Révisions : Vecteurs, Matrices et Courbes en Python

1 Vecteurs

Python utilise principalement les vecteurs ou les matrices pour effectuer des calculs. En fait, un vecteur en Python est un tableau de nombres rangés en ligne, tout comme un vecteur de $\mathbb{R}^n$ (attention, il ne s'agit pas d'une matrice ligne). Après avoir importé la bibiliothèque `numpy`, la commande pour créer un vecteur est `np.array`, dans laquelle on met entre crochets les composantes du vecteur.

Exemple 1.1 *Pour créer le vecteur* $v = (1, 2, 4, -5)$*, on tapera la commande :* `v=np.array([1,2,4,-5])`.

Il existe en Python toute une série de commandes qui permettent de créer des vecteurs. La première commande est `np.linspace(a,b,n)`, qui permet de construire le vecteur de premier élément a et de dernier élément b et dont les n éléments sont espacés régulièrement.

Exemple 1.2 *Pour créer le vecteur* $v = (0.2, 0.4, 0.6, 0.8, 1, 1.2)$*, on pourra utiliser la commande :*

`v=np.linspace([0.2,1.2,6])`.

Pour créer un vecteur à n composantes toutes égales à 0 (resp. à 1), on utilisera la commande `np.zeros(n)` (resp. `np.ones(n)`). Enfin, la commande `np.arange(a,b,p)` crée le vecteur dont la première composante vaut a, dont les composantes sont en progression arithmétique de raison p, qui sont toutes $< b$.

Exemple 1.3 *En Python, la commande* `v=np.zeros(5)` *correspond au vecteur* $v = (0, 0, 0, 0, 0)$*. De même, la commande* `x=np.ones(4)` *correspond au vecteur* $x = (1, 1, 1, 1)$.

A noter que la commande `np.arange(a,b,p)` fonctionne exactement comme la commande `range()` utilisée dans les boucles `for`, sauf que la première crée un vecteur et l'autre rien. On peut donc ne pas spécifier le pas s'il vaut 1, voire ne spécifier que b si $a = 0$.

En Python, on peut aussi procéder à l'extraction d'éléments d'un vecteur. Plus précisément, pour accéder à l'élément d'indice k du vecteur v, il suffit de taper : `v[k]`. Attention, comme Python compte les éléments à partir de 0, il y a toujours un phénomène de décalage. L'avantage ici de la commande `v[k]` est qu'elle permet de récupérer une composante donnée d'un vecteur, voire de la modifier à l'aide de la commande `v[k]=x`.

Exemple 1.4 *Pour accéder à la deuxième composante du vecteur* $v = (1, 2, 4, 3)$*, on tapera :* `v[1]`*. Dans ce cas, Python renverra la valeur* 2 *et non* 1 *! Si l'on souhaite modifier le vecteur* v *pour avoir* $v = (1, 2, 5, 3)$*, on pourra utiliser la commande* `v[2]=5`.

De même, on peut récupérer toute une partie d'un vecteur, à l'aide de la commande `v[a:b]`. Cette dernière permet d'extraire toutes les composantes du vecteur v de la position a (incluse) à la position b (exclue). En outre, la commande `v[a:]` permet de récupérer toutes les composantes de v à partir de la position a.

Exemple 1.5 *Si* $v = (1, 2, 4, 3)$*, la commande* w=`v[1:3]` *retournera le vecteur* $w = (2, 4)$*. De plus, la commande* w=`v[1:]` *retournera le vecteur* $w = (2, 4, 3)$.

2 Matrices

Tout comme pour les vecteurs, on peut définir une matrice en Python à l'aide de la commande `np.array` en délimitant chaque ligne par des crochets.

Exemple 2.1 *Pour écrire la matrice* $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$*, on pourra utiliser la commande :*

`M=np.array([[1,2,3],[4,5,6]])`.

De plus, pour créer une matrice de $\mathcal{M}_{n,m}(\mathbb{R})$ dont toutes les composantes sont égales à 0 (resp. à 1), on utilisera la commande `np.zeros([n,m])` (resp. `np.ones([n,m])`). Enfin, la commande `np.eye(n)` crée la matrice identité de taille n.

En Python, on peut aussi extraire un élément d'une matrice. Plus précisément, pour accéder à l'élément d'indices i, j de la matrice M, il suffit de taper : `M[i,j]`. Attention au décalage, vu que Python compte les éléments à partir de 0. L'avantage ici de la commande `M[i,j]` est qu'elle permet de récupérer

une composante donnée d'une matrice, voire de la modifier à l'aide de la commande `M[i,j]=x`. Plus généralement, la commande `M[i,:]=L` (resp. `M[:,j]=C`) permet de remplacer la ligne d'indice i de M par L (resp. la colonne d'indice j de M par C). A noter que la commande `np.zeros([n,m])` peut être très utile si l'on veut créer une matrice *creuse*, c'est-à-dire *avec beaucoup de zéros*. Dans ce cas, on choisit d'abord de taper la commande `np.zeros([n,m])` pour créer une matrice de taille $n \times m$ remplie de zéros, puis on modifie les valeurs que l'on souhaite.

