
TP 9 - Réduction d'une matrice et d'un endomorphisme

Pour ce TP, nous avons besoin des bibliothèques suivantes :

`NUMPY`

```
import numpy as np
np.array() — Transforme une liste en matrice numpy
np.zeros([n, m]) — Créé la matrice nulle de taille  $n \times m$ 
np.eye(n) — Créé la matrice identité de taille  $n$ 
np.diag(L) — Créé la matrice diagonale dont les termes diagonaux sont les éléments de la liste  $L$ 
np.transpose(M) — Renvoie la transposée de  $M$ 
np.dot(M, P) — Renvoie le produit matriciel  $MP$ 
```

`NUMPY.LINALG`

```
import numpy.linalg as la
la.inv(M) — Renvoie l'inverse de la matrice  $M$  si elle est inversible
la.matrix_rank(M) — Renvoie le rang de  $M$ 
```

1 Réduction d'une matrice

On pose $A = \begin{pmatrix} 3 & 0 & -1 \\ 2 & 4 & 2 \\ -1 & 0 & 3 \end{pmatrix}$. On veut calculer A^n pour tout $n \in \mathbb{N}$ de différentes manières.

1.1 Calculer A^n sans réduction

1. En utilisant la fonction `array` du module `numpy`, définir la matrice A en Python.

On définit la matrice A en Python comme suit :

```
1 A=np.array([[3,0,-1],[2,4,2],[-1,0,3]])
```

2. Écrire une fonction itérative (avec un `for`) nommée `Puissance(n)` qui renvoie A^n .

On utilise la relation $\forall n \in \mathbb{N}, A^{n+1} = A \times A^n$ pour construire cette fonction. On introduit une matrice B qui va stocker les puissances successives de A .

```
1 def Puissance(n):
2     B=np.eye(3)
3     for k in range(1,n+1):
4         B=np.dot(B,A)
5     return B
```

3. Écrire une fonction récursive (avec un `if` et un appel à la fonction au rang précédent) nommée `PuissanceRec(n)` qui renvoie A^n .

On utilise la relation $\forall n \in \mathbb{N}, A^{n+1} = A \times A^n$ pour construire cette fonction. C'est une fonction récursive, donc on fait appel à la fonction au rang précédent pour calculer le rang suivant.

```
1 def PuissanceRec(n):
2     if n==0:
3         return np.eye(3)
4     else:
5         return np.dot(A,PuissanceRecur(n-1))
```

1.2 Déterminer à la main des valeurs propres et des vecteurs propres pour A

4. (a) Montrer avec Python que 4 est une valeur propre de A en utilisant la notion de rang.
D'après le cours, 4 est valeur propre de A si et seulement si la matrice $A - 4I_3$ est de rang strictement inférieur à 3. On utilise la fonction `rank` de Python pour calculer le rang, et la fonction `eye` pour programmer la matrice identité.

```
1 r=la.matrix_rank(A-4*np.eye(3))
```

Ici, $r = 1$ donc 4 est valeur propre de A .

- (b) Qu'en déduisez-vous concernant la dimension de $E_4(A)$?
On applique le théorème du rang. On a :

$$3 = \dim(E_4(A)) + \text{rg}(A - 4I_3) \iff \boxed{\dim(E_4(A)) = 2}.$$

5. On veut déterminer avec Python si un vecteur $X \in \mathcal{M}_{3,1}(\mathbb{R})$ est un vecteur propre de A .

- (a) À quelles conditions sur les vecteurs AX et X a-t-on X vecteur propre de A ?
Par définition, X est un vecteur propre de A si X est non nul et qu'il existe $\lambda \in \mathbb{R}$ tel que $AX = \lambda X$, autrement dit si AX et X sont colinéaires.

- (b) Comment lier cette information au rang de la matrice constituée en ligne des vecteurs AX et X ?

Les vecteurs AX et X sont colinéaires si et seulement si cette matrice est de rang strictement inférieur à 2. On veut de plus que X soit non nul, donc le rang doit être non nul. Ainsi, le rang de la matrice doit valoir 1.

