

Introduction

Cette partie sera consacrée à la librairie numpy qui est largement utilisée pour la manipulation des tableaux de données. En gros, dès qu'il s'agit d'un tableau de dimension quelconque, on pense à numpy !!

Dans le domaine de l'apprentissage, les données peuvent avoir plusieurs formats numériques qui seront par la suite traduites en **tableaux** ou **dataframe** avant toute manipulation.

Par exemple, une image numérique en niveaux de gris est un tableau de deux dimensions (matrice), un texte peut être représenté par un tableau (word2vecor), un fichier .csv contenant des informations sur les cartes GPU et leurs performances sur 20 ans, etc.

Il est à noter que le codage ainsi que la manipulation des tableaux numériques est une partie fondamentale pour une application rapide et efficace.

Pour les petits tableaux on pourrait se contenter des listes mais, comme nous allons le voir par la suite, il est recommandé de se familiariser avec les tableaux avec l'aide de numpy

```
In [ ]: # comme n'importe quelle librarire, il faut commencer par la charger
        # à l'aide de la commande import
```

```
import numpy
# maintenant que c'est fait on peut utiliser son contenu
# par exemple vérifier la version installer
numpy.__version__
```

```
In [ ]: # et si on lui donne un nom pour faciliter les appels
import numpy as np
np.__version__
```

```
In [ ]: # par la suite, si on ne se rappelle pas d'une commande, on peut se faire ai
        #par tabulation (magique)
```

```
#np.TAB : on choisit dans la liste ou on commence par ...
a=np.array([-1, -3, -10, 1])
print(a)

a=np.abs(a)
print('après la valeur absolue',a)

print(a.max())
#type(a)
print('Le min',a.min())
```

Avant de commencer à manipuler les tableaux, il est important de savoir ce qu'on manipule : type, structure, indices, etc.

Afin de remplir des tableaux aléatoirement, on va souvent utiliser les fonctions de la classe **random**

```
In [ ]: #numpy a été déjà chargé, la ligne suivante est juste pour nous rappeler  
np.random.seed(20) # important pour reproduire les mêmes data  
a1 = np.random.randint(5,15, size=5) # tableau : une seule dimension  
print('a1=', a1)
```

```
In [ ]: print('-----\n')  
a2 = np.random.randint(10, size=(3, 4)) # tableau : 2 dimensions  
print('a2= \n',a2)
```

```
In [ ]: #pour connaître les propriétés d'un tableau  
print("a2 ndim: ", a2.ndim)  
print("a2 shape:", a2.shape)  
  
print('nbre de lignes',a2.shape[0])  
print('nbre de colonnes',a2.shape[1])  
#remarquer que shape contient tous les autres informations : c'est le  
#premier reflex dès qu'il s'agit d'un tableau...
```

```
In [ ]: print('-----\n')  
a3 = np.random.randint(10, size=(3, 4, 5)) # tableau : 3 dimensions  
print('a3=', a3)
```

```
In [ ]: print("une dimension", a3.shape[2])  
print(a2.shape[0], 'lignes')  
print(a2.shape[1])
```

```
In [ ]: #pour connaître le type  
print("le type de ce tableau est : ", a2.dtype)
```

```
In [ ]: #qui dit tableau dit indice  
print(a1, 'a comme taille : ', a1.shape)  
#il s'agit d'un tableau de trois lignes
```

```
In [ ]: #on affiche un élément si on connaît un indice valide qui  
#commence par ZERO et finit par taille-1 (équivalent -1)  
print(a1)  
print('Le premier élément est :',a1[0])  
print('Le2ème élément est :',a1[-1])  
print('Le dernier élément est :',a1[-2])
```

```
In [ ]: print('Le nombre d\'éléments est ', a1.shape[0], 'du tableau',a1)
print('-----')
#Afficher tous les éléments grâce à shape
for i in range(a1.shape[0]):
    print(a1[i]**2)
    print('-----')
```

```
In [ ]: print(a1**2)
```

```
In [ ]: #Afficher et vérifier le dernier élément
a1[-1]
```