Exemple 2.2 *Supposons que l'on veuille construire une matrice a de taille 2×3 telle que $a_{1,1} = a_{2,3} = 1$ et telle que tous les autres coefficients soient nuls. Dans ce cas, on utilisera d'abord la commande* `a=np.zeros([2,3])`*, puis les commandes* `a[0,0]=1` *et* `a[1,2]=1`.

On peut aussi extraire la i-ème ligne d'une matrice à l'aide de la commande `M[i-1,:]`. De même, on peut extraire la j-ème colonne d'une matrice à l'aide de la commande `M[:,j-1]`. Les résultats obtenus seront alors des vecteurs, et non des matrices ligne ou colonne. Enfin, comme pour les vecteurs, on peut récupérer toute une partie d'une matrice, à l'aide de la commande `M[a:b,c:d]`. Cette dernière permet d'extraire toutes les composantes de la matrice M de la ligne a (incluse) à la ligne b (exclue), et de la colonne c (incluse) à la colonne d (exclue).

Exemple 2.3 *Pour écrire la matrice* $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$*, on pourra utiliser la commande :*

`M=np.array([[1,2,3],[4,5,6],[7,8,9]])`.

La commande `M[1,2]` *retourne la valeur 6, la commande* `N=M[1:3,1:3]` *la matrice :*

$$N = \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix},$$

ce qui peut être obtenu aussi avec la commande `N=M[1:,1:]`. *Si l'on écrit* `M[0,:]=np.array([0,0,0])`*, alors Python retournera la matrice :*

$$M = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

2.1 Opérations sur les vecteurs et les matrices

Toutes les commandes usuelles sur les vecteurs et les matrices de même taille (`+,-,*,/`) sont disponibles sur Python. Ces commandes vont alors s'appliquer élément par élément. Par exemple, étant donnés deux vecteurs v et w, la commande `v*w` va calculer le vecteur qui a autant de composantes que v et w, dont les composantes sont les $v(i) \times w(i)$.

Exemple 2.4 *Pour calculer le vecteur $v = b - a$ où $a = (1, 2, 3)$ et $b = (3, 4, 5)$, on utilisera la commande* `v=b-a`. *Dans ce cas, Python retournera le vecteur $v = (2, 2, 2)$.*

A noter que la commande `A=A+k` ajoute à chaque coefficient de A le réel k. A noter aussi que les fonctions usuelles (puissance, exponentielle, logarithme népérien, cosinus, sinus, racine carrée, etc) peuvent s'appliquer sans problème à des vecteurs ou à des matrices. Dans ce cas, la fonction f utilisée sera appliquée par Python élément par élément. En d'autres termes, Python va créer le vecteur ou la matrice des valeurs de la fonction f aux points donnés par le vecteur ou la matrice de départ.

Exemple 2.5 *Pour appliquer la fonction cosinus au vecteur $v = (0, \pi, 2\pi)$, on pourra écrire au préalable* `v=np.array([0,np.pi,2*np.pi])`*, puis* `w=np.cos(v)`.

Comme les commandes $*$ et $**$ s'applique élément par élément, on aura besoin de commandes spécifiques pour le produit matriciel ou la puissance d'une matrice. Pour calculer le produit matriciel MN, on utilisera la commande `np.dot(M,N)`. Si k est un réel, alors la commande `np.dot(M,k)` va calculer kM, ce que l'on pourra aussi effectuer avec la commande `k*M`. De plus, pour transposer une matrice M, on utilisera la commande `np.transpose(M)`. Pour les autres opérations matricielles, on aura besoin d'une sous-bibliothèque de `numpy`, que l'on importera avec la commande :

```
import numpy.linalg as al
```

Pour calculer l'inverse d'une matrice carrée inversible M, on utilisera la commande `al.inv(M)`. De plus, pour calculer le rang d'une matrice M, on utilisera la commande `al.matrix_rank(M)`. En outre, pour calculer la puissance M^n d'une matrice carrée M, on utilisera la commande `al.matrix_power(M,n)`. Enfin, pour résoudre le système linéaire $AX = b$, on utilisera la commande `al.solve(A,b)`. A noter que Python renvoie un message d'erreur si la matrice A n'est pas carrée ou n'est pas inversible, et ce même si le système admet au moins une solution. Terminons par quelques commandes spéciales en Python :

— la commande `np.shape(a)` donne la taille de a.