- (c) Écrire une fonction `EstVP(X)` qui prend en entrée un vecteur $X = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \in \mathcal{M}_{3,1}(\mathbb{R})$ défini en Python par `np.array([a,b,c])` et qui renvoie `True` si X est un vecteur propre de A , `False` sinon. Pour créer la matrice qui contient en ligne les vecteurs X et Y , on utilise la syntaxe `np.array([X,Y])`.

D'après la question 5b, on propose le programme suivant :

```
1 def EstVP(X):
2     Y=np.dot(A,X)
3     B=np.array([X,Y])
4     r=la.matrix_rank(B)
5     return r==1
```

- (d) Tester cette fonction sur le vecteur $U = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}$ pour confirmer qu'il s'agit d'un vecteur propre de A .

Le test confirme que U est un vecteur propre de A .

```
>>> U=np.array([1,-2,1])
```

```
>>> EstVP(U)
True
```

6. (a) Écrire une fonction `DeterValP(X)` qui prend en entrée un vecteur $X \in \mathcal{M}_{3,1}(\mathbb{R})$ et qui renvoie la valeur propre associée si X est un vecteur propre de A , `False` sinon.

On commence par déterminer si le vecteur X est un vecteur propre de A . Si oui, on veut renvoyer le coefficient de proportionnalité entre AX et X . On choisit donc le premier coefficient non nul du vecteur X (qui existe car X est non nul) et on renvoie le quotient entre le coefficient de AX correspondant et celui de X .

```

1 def DeterValP(X):
2     if EstVP(X)==True:
3         Y=np.dot(A,X)
4         k=0
5         while X[k]==0:
6             k+=1
7         return Y[k]/X[k]
8     else:
9         return False

```

- (b) Tester cette fonction sur le vecteur U pour déterminer une deuxième valeur propre de A .
Le test nous dit que U est vecteur propre de A associée à la valeur propre 2.

```

>>> DeterValP(U)
2.0

```

7. Connaît-on le spectre de A ? La matrice A est-elle diagonalisable?

Pour l'instant, on sait que $\{2, 4\} \subset \text{Sp}(A)$. On a montré en question 4b que $\dim(E_4(A)) = 2$. Or $\dim(E_2(A)) \geq 1$, et $\dim(E_4(A)) + \dim(E_2(A)) \leq 3$. Par conséquent, on a $\dim(E_2(A)) = 1$, $\text{Sp}(A) = \{2, 4\}$ et comme $\dim(E_4(A)) + \dim(E_2(A)) = 3$, la matrice A est diagonalisable.

8. (a) En utilisant la fonction `EstVP`, montrer que les vecteurs $V = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ et $W = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$ sont des vecteurs propres de A et déterminer à quelle valeur propre ils sont associés.

Le test confirme que V et W sont des vecteurs propres de A , associés à la valeur propre 4.

```

>>> V=np.array([0,1,0])
>>> W=np.array([-1,0,1])

>>> EstVP(V)
True
>>> EstVP(W)
True

>>> DeterValP(V)
4.0
>>> DeterValP(W)
4.0

```

- (b) En déduire une matrice inversible P et une matrice diagonale D telles que $A = PDP^{-1}$.
Les vecteurs V et W ne sont pas colinéaires, donc ils forment une famille libre de cardinal 2 de $E_4(A)$, et $\dim(E_4(A)) = 2$, donc (V, W) est une base de $E_4(A)$. Le vecteur U est non nul et $\dim(E_2(A)) = 1$ donc (U) forme une base de $E_2(A)$. On propose donc les matrices suivantes :

$$P = \begin{pmatrix} 1 & 0 & -1 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \text{ et } D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}.$$

- (c) Que vaut alors A^n pour tout $n \in \mathbb{N}$?

On montre par récurrence que pour tout $n \in \mathbb{N}$, on a $A^n = PD^nP^{-1}$.

- (d) Écrire une fonction `PuissanceDiago(n)` qui utilise la formule obtenue en question 8c pour calculer A^n .

