Le calcul scientifique en Python avec Le module NumPy

Présentation des bibliothèques

Python ne sait pas faire grand chose dans le domaine mathématique, **comme tracer une fonction, calculer des valeurs de fonctions usuelles, réaliser des opérations matricielles.**

Cependant, de nombreux modules ont été développés pour pallier ce manque, parmi lesquels :

- **Module *numpy***
- **Module *scipy***
- **Module *matplotlib***

Introduction

Python ne propose de base que le type **list**, conteneur dynamique hétérogène puissant, mais elle est plus difficile dans le contexte mathématique (pas orienté calcul numérique).

Exemple l'opérateur "*" ou "+" ne correspond pas à la multiplication/somme d'un vecteur avec un nombre.

Le module numpy a été créé pour résoudre ce problème. En plus d'un grand nombre de fonctions, elle définit un autre type de tableaux, spécialement conçus pour les opérations mathématiques, tels que **vecteurs**, **matrices**.

Le module numpy

- ❖ La bibliothèque **"numpy"** permet de manipuler facilement les tableaux(**array**), **vecteurs** ou **matrices**.
- ❖ Les tableaux (**array**) **"numpy"** ne gère que les objets de même type.
- ❖ La bibliothèque **"numpy"** propose un grand nombre de fonctions comme :
 - **l'accès rapide aux données d'une matrice ou d'un vecteur.**
 - **La recherche, et l'extraction**
 - **Le tri de données**
 - **calcul statistique de données(moyenne, médiane, somme, somme cumulé,...)**
 - **Calcul scientifique (algèbre linéaire)**

Les sous-module numpy

Le module **numpy** propose un ensemble de classes, d'objets et de fonctions dédiés aux calculs numériques :

❖ **Classe ndarray :**

Créer des tableaux homogènes multi-dimensionnels.

❖ **Classe matrix :**

permet de faire de calcul matriciel.

❖ **numpy.linalg :**

Un module d'algèbre linéaire basique.

❖ **numpy.random :**

Un module pour les générateurs aléatoires.

Importer le module numpy

Afin d'utiliser le module "numpy" il faut tout d'abord l'importer via l'instruction suivante :

❖ Importation de "numpy" :

```
import numpy as np
```

NB: Si le nom du module est un peu long il est préférable de créer un alias via l'instruction **as** dans notre cas c'est **np**.

Manipulation du module numpy

❖ Création d'objets ndarray :

```
>>> import numpy as np
>>> m1=np.ndarray((2,2)) #création d'une matrice de type float par défaut
>>> m1 #affichage
array([[ 5.          ,  8.54987973],
       [14.62008869, 25.          ]])
>>> m2=np.ndarray((2,2),dtype=int) #création d'une matrice de type int
>>> m2 #affichage
array([[ 0, 1072693248],
       [ 0, 1072693248]])
>>> m3=np.ndarray((2,2),dtype=bool) #création d'une matrice de type booleen
>>> m3 #affichage
array([[False, False],
       [False, False]])
```

Manipulation du module numpy

❖ Création d'objets **matrix** :

La classe **np.matrix** permet d'appliquer des méthodes spécifiques aux matrices.

```
>>> import numpy as np
>>> #Créer une matrice par la classe matrix
>>> m=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> m #affichage
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
>>> #Créer une matrice a partir d'un vecteur redimensionné par reshape()
>>> v=np.arange(9) #création d'un vecteur
>>> v
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> m2=np.matrix(v.reshape(3,3)) #matrice de 3x3 a partir de vecteur v
>>> m2 #affichage
matrix([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]])
```

Manipulation du module numpy

❖ La fonction `array()`:

La fonction `array()` convertit un objet **list** en objet **ndarray**.

```
>>> import numpy as np
>>> m1=np.array([1,3,5,6,7])#convertir un objet list en ndarray
>>> m1
array([1, 3, 5, 6, 7])
>>> m2=np.array([1,3,5,6,7],float)#convertir un objet list en ndarray float
>>> m2
array([1., 3., 5., 6., 7.])
>>> m3=np.array([[1,3,5,6,7],[3,8,9,1,0]],float)#list de list: matrice float
>>> m3
array([[1., 3., 5., 6., 7.],
       [3., 8., 9., 1., 0.]])
```

Manipulation du module numpy

❖ La fonction `arange()` et `linspace()` :

La fonction `arange(début,fin,pas)` permet la création d'un vecteur correspondant à progression arithmétique.