— la commande `np.min(a)` donne le minimum des coefficients de a.

— la commande `np.max(a)` donne le maximum des coefficients de a.

— la commande `np.sum(a)` donne la somme des coefficients de a.

— la commande `np.cumsum(a)` donne la matrice des sommes cumulées des coefficients de a, c'est-à-dire la matrice de même taille que a, dont les éléments sont les sommes partielles des éléments de a, ces derniers étant ajoutés en commençant par ceux de la première ligne de gauche à droite, puis ceux de la deuxième ligne de de gauche à droite, et ainsi de suite.

Ainsi, pour connaître la taille d'un vecteur, on pourra utiliser la commande `a=np.shape(v)[0]` ou `(a,)=np.shape(v)`. De même, pour connaître la taille d'une matrice (nombre de ligne et nombre de colonnes), on utilisera la commande `a,b=np.shape(M)` ou `(a,b)=np.shape(M)`. En outre, pour les commandes `np.min`, `np.max`, `np.sum`, on peut appliquer la commande en question seulement colonne par colonne (avec `np.min(M,0)`) ou ligne par ligne (avec `np.min(M,1)`). Enfin, il reste à noter que les tests logiques (`==`, `!=`, `<`, etc) peuvent être appliqués à des vecteurs ou des matrices. Dans ce cas, Python se chargera de comparer les coefficients des vecteurs ou des matrices un par un. Plus précisément, il retournera un vecteur ou une matrice dont tous les coefficients seront `True` ou `False` selon que le test est vrai ou faux entre les coefficients donnés.

3 Tracés de courbes

Pour tracer des courbes en Python, on aura besoin d'importer la bibliothèque `matplotlib.pyplot` avec la commande :

```
import matplotlib.pyplot as plt
```

Soient $x = (x_1, ..., x_n)$ et $y = (y_1, ..., y_n)$ deux vecteurs de même taille. Pour tracer une ligne brisée dans le plan qui relie les points de cordonnées $M_1 = (x_1, y_1), M_2 = (x_2, y_2), ..., M_n = (x_n, y_n)$, on utilisera la commande `plt.plot(x,y)`. Si l'on veut se contenter de tracer l'ensemble des points $M_1, ..., M_n$ sans les relier par une ligne brisée, mais seulement en les représentant par des croix, on utilisera la commande `plt.plot(x,y,'x')`. On peut bien sûr modifier l'option 'x' pour représenter les points (+ pour la croix du +, . pour un petit point, o pour un gros point, s pour un carré, $*$ pour une étoile, etc). On peut aussi modifier la couleur en rajoutant l'initiale de cette dernière en anglais (b pour "blue", etc). On peut enfin modifier le tracé ($-$ pour la ligne brisée par défaut, $--$ pour les pointillés, etc). A noter que ces options ne sont pas exigibles des candidats pour les concours, mais vous pouvez les retrouver dans des sujets, et donc il peut être utile de savoir ce qu'elles font.

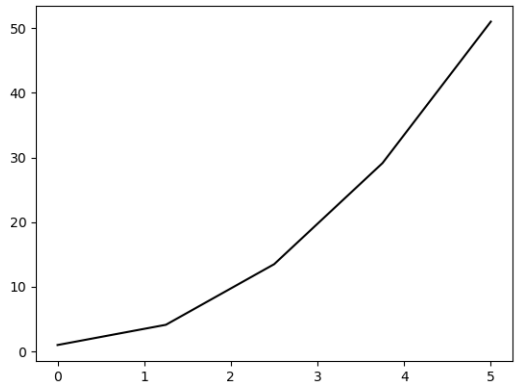
Une fois la commande `plt.plot` utilisée, la commande `plt.show()` permet d'afficher la courbe obtenue, voire les courbes obtenues si l'on a exécuté plusieurs fois la commande `plt.plot` au préalable. Dans ce cas, les courbes obtenues apparaissent (par défaut) dans des couleurs différentes.