En utilisant les propriétés des matrices diagonales, on obtient le programme suivant :

```

1 def PuissanceDiago(n):
2     P=np.array([[1,0,-1],[-2,1,0],[1,0,1]])
3     D=np.diag([2**n,4**n,4**n])
4     return np.dot(P,np.dot(D,la.inv(P)))

```

1.3 Utiliser une fonction Python pour diagonaliser A

NUMPY.LINALG

`import numpy.linalg as la`

`la.eigvals(M)` — Renvoie la liste des valeurs propres de M

`la.eig(M)` — Renvoie un couple L, P où L est la liste des valeurs propres de M et P la matrice de passage associée

9. Comment utiliser les fonctions décrites dans l'aide Python pour retrouver les matrices D et P déterminées en question 8b ?

On utilise les fonctions Python comme suit :

```
1 L,P=la.eig(A)
2 D=np.diag(L)
```

On obtient le résultat suivant :

```
>>> P
array([[ 0.          ,  0.70710678,  0.40824829],
       [ 1.          ,  0.          , -0.81649658],
       [ 0.          , -0.70710678,  0.40824829]])

>>> D
array([[4., 0., 0.],
       [0., 4., 0.],
       [0., 0., 2.]])
```

On remarque que l'ordre des colonnes de D est changé et que les colonnes de la matrice de passage sont colinéaires (mais non égales) à celles que nous avons trouvées, mais tout est cohérent.

2 Réduction d'un endomorphisme

Soit $n \geq 1$. On pose φ l'application telle que :

$$\forall P \in \mathbb{R}_n[X], \varphi(P) = XP' + P.$$

10. Montrer que φ est un endomorphisme de $\mathbb{R}_n[X]$.

Soit $P \in \mathbb{R}_n[X]$, on a $P' \in \mathbb{R}_{n-1}[X]$, donc $XP' \in \mathbb{R}_n[X]$. Ainsi, $XP' + P \in \mathbb{R}_n[X]$, donc φ est à valeurs dans $\mathbb{R}_n[X]$.

Soient $(P, Q) \in \mathbb{R}_n[X]^2$ et $\lambda \in \mathbb{R}$, on a :

$$\begin{aligned} \varphi(\lambda P + Q) &= X(\lambda P + Q)' + \lambda P + Q \\ &= X(\lambda P' + Q') + \lambda P + Q \text{ par linéarité de la dérivation} \\ &= \lambda(XP' + P) + XQ' + Q \\ &= \lambda\varphi(P) + \varphi(Q) \end{aligned}$$

donc φ est une application linéaire.

Ainsi, φ est un endomorphisme de $\mathbb{R}_n[X]$.

En Python, le polynôme $P = a_0 + a_1X + a_2X^2$ sera représenté par la liste $[a_0, a_1, a_2]$, le polynôme $Q = \sum_{k=0}^n a_k X^k$ par la liste $[a_0, a_1, \dots, a_n]$. Pensez bien à tester toutes les fonctions sur un polynôme simple, par exemple $P = X^2 + 3X + 2$, représenté par la liste $[2, 3, 1]$.

11. Écrire une fonction `Derive(L)` qui prend en entrée la liste `L` correspondant au polynôme $P \in \mathbb{R}_n[X]$ et qui renvoie la liste correspondant au polynôme dérivé P' .

On considère le polynôme $Q = \sum_{k=0}^n a_k X^k$ représenté par la liste $[a_0, a_1, \dots, a_n]$. On a

$$Q' = \sum_{k=0}^n k a_k X^{k-1} = \sum_{k=1}^n k a_k X^{k-1} = \sum_{\ell=0}^{n-1} (\ell+1) a_{\ell+1} X^{\ell}$$

en posant $\ell = k - 1$.

On propose donc le programme suivant :

```
1 def Derive(L):
2     D=[]
3     for k in range(len(L)-1):
4         D.append((k+1)*L[k+1])
5     return D
```

12. Écrire une fonction `Multiplie(L)` qui prend en entrée la liste `L` correspondant au polynôme $P \in \mathbb{R}_n[X]$ et qui renvoie la liste correspondant au polynôme XP .

