Numpy et la gestion des tableaux

Quand on apprend à travailler sur les listes, on comprend qu'on peut aussi construire des listes de listes et ainsi, on peut toujours représenter des matrices ou des tableaux sous cette forme. Pourtant le langage Python nous permet de manipuler une classe d'objets dédiés à cet effet : ce sont les objets du type array, et pour lesquels on pourra importer la librairie ***Numpy*** *avec son lot de fonctions pratiques.*

1 Présentation rapide des tableaux et premières opérations

Dans le langage Python, il est possible de travailler sur des tableaux grâce à la bibliothèque **numpy**. Celle-ci nous permet par exemple de définir une matrice comme un élément de la classe *array* :

```
In : from numpy import *
A=array([[1,2,3],[4,5,6]])
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

python

On retiendra en particulier qu'**une telle matrice doit être entrée ligne par ligne**. Les coefficients m_{ij} de la matrice, ainsi que les lignes ou colonnes, pourront alors être modifiés à condition de se souvenir que les indices commencent à 0 :

```
In : A[0,0]=2; A
```

```
array([[2, 2, 3],
       [4, 5, 6]])
```

python

```
In : A[:,2]=[6,12]; A
```

```
array([[2, 2, 6],
       [4, 5, 12]])
```

python

Par ailleurs, on peut retrouver toutes les opérations du calcul matriciel :

- l'addition de deux matrices à l'aide du symbole d'addition : `A+B`,
- la multiplication d'une matrice par un scalaire à l'aide du symbole de multiplication : λ`*A`,
- le produit de deux matrices à l'aide de la commande `dot` : `dot(A,B)`.

Encore une fois, on sera vigilant avec le produit puisque l'interpréteur nous renverra un message d'erreur si les dimensions des matrices ne sont pas compatibles :

```
In : dot(A,A)
```

```
ValueError          Traceback (most recent call last)
(...)
ValueError: objects are not aligned
```

python

On pourra éventuellement vérifier la taille des matrices données à l'aide de la commande `shape` :

```
In : a,b=shape(A); a,b;
```

```
2,3
```

python

De plus, si on souhaite calculer les puissances d'une matrice, on ne pourra pas utiliser le symbole habituel et on sera obligé de redéfinir une telle fonction.

Malgré tout, on trouvera quelques commandes fort pratiques :

Commande dans le langage Python	résultat
`transpose(A)`	renvoie la transposée de la matrice A
`zeros((n,p))`	renvoie la matrice nulle de $\mathcal{M}_{np}(\mathbb{K})$
`eye(n)`	renvoie la matrice identité I_n
`A.copy()`	construit une copie de A non solidaire

Remarque Globalement, il ne faudra donc pas hésiter à travailler avec les objets de cette classe, d'autant plus que le type *array* vous permettra aussi de manipuler des images dans le langage Python !

2 Une librairie dans la librairie... pour l'algèbre linéaire

Le langage Python n'est pas un langage formel et il n'est pas simple a priori de faire de l'algèbre linéaire avec celui-ci. Néanmoins, on pourra parfois importer la sous-librairie **linalg**... très pratique si vous allez aux oraux du concours Centrale :

```
In : from numpy.linalg import *
```
python

On récupère alors de nombreuses fonctions qui peuvent vous faire gagner du temps :

Commande dans le langage Python	résultat
`matrix_rank(A)`	renvoie le rang de la matrice A
`trace(A)`	renvoie la trace de la matrice A
`det(A)`	renvoie le déterminant de la matrice A
`inv(A)`	renvoie l'inverse de la matrice A quand celle-ci est inversible

D'ailleurs, si on considère une matrice A donnée, par exemple :

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

alors il existe des fonctions permettant de récupérer tous les éléments propres... même si on fera attention, les résultats fournis sont souvent donnés de façon approchée. Ainsi, on peut récupérer les coefficients du **polynôme caractéristique** :

```
In : A=array([[2,-4],[1,-3]]); poly(A)
                                    array([ 1., 1., -2.])
```
python

Pour en déduire le spectre de A, on peut évidemment recherche les racines associées, mais on peut aussi appeler la commande `eigvals` ou directement `eig` :

```
In : eigvals(A)
                                    array([ 1., -2.])
```
python

