calculer le déterminant et la trace d'une matrice, et l'appliquer aux cas précédents.
- 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 un affichage graphique qualitatif de la matrice, surtout s'il s'agit d'une matrice de grand rang. Un exemple d'une telle visualisation est le suivant

```
from matplotlib import pyplot as plt
```


$x^{(k)}$ dont les premiers termes sont calculés 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 $z_o \notin \text{Ker}(A - \lambda_a I_n)^\perp$], la suite $\lambda^{(k)}$ définie par l'algorithme converge vers la valeur λ_1 .

Exercice 6.

- 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 de l'équation (1) et avec une matrice aléatoire de taille 25×25 .

4 Calcul de valeurs propres avec la méthode QR

Une matrice Q est dite orthogonale si elle satisfait $Q^t Q = Q Q^t = I_n$ ou, d'une manière équivalente, $Q^{-1} = Q^t$. Toute matrice $A \in M_n(\mathbb{R})$ admet une décomposition

$$A = QR, \quad (2)$$

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

$$Q, R = \text{LA.qr}(A)$$

Exercice 7. Déterminer avec python la décomposition QR de la matrice Δ_7 . Montrer (numériquement) que les vecteurs colonne de la matrice Q sont orthogonaux. Justifier (de manière théorique et non pas numérique) si les vecteurs lignes sont, ou pas, orthogonaux.

4.1 Algorithme QR pour la détermination des valeurs propres

L'algorithme QR de détermination des valeurs propres est une méthode itérative s'appuyant sur une suite de matrices $A^{(k)}$ dont la diagonale converge vers les valeurs propres de A . L'algorithme ci-dessous calcule les premiers termes de cette suite :

1. On pose $k = 0$ et $A^{(0)} = A$ et on fixe un seuil de convergence ϵ .
2. Si $\max_{i>j} |A_{ij}^{(k)}| \geq \epsilon$:
 - i) On calcule $Q^{(k)}$ et $R^{(k)}$ telles que

$$A^{(k)} = Q^{(k)} R^{(k)} \quad (3)$$

- ii) On définit

$$A^{(k+1)} = R^{(k)} Q^{(k)} \quad (4)$$

3. On itère en k .

Sous des conditions assez génériques (mais aussi assez techniques), les éléments diagonaux de A convergent vers les valeurs propres : $\lim_{k \rightarrow \infty} (A^{(k)})_{kk} = \lambda_k$, avec λ_k valeurs propres de A . Si A est symétrique, $A^{(k)}$ converge vers une matrice diagonale.

Exercice 8.

- i) Construire une fonction `diagonale_superieure_max(A)` qui calcule $\max_{i>j} |A_{ij}^{(k)}|$.
- ii) Implémenter la méthode QR pour la matrice A de l'équation (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` construite précédemment. Au lieu d'afficher directement les matrices, vous pourrez enregistrer l'image en pdf en utilisant une instruction du type


```
plt.close()
plt.imshow(A, interpolation='nearest', cmap=plt.cm.gray_r)
plt.colorbar()
plt.savefig("matrix_QR.pdf")
```
- iv) Répéter pour une matrice aléatoire symétrique et vérifier que l'algorithme QR converge vers une matrice diagonale.