On considère le polynôme $Q = \sum_{k=0}^n a_k X^k$ représenté par la liste $[a_0, a_1, \dots, a_n]$. On a

$$XQ = \sum_{k=0}^n a_k X^{k+1} = \sum_{\ell=1}^{n+1} a_{\ell-1} X^{\ell}$$

en posant $\ell = k + 1$.

On propose donc le programme suivant :

```
1 def Multiplie(L):
2     M=[0]
3     for k in range(1, len(L)+1):
4         M.append(L[k-1])
5     return M
```

13. Écrire une fonction `Phi` qui prend en entrée la liste `L` correspondant au polynôme $P \in \mathbb{R}_n[X]$ et qui renvoie la liste correspondant au polynôme $\varphi(P)$.

On utilise les fonctions précédentes. Nos listes représentant XP' et P sont de même longueur. Attention, Python ne sait pas additionner deux listes, il faut le faire à la main grâce à une boucle `for`.

```
1 def Phi(L):
2     D=Derive(L)
3     M=Multiplie(D)
4     P=[]
5     for k in range(len(L)):
6         P.append(M[k]+L[k])
7     return P
```

14. Déterminer les images des polynômes de la base canonique par φ .

Soit $k \in \llbracket 0, n \rrbracket$, on a :

$$\begin{aligned} \varphi(X^k) &= X \times (kX^{k-1}) + X^k \\ &= kX^k + X^k \\ &= (k+1)X^k \end{aligned}$$

Ainsi, on a $\forall k \in \llbracket 0, n \rrbracket, \varphi(X^k) = (k+1)X^k$, donc X^k est un vecteur propre de φ associé à la valeur propre $k+1$.

15. Soit $\ell \in \mathbb{N}$.

(a) Déterminer l'image des polynômes de la base canonique par φ^ℓ .

On fixe $k \in \llbracket 0, n \rrbracket$, on a :

$$\begin{aligned}\varphi^\ell(X^k) &= \varphi^{\ell-1}(\varphi(X^k)) \\ &= \varphi^{\ell-1}((k+1)X^k) \\ &= (k+1)\varphi^{\ell-1}(X^k) \text{ par linéarité de } \varphi^{\ell-1}\end{aligned}$$

Par récurrence, on montre que $\varphi^\ell(X^k) = (k+1)^\ell X^k$.

(b) En déduire l'image d'un polynôme $P \in \mathbb{R}_n[X]$ quelconque par φ^ℓ .

Soit $P \in \mathbb{R}_n[X]$, on a $P = \sum_{k=0}^n a_k X^k$. On en déduit :

$$\begin{aligned}\varphi^\ell(P) &= \sum_{k=0}^n a_k \varphi^\ell(X^k) \text{ par linéarité de } \varphi^\ell \\ &= \sum_{k=0}^n a_k (k+1)^\ell X^k \text{ par la question précédente.}\end{aligned}$$

(c) Écrire une fonction **PuissancePhi**(1,L) qui prend en entrée un entier 1 et la liste L correspondant au polynôme $P \in \mathbb{R}_n[X]$ et qui renvoie la liste correspondant au polynôme $\varphi^\ell(P)$.

On écrit le programme suivant :

```
1 def PuissancePhi(1,L):
2     Q=[]
3     for k in range(len(L)):
4         Q.append((k+1)**1*L[k])
5     return Q
```

3 Un algorithme de détermination de valeurs propres : l'algorithme de la puissance itérée

Il est légitime de se demander comment Python réussit à déterminer les valeurs propres et les vecteurs propres lorsqu'on utilise la fonction `la.eig`.

