Manipulation des Vecteurs et des Matrices avec Numpy

On utilise le package `numpy` pour construire et manipuler les vecteurs/matrices. On commencera donc par charger ce package:

```
In []: import numpy as np
```

Tous les appels à des fonctions de ce package seront donc précédés de **np**.

1 Les vecteurs

1.1 Définition

a) A la main

On peut définir chacun des éléments de la matrice:

```
In []: v = np.array([1.,2.,3.]);
```

b) Pas constant : `arange`

Si on souhaite construire un vecteur dont les composantes varient d'un pas constant, on peut utiliser l'instruction `np.arange(a,b,p)` qui désigne le vecteur :

$$(a, a+p, a+2p, \cdots, a+np),$$

avec n le plus grand entier strictement inférieur à la partie entière de $(b-a)/p$.

Attention, par conséquence, si $(b-a)/p$ est entier, le nombre b ne sera pas atteint. Si $(b-a)/p$ n'est pas entier, b sera atteint...

```
In []: np.arange(0,10)
Out[]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In []: np.arange(3,0,-0.5) # le pas peut etre negatif
Out[]: array([ 3. ,  2.5,  2. ,  1.5,  1. ,  0.5])

In []: np.arange(2,3,0.2) #(3-2)/0.2 est entier
Out[]: array([ 2. ,  2.2,  2.4,  2.6,  2.8])

In []: np.arange(2,3.2,0.2) #(3.2-2)/0.2 n'est pas entier
Out[]: array([ 2. ,  2.2,  2.4,  2.6,  2.8,  3. ,  3.2])
```

c) Pas constant : `linspace` Pour un vecteur à pas constant, si cette fois on veut indiquer le nombre d'éléments, on peut utiliser la commande `linspace(a,b,num=n)` qui construit:

$$(a, a+\frac{b-a}{n-1}, a+2\frac{b-a}{n-1}, \cdots, a+(n-2)\frac{b-a}{n-1}, b).$$

```
In [ ]: np.linspace(2.0, 3.0, num=5)
Out[ ]: array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])

In [ ]: np.linspace(2.0, 3.0, num=5, endpoint=False)
Out[ ]: array([ 2. ,  2.2,  2.4,  2.6,  2.8])

In [ ]: x,pas=np.linspace(2.0, 3.0, num=5, retstep=True)
In [ ]: x
Out[ ]: array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])

In [ ]: pas
Out[ ]: 0.25
```

1.2 Accès aux éléments

Les indices des tableaux commencent à 0. Les indices peuvent être négatifs: ils désignent les éléments comptés à partir de la fin.

```
In [ ]: x  = np.arange(1,5);
In [ ]: x[0]                    # Premier element
Out[ ]: 1

In [ ]: x[:]                    # Tous les elements
Out[ ]: array([1, 2, 3, 4])

In [ ]: x[-1]                   # Dernier element
Out[ ]: 4

In [ ]: x[-2]                   # Avant-dernier element
Out[ ]: 3
```

On peut avoir accès à des morceaux de vecteurs: deb(inclu):fin(exclu):increment
Par défaut `deb` vaut 0, `fin` vaut -1 et `increment` vaut 1.

```
In [ ]: x[1:3]                  # L'increment est 1 par defaut
Out[ ]: array([2, 3])

In [ ]: x[-3:-1]                # L'increment est 1 par defaut
Out[ ]: array([2, 3])

In [ ]: x[2:]                   # On prend jusqu'au dernier
Out[ ]: array([3, 4])

In [ ]: x[1::2]                 # L'incr\'ement vaut 2
Out[ ]: array([2, 4])

In [ ]: x1[::-1]                # Lecture de x a l'envers
```

1.3 Copies de vecteurs

```
In []: a = np.array([1, 2, 3], float)
In []: b = a
In []: c = a.copy()
In []: a[0] = 0   # On modifie a
In []: a
Out[]: array([0., 2., 3.])

In []: b          # b a ete modifi\'e
Out[]: array([0., 2., 3.])

In []: c
Out[]: array([1., 2., 3.]) # c n'a pas \'et\'e modifi\'e
```

Attention, l'instruction `b=a` n'effectue une copie que de l'adresse d'un vecteur.

1.4 Taille des vecteurs

```
In []: a=np.zeros(5)
In []: a
Out[]: array([0., 0., 0., 0., 0.])

In []: np.alen(a)
Out[]: 5

In []: a.size
Out[]: 5

In []: a.shape
Out[]: (5,)
```

1.5 Opérations sur les vecteurs

Des fonctions prédéfinies sont à votre disposition pour manipuler les vecteurs :

`np.sum(v)`	additionne tous les éléments du vecteur v.
`np.prod(v)`	multiplie tous les éléments du vecteur v.
`np.max(v)`	calcule le plus grand élément du vecteur v.
`np.min(v)`	calcule le plus petit élément du vecteur v.

2 Les Matrices

2.1 Définition

a) A la main

```
In []: A = np.array([[1,2,3],[4,5,6]]);
```

b) Matrices particulères

`np.eye(n)`	matrice identité de taille n.
`np.zeros((n,m))`	matrice de taille n×n ou n×m remplie de zeros.
`np.ones((n,m))`	matrice de taille n×n ou n×m remplie de uns.
`np.random.rand(n,m)`	matrice de taille n×n ou n×m remplie de nombres aléatoires.
`np.diag(v)`	matrice dont la diagonale est le vecteur v.