```
In : eig(A)
        (array([ 1., -2.]), array([[0.9701425 , 0.70710678],[0.24253563, 0.70710678]]))
```
python

Dans ce dernier cas, on retiendra qu'en fait la commande `eig` renvoie deux tableaux, l'un correspondant aux **valeurs propres** et l'autre donnant **les vecteurs propres colonne par colonne** : ici, on a par exemple $\begin{pmatrix} 0.70710678 \\ 0.70710678 \end{pmatrix}$ désigne un vecteur propre associé à la valeur propre -2.

Exercice 1 (calcul de puissances entières)**.** [ ]

On importe la bibliothèque Numpy et on rappelle que la fonction `dot` nous donne le produit de deux matrices.

1. Dans le langage Python, construire la fonction $puiss(n : int, A : array) \longrightarrow array$ qui calcule de façon itérative A^n, c'est à dire les puissances itérées de A.
   On pourra vérifier son programme avec la matrice $J = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.
2. Définir alors une procédure récursive pour calculer A^n. En déduire la fonction $puissrec(n : int, A : array) \longrightarrow array$ qui calcule de façon récursive A^n.
3. On pose $A = \begin{pmatrix} 3 & -2 & 2 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \end{pmatrix}$.
   (a) Déterminer les valeurs propres de A, puis justifier que A est diagonalisable.
   (b) Construire alors le programme $expo(n : int) \longrightarrow array$ qui calcule la somme :

$$\sum_{k=0}^{n} \frac{A^k}{k!}$$

3 Applications

Exercice 2 (recherche d'un plus grand élément et la place de celui-ci). [ ]
On considère une matrice $A \in \mathcal{M}_{np}(\mathbb{R})$.

Construire la fonction *cherchermatrice*(A : *array*) $\rightarrow$ *tuple* qui renvoie la valeur du plus grand élément de la matrice A, ainsi que les indices correspondant à sa place.

Exercice 3 (calcul matriciel de la suite de Fibonacci). [ ]
On rappelle que la suite de Fibonacci est définie par :

$$\begin{cases} u_0 = u_1 = 1 \\ \forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

On note alors $X_n = \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix}$.

1. Démontrer qu'il existe une matrice $A \in \mathcal{M}_2(\mathbb{R})$ et indépendante de n tel que :
$$\forall n \in \mathbb{N},\ X_{n+1} = AX_n$$
2. En déduire un nouvel algorithme $fibo(n : int) \longrightarrow array$ qui calcule les termes de la suite de Fibonacci, ne faisant intervenir qu'une récursivité simple et renvoie X_n.

Exercice 4 (matrices inversibles telles que M et M^{-1} sont à coefficients dans $\mathbb{Z}$). [ ]

On note $\mathcal{GL}_n(\mathbb{Z})$ l'ensemble des matrices inversibles et pour lesquelles M et M^{-1} sont à coefficients dans $\mathbb{Z}$. On rappelle que M est d'ordre fini s'il existe $\alpha \in \mathbb{N}^*$ tel que :

$$M^\alpha = I_n$$

1. Dans le langage Python, importer le module **random**, puis construire un programme *mataleatoire* qui renvoie une matrice aléatoire de $\mathcal{GL}_2(\mathbb{R})$.

2. En adaptant le script précédent, construire votre programme afin que celui-ci renvoie une matrice aléatoire à coefficients dans $\mathbb{N}$, puis renvoie son inverse si celle-ci est inversible.

3. (a) Si M est inversible et à coefficients entiers, M^{-1} est-elle nécessairement à coefficients entiers ?

   (b) Si M est inversible et à coefficients entiers et que M^{-1} est aussi à coefficients entiers, quelle hypothèse pouvez-vous faire sur son déterminant ?

4. Construire la fonction *ordre* qui, pour tout entier N donné, teste si une matrice donnée est d'ordre $\alpha \in [\![1, N]\!]$, puis renvoie 0 si elle n'est pas d'ordre fini.

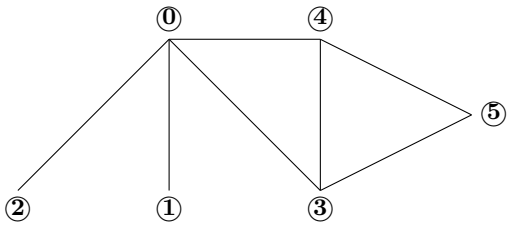
Exercice 5 (représentation d'un graphe). [ ]

Pour représenter un graphe dans le langage Python, on peut le définir par une **liste d'adjacence**, c'est à dire qu'on fait le choix de décrire les relations entre les sommets à l'aide d'une liste (ou d'un dictionnaire), pour lesquelles l'indice du sommet renvoie le nom de ses **voisins** : les sommets qui lui sont adjacents.

Par exemple, si on définit la liste d'adjacence :

```
In :   G=[[1,2,3,4],[0],[0],[0,4,5],[0,3,5],[4,3]]
```

alors, on peut lire les voisins pour chacun des sommets $0, 1, 2, 3, 4, 5$ de sorte que le graphe correspond en fait à :



Dans le langage Python, construire la fonction `listetomatrice(L:list)->array` qui pour tout graphe de liste d'adjacence L renvoie la **matrice d'adjacence** $M = (m_{ij}) \in \mathcal{M}_n(\mathbb{K})$ telle que :

$$m_{ij} = \begin{cases} 1, & \text{s'il existe une arête entre les sommets } i \text{ et } j \\ 0, & \text{sinon} \end{cases}$$, et avec n correspondant au nombre de sommets

Remarque On peut aussi construire l'application réciproque `matricetoliste`, et ainsi pour représenter un tel graphe sans poids, on pourra indifféremment choisir de travailler avec des listes d'adjacence ou des matrices d'adjacence.