Il existe plusieurs algorithmes de détermination de valeurs propres et vecteurs propres. Le plus simple est l'algorithme de la puissance itérée. Il permet d'approcher la plus grande valeur propre d'une matrice (en valeur absolue) et un vecteur propre associé. On travaille sur la matrice $A \in \mathcal{M}_3(\mathbb{R})$ définie en partie 1. L'algorithme fonctionne selon le principe suivant :

- On choisit un vecteur au hasard : X_0
- On calcule $Y_1 = AX_0$.
- On calcule $X_1 = \frac{Y_1}{\|Y_1\|}$.
- On continue : Y_2, X_2 , etc
- On s'arrête lorsque X_n et X_{n+1} sont très proches. Ici, on va choisir $\|X_n - X_{n+1}\| \leq 10^{-4}$.

Le vecteur X_n est alors une approximation d'un vecteur propre de A , et on peut trouver la valeur propre associée en reproduisant le raisonnement mis en place dans notre fonction **DeterValP**.

On rappelle que pour un vecteur $Y = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \in \mathcal{M}_{3,1}(\mathbb{R})$, on a $\|Y\| = \sqrt{a^2 + b^2 + c^2}$.

16. Écrire une fonction `norm(X)` qui prend en entrée un vecteur $X \in \mathcal{M}_{3,1}(\mathbb{R})$ et qui renvoie le réel $\|X\|$.

L'utilisation de `sqrt` nous fait importer la bibliothèque `math`. On propose la fonction suivante :

```
1 from math import *
2 def norm(X):
3     return sqrt(X[0]**2+X[1]**2+X[2]**2)
```

17. Comment peut-on piocher au hasard un vecteur X_0 avec Python ?

On va utiliser une fonction aléatoire de la bibliothèque `random`, par exemple la fonction `rd.random()`.

On peut proposer par exemple le code suivant :

```
1 import random as rd
2 X=np.array([rd.random(),rd.random(),rd.random()])
```

18. Écrire un programme `AlgoPuissance(A)` qui prend en entrée une matrice $A \in \mathcal{M}_3(\mathbb{R})$, qui applique l'algorithme de la puissance itérée et qui renvoie une approximation de la plus grande valeur propre d'une matrice (en valeur absolue) et un vecteur propre associé.

On propose la fonction suivante :

```
1 def AlgoPuissance(A):
2     #On choisit X0 au hasard
3     X=np.array([rd.random(),rd.random(),rd.random()])
4     #On calcule Y1, stocké dans Y et X1, stocké dans Z
5     Y=np.dot(A,X)
6     Z=Y/norm(Y)
7     #On calcule l'erreur pour notre critère d'arrêt
8     err=norm(X-Z)
9     #On répète jusqu'à réalisation de la condition d'arrêt
10    while err>10**(-4):
11        X=Z
12        Y=np.dot(A,X)
13        Z=Y/norm(Y)
14        err=norm(X-Z)
15    #Le vecteur X est un vecteur propre et Y=AX, on détermine la valeur propre
16    #associée
17    k=0
18    while X[k]==0:
19        k+=1
20    lam=Y[k]/X[k]
21    return lam, Z
```

19. Tester ce programme sur la matrice A définie en partie 1. Le résultat est-il cohérent avec ceux de la partie 1 ?

On teste sur la matrice A et on obtient :

```
>>> AlgoPuissance(A)
(4.0158363148145995, array([-0.01175391,
 0.99986074,  0.01184661]))

>>> AlgoPuissance(A)
(3.9963248503327327, array([ 0.05841122,
 0.99658858, -0.05830378]))

>>> AlgoPuissance(A)
(3.9990236211955357, array([ 0.15437743,
 0.97588856, -0.15430205]))
```

Le tirage de X_0 est aléatoire, il est donc normal d'obtenir des résultats différents à chaque lancer. Pour autant, la valeur propre obtenue est toujours proche de 4, qui est bien la plus grande valeur propre de A en valeur absolue.

Les vecteurs associés sont bien des combinaisons linéaires de V et W :

$$[0.15437743, 0.97588856, -0.15430205] = 0.97588856 \times V - 0.15437743 \times W$$

par exemple, donc ce sont bien des éléments de $E_4(A) = \text{Vect}(V, W)$. Ces résultats sont tout à fait cohérents avec ceux de la partie 1.