Pour toutes ces fonctions, on peut préciser le type des données: np.zeros((n,m),dtype=float).

Si on ne précise qu'un seul argument, on obtient un vecteur:

```
In [ ]: a = np.zeros(5)
In [ ]: a
Out[ ]: array([ 0.,  0.,  0.,  0.,  0.])
In [ ]: a.shape
Out[ ]: (5,)
```

2.2 Informations sur les matrices

Taille des matrices:

```
In [ ]: E= =np.array([[1,2],[3,4]])
In [ ]: n,m=E.shape
```

Type des éléments de la matrice:

```
In [ ]: a = np.array([1, 2, 3], int)
In [ ]: a.dtype
Out[ ]: dtype('int64')
```

On a également les types bool, float, complex...

2.3 Accès aux éléments

```
In [ ]: A=np.array([np.arange(0,3),np.arange(0,3)*2])
Out[ ]: array([[0, 1, 2],
               [0, 2, 4]])

In [ ]: A[1,:] # Acces \'a la 2eme ligne
Out[ ]: array([0, 2, 4])
```

2.4 Opérations sur les matrices

a) Transposition

Pour transposer une matrice (ne fonctionne pas sur les vecteurs):

```
In [ ]: A=np.array([[1,2,3],[4,5,6]])
In [ ]: B=np.transpose(A);
```

b) Opérations élémentaires

Les opérations élémentaires comme les additions, multiplications, divisions et puissances, se font terme à terme.

```
In []: A=np.array([[1,2,3],[1,1,1]]);
In [];: B=np.array([[1,1,1],[3,2,1]]);
In [];: A*B
Out[]: array([[1, 2, 3],
              [3, 2, 1]])
```

c) Algèbre linéaire

`np.dot(A,B)`	Produit des matrices A et B
`np.vdot(x,y)`	Produit scalaire des vecteurs x et y
`np.linalg.inv(A)`	Inverse de A
`np.linalg.solve(A,b)`	Résolution de $AX = b$
`np.linalg.det(A,b)`	déterminant de A
`np.trace(A)`	trace de A
`np.rank(A)`	rang de A

d) Autres opérations

```
# Calcul du maximum, par colonne, par ligne
In [];: A=np.array([[1,10,3],[5,2,7]])
In [];: np.amax(A,axis=0)  # Maximum sur chaque colonne
Out[]: array([ 5, 10,  7])
In [];: np.amax(A,axis=1)  # Maximum sur chaque ligne
Out[]: array([10,  7])
```

Les fonctions `sum` et `prod` fonctionnent de la même manière.

```
# Conversion en entier d'une matrice de float
In [];: A=np.random.rand(2,2)*10
Out[]: array([[4.49206821, 1.96302826],
              [8.03873712, 1.47807917]])

In [];: A.astype(int)
Out[]:array([[4, 1],
             [8, 1]])
```

2.5 Manipulation des lignes et colonnes des matrices

a) Suppression de ligne ou colonne

```
In [];: A=np.array([[1,2,3],[1,1,1],[2,2,2]]);
In [];: B=np.delete(A,0,axis=0);B  # supprime la 1ere ligne
Out[]: array([[1, 1, 1],
              [2, 2, 2]])

In [];: C=np.delete(A,1,axis=1);C  # supprime la 2eme ↩
    colonne
Out[]: array([[1, 3],
```

```
                [1, 1],
                [2, 2]])
```

Attention, ces instructions ne modifient pas la matrice A.

b) Ajout de ligne ou colonne

Les instructions suivantes permettent d'ajouter une ligne et une colonne

```
In []: np.insert(A,1,np.array([5, 5, 5]),axis=0)
Out[]: array([[1, 2, 3],
              [5, 5, 5],
              [1, 1, 1],
              [2, 2, 2]])
```

c) Redimensionnement

```
In []: a = np.array(range(6)); a
Out[]: array([0, 1, 2, 3, 4, 5])

In []: a.reshape(2,3)
Out[]: array([[0, 1, 2],
              [3, 4, 5]])
```

d) Mettre bout à bout des matrices

On peut ajouter des lignes:

```
In []: a=np.array([[1,2,3],[9,1,0]])
In []: b=np.vstack((a,np.array([8,8,8])))
In []: b
Out[]: array([[1, 2, 3],
              [9, 1, 0],
              [8, 8, 8]])
```

Pour ajouter une matrice en colonne, il faut redimensionner le vecteur

```
In []: a=np.array([[1,2,3],[9,1,0]])
In []: b=np.hstack((a,np.array([8,8]).reshape(2,1)))
In []: b
Out[]: array([[1, 2, 3, 8],
              [9, 1, 0, 8]])
```

e) Commande `tile` pour répéter une matrice

Cette fonction permet de construire une matrice à partir de la répétition en ligne et en colonne d'une autre matrice.

```
In []: A=np.array([[1,2],[3,4]])
In []: np.tile(A,[2,1])
Out[]: array([[1, 2],
              [3, 4],
              [1, 2],
              [3, 4]])
```

3 Sauvegarder des matrices dans un fichier

```
In []: A=1.978*np.ones((3,2))
In []: np.savetxt('matA.txt', A, fmt='  %6.4f')
In []: B=np.loadtxt('matA.txt')
```