La fonction `linspace(début,fin,nbElement)` permet la création d'un vecteur correspondant à progression arithmétique.

```
>>> import numpy as np
>>> v1=np.arange(1,10,2) # arange(début,fin,pas) int
>>> v1
array([1, 3, 5, 7, 9])
>>> v2=np.arange(10,dtype=float)#vecteur des nombres réels
>>> v2
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> v3=np.linspace(1,2,10)#Vecteur v3 en subdivision de [1,2]
>>> v3
array([1.          , 1.11111111, 1.22222222, 1.33333333, 1.44444444,
       1.55555556, 1.66666667, 1.77777778, 1.88888889, 2.          ])
```

Manipulation du module numpy

❖ La fonction `reshape()`:

La fonction `reshape(n,p)` convertit un vecteur en une matrice.

```
>>> import numpy as np
>>> v=np.arange(25) #créer un vecteur de 25 éléments
>>> v
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24])
>>> vMatrice=v.reshape(5,5) #convertir vecteur en une matrice de 5x5
>>> vMatrice
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

Manipulation du module numpy

❖ Les vecteurs particuliers :

Les fonctions **zeros(n)**, **ones(n)** et **rand(n)** permettent la création des vecteurs nulles, vecteurs remplies de 1, et vecteurs remplie à coefficients aléatoires.

```
>>> import numpy as np
>>> v=np.array([]) # création d'une vecteur vide
>>> v
array([], dtype=float64)
>>> v1=np.zeros(5) #vecteur null de taille 5
>>> v1
array([0., 0., 0., 0., 0.])
>>> v1=np.ones(5) #vecteur de taille 5 rempli de 1
>>> v1
array([1., 1., 1., 1., 1.])
>>> #vecteur de taille 5 à valeurs aléatoires uniformes sur [0, 1]
>>> v2=np.random.rand(5)
>>> v2
array([9.31310178e-01, 7.72950941e-01, 8.71122322e-01, 1.78233280e-01,
       5.56609064e-04])
```

Manipulation du module numpy

❖ Les matrices particuliers :

Les fonctions `zeros((n,p))`, `ones((n,p))`, `eye(n)`, `diag(v)` et `rand((n,p))` permettent la création des matrices nulles, remplies de 1, d'identités, dont le diagonale un vecteur et matrice remplie à coefficients aléatoires.

```
>>> import numpy as np
>>> m1=np.zeros((2,3)) #matrice de null de tdimension 2x3
>>> m1
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> m2=np.ones((2,3)) #matrice de 1 de tdimension 2x3
>>> m2
array([[1., 1., 1.],
       [1., 1., 1.]])
>>> m3=np.eye(2) #matrice d'identité de dimension 2x2
>>> m3
array([[1., 0.],
       [0., 1.]])
>>> m4=np.diag([1,2,4]) #matrice dont le diagonale est [1,2,4]
>>> m4
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 4]])
>>> #matrice de taille 2x2 à valeurs aléatoires uniformes sur [0, 1]
>>> m5=np.random.rand(2,2)
>>> m5
array([[0.79503746, 0.26643047],
       [0.40068168, 0.35370144]])
```

Manipulation du module numpy

❖ L'accès aux éléments d'un vecteur:

On accède aux éléments d'un vecteur par **indice**. Les indices commencent à **0** et finissent à **N-1**. On accède à un élément du vecteur avec la notation entre crochets **Vec[indice]**.