Exemple 3.1 *Pour tracer la ligne brisée qui relie l'ensemble des points $M_i = (x_i, y_i)$ avec $x_i = i$ et $y_i = 2i^2 + 1$ pour tout $i \in \{1, ..., 5\}$, on utilisera les commandes suivantes (après avoir importé les bibliothèques adéquates) :*

```
import numpy as np
import matplotlib.pyplot as plt

x=np.array([1,2,3,4,5])
y=2*(x**2)+1
plt.plot(x,y,color='k')
plt.show()
```

A noter que la couleur utilisée par Python pour le tracé de courbes est le bleu par défaut. Ici on a utilisé le noir pour le tracé (spécifiée dans la commande `plt.plot( )` *par l'instruction* `color='k'`*).*



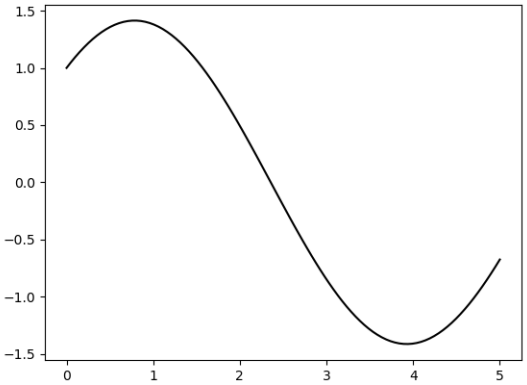
Dès lors, pour tracer la courbe représentative d'une fonction f, l'idée consiste à tracer une ligne brisée reliant des points d'abscisses très proches et dont les ordonnées sont les images des abscisses par f. Si l'espacement entre les abscisses est suffisamment petit, alors la ligne brisée nous donne l'illusion d'une courbe. Le mieux pour obtenir une suite x d'abscisses très proches est de les choisir régulièrement espacées, et donc d'utiliser la commande `np.arange(a,b,p)` (avec p assez petit) ou la commande `np.linspace(a,b,n)` (avec n assez grand) pour la générer. Quant à la suite y des ordonnées, on l'obtient facilement en appliquant la fonction f à x. En général, une précision de l'ordre du centième est largement suivante pour un tracé satisfaisant.

Exemple 3.2 *Pour tracer la courbe représentative de la fonction* $f : x \longmapsto \sin(x) + \cos(x)$ *sur l'intervalle* $[0, 5]$*, on pourra utiliser la liste de commandes suivantes :*

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(0,5,500)
y=np.sin(x)+np.cos(x)
plt.plot(x,y,color='k')
plt.show()
```

A noter qu'ici, on a spécifié le noir comme couleur pour le tracé de la courbe. On obtient alors la courbe suivante en Python :



Exemple 3.3 *Considérons la fonction f définie sur $\mathbb{R}$ par :*

$$f : x \longmapsto \sum_{k=1}^{10} \frac{x^k}{k^2}.$$

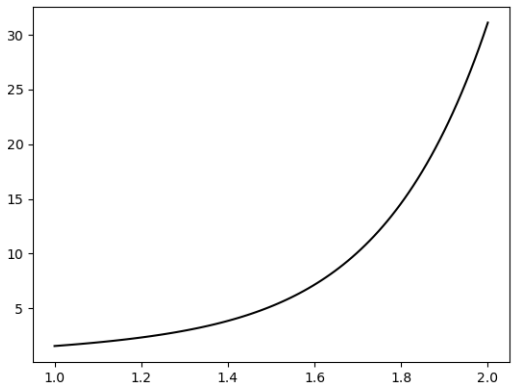
Pour tracer la courbe représentative de cette fonction sur l'intervalle $[1,2]$, on pourra écrire la fonction f au préalable (ce qui est plus approprié vu que l'écriture de f est assez longue), puis l'appliquer au vecteur x, ce qui nous donnera le script suivant :

```
import numpy as np
import matplotlib.pyplot as plt

def fonct(x):
    y=0
    for k in range(1,11):
        y=y+((x**k)/(k**2))
    return y

x=np.linspace(1,2,100)
y=fonct(x)
plt.plot(x,y,color='k')
plt.show()
```

On obtient alors la courbe suivante en Python :



Enfin, terminons par quelques options qui ne sont pas exigibles des candidats mais peuvent apparaître dans des sujets de concours. Bien sûr, toutes ces commandes doivent précéder la commande `plt.show()`.

— Les commandes `plt.xlim` et `plt.ylim` permettent de changer les limites des axes des abscisses et des ordonnées. Par exemple, on écrira la commande `plt.xlim([-4,12])` si l'on veut que l'axe des abscisses aille de $x = -4$ à $x = 12$.

— La commande `plt.axis` permet de modifier les deux axes en une seule fois. Par exemple, on écrira `plt.axis([-4,11,-2,7])` si l'on veut que l'axe des abscisses aille de $x = -4$ à $x = 11$ et que l'axe des ordonnées aille de $y = -2$ à $y = 7$.

— La commande `plt.grid` permet d'afficher une grille, tout simplement en écrivant `plt.grid()`.

— La commande `plt.legend` permet d'afficher une légende s'il y a plusieurs courbes. Par exemple, on écrira `plt.legend(['courbe 1','courbe 2'])` si l'on a deux courbes.