```
>>> import numpy as np
>>> v=np.array([1,2,7,9,3,5,6])
>>> v[0]# récupérer le 1er élément
1
>>> v[-1]# récupérer le dernier élément
6
>>> v[1:5]#récupérer la sous vecteurs[2,7,9,3]
array([2, 7, 9, 3])
>>> v[:]#récupérer tous les éléments
array([1, 2, 7, 9, 3, 5, 6])
```

Manipulation du module numpy

❖ L'accès aux éléments d'une matrice:

On accède aux éléments d'une matrice par **indice i de ligne et indice de colonne j**. On accède à un élément d'une matrice avec la notation entre crochets **Mat[i,j]**. (alors qu'avec une matrice python "classique" il faudrait écrire **Mat[i][j]**).

```
>>> import numpy as np
>>> m1=np.array([[1,2,3],[4,5,6],[7,8,9]])#matrice de 3x3
>>> m1
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> m1[1,2]#Récupérer l'élément de ligne 2 et de la colonne 3
6
>>> m1[1]#Récupérer les éléments de la ligne 2
array([4, 5, 6])
>>> m1[-1]#Récupérer les éléments de la dernière ligne
array([7, 8, 9])
>>> m1[2,:]#récupérer les éléments de la 3 ligne
array([7, 8, 9])
>>> m1[:,1]#récupérer les éléments de la 2 colonne
array([2, 5, 8])
>>> m1[1:3,0:3]#Récupérer la sous matrice
array([[4, 5, 6],
       [7, 8, 9]])
```

Manipulation du module numpy

❖ Les opérations sur les tableaux numpy:

Les opérations usuelles (addition, soustraction, multiplication, division) s'appliquent aux tableaux **Numpy (Vecteur/Matrice)** en opérant coefficient par coefficient.

```
>>> import numpy as np
>>> a=np.array([[1,4],[2,3]]) #matrice a
>>> b=np.ones((2,2)) #matrice b
>>> a*b #produit
array([[1., 4.],
       [2., 3.]])
>>> a+b #addition
array([[2., 5.],
       [3., 4.]])
>>> a-b #soustraction
array([[0., 3.],
       [1., 2.]])
>>> a/b #divison
array([[1., 4.],
       [2., 3.]])
>>> a**2 # puissance
array([[ 1, 16],
       [ 4,  9]], dtype=int32)
```

Les méthodes sur les tableaux numpy

Méthode	Description
<code>np.size(V/M)</code>	Renvoie le nombre d'éléments d'un vecteur ou d'une matrice.
<code>np.shape(M)</code>	Renvoie les dimensions d'une matrice
<code>np.sum(V/M)</code>	Somme des éléments d'un vecteur ou d'une matrice
<code>np.sum(M,axis=0)</code>	Somme des lignes d'une matrice (addition vertical).
<code>np.sum(M,axis=1)</code>	Somme des colonne d'une matrice (addition horizontal).
<code>np.vdot(V1,V2)</code>	Produit scalaire de deux vecteurs
<code>V.min()</code> ; <code>V.max()</code> ; <code>V.sum()</code>	Minimum, maximum et somme des éléments d'un vecteur
<code>M.min()</code> ; <code>M.max()</code> ; <code>M.sum()</code>	Minimum, maximum et somme des éléments d'une matrice
<code>M.sum(axis=0)</code> ; <code>M.min(axis=0)</code>	Somme des lignes et minimum des lignes d'une matrice
<code>M.cumsum(axis=0)</code> ; <code>M.cumsum(axis=1)</code>	somme cumulée des lignes et somme cumulée des colonnes

Les méthodes sur les tableaux numpy

Méthode	Description
<code>np.cross(V1,V2)</code>	Produit vectoriel de deux vecteurs.
<code>np.linalg.inv(M)</code>	Inverse d'une matrice
<code>np.transpose(M)</code>	Transposée d'une matrice
<code>np.dot(M1,M2)</code>	Produit matriciel
<code>np.linalg.det(M)</code>	Déterminant d'une matrice
<code>np.diag()</code>	Création Matrice diagonale
<code>np.trace(M)</code>	La trace d'une matrice
<code>np.linalg.matrix_power(M,n)</code>	M à la puissance n
<code>np.sin(M); np.cos(M); np.tan(M)</code>	Les fonctions trigonométriques de base
<code>np.linalg.solve(A,B)</code>	résolution du système linéaire $A x = B$