Même principe pour des dimensions supérieures

```
In [ ]: #pour connaître le type
print("le type de ce tableau est : ", a2.dtype)
```

```
In [ ]: #qui dit tableau de 2 dimensions dit indice 2 indices
print(a2, ' \n a pour taille : \n', a2.shape )
#il s'agit d'un tableau de trois lignes
#print(a2.shape[1])
```

```
In [ ]: #on affiche une ligne si on connaît un indice valide
print('-----')
print('Tableau : \n',a2)
print('-----')
print('élément 0 ème ligne et 0 ème colonne : ',a2[1,3])
# Ou une colonne
print('-----')
print('La 1ère ligne du tableau : ',a2[0,:])
print('-----')
print('La 3ème colonne du tableau : ',a2[:,-1])
#il s'agit de la 3 ème colonne

#print('un bloc : ',a2[0:3,0:2])
```

```
In [ ]: ## Juste des parties d'un tableau (slicing ou sous-tableau)
a = np.arange(5,10)
print('-----')
print('Le contenu du tableau : ',a)
print('-----')
print('Le type du tableau : ',type(a))
```

```
In [ ]: print('A partir du 2ème élément : ',a[1:]) #2 ème élément
```

```
In [ ]: print(a) #rien ne change dans le tableau
```

```
In [ ]: a=np.array([6, 7 ,8, 9])
print(a,'\n')
#print(a[1:3]) #2 et 3 ème élément
print('-----')
```

```

print(a[::5]) #début jusqu'au n-1 ème (n=5)
#print('-----')
print(a[0:5:2]) #par pas de deux
#print('-----')
print(a[::2]) #par pas de deux
#print('-----')
print(a[::-1])#order inversé

```

Changer le format (taille) d'un tableau

```

In [ ]: tab = np.random.randint(10,size=(3,6))# 3 lignes et 4 colonnes
print(tab,'\n')
print('nombre de lignes :',tab.shape[0])
print('-----')
print('nombre de colonnes : ',tab.shape[1])

```

```

In [ ]: print(tab)
tab=tab.ravel()
print('-----\n')
print(tab)

```

```

In [ ]: # si on veut transformer le tableau en une ligne :
# il nous faudra 3x4=12 éléments
a=tab
print(tab,'\n')
a=a.reshape(a.shape[0]*a.shape[1],1)#c'est possible
print('nombre de lignes :',a.shape[0])
print('-----')
print('nombre de colonnes : ',a.shape[1])

print(a.reshape([2,6]))

```

```

In [ ]: print('-----')
a=a.reshape((3,4)) # retour
print('nombre de lignes :',a.shape[0])
print('-----')
print('nombre de colonnes : ',a.shape[1])

```

On peut fusionner plusieurs tableaux si on fait attention aux dimensions

```

In [ ]: x = np.array([1, 2, 3,-1]) # tableau 1 avec 4 elts
y = np.array([0, 6, 7, 8, 9]) # tableau 2 avec 5 elts
z=np.array(np.concatenate([ x,y,x])) #fusion de 1&2 en faisant attention aux
print('nombre de lignes et de colonnes dans z :',z.shape)
print('-----')
w=np.array(np.concatenate([x, y,z])) #fusion de 1&2&3
print('nombre de lignes et de colonnes dans w: \n',w.shape)

print(z)
z.mean()

```

```
In [ ]: z.max()
```

Les fonctions ou méthodes (pré-définies)

```
In [ ]: #l'avantage est de faire un seul appel pour tout le tableau sans boucle
x = np.array([-2, -1, 0, 1, 2])
print('-----')
print('La valeur absolue appliquée à un tableau : \n',np.abs(x))
print('-----')
print('La valeur absolue appliquée à un tableau plus grand : \n',np.abs(a3))
```

```
In [ ]: #Exemples de fonctions : fonctions trigonométriques
theta = np.linspace(0, np.pi, 3) #très pratique
y=np.cos(theta)
print(y) #cos, tan, arcsin, etc (à l'aide de TAB)
```

```
In [ ]: import matplotlib.pyplot as plt
#la fameuse biblio pour l'affichage
# A quoi sert la ligne suivante ?
%matplotlib inline

theta = np.linspace(0, np.pi, 100)
y=np.sqrt(theta)*np.sin(-theta**3)**2
plt.plot(theta,y,'r-o')
```

```
In [ ]: # log et exp
a=np.array([1, 2, 4, 16])
print(np.log(a))#log_n
print(np.exp(a))#exp_n
#il y a aussi d'autres fonctions : max, min, sum, mean,std, power, add.accum
```

```
In [ ]: a=np.linspace(-10,10,1000000)
y=1/(np.sqrt(2*np.pi))*np.exp(-.5*(a-5)**2)

print(a.shape)
print(y.shape)
plt.plot(a,y,'b*')
```

```
In [ ]: print('- la moyenne : ', np.mean(a))
print('- l\'écart type : ',np.std(a))
```

```
In [ ]: somme=0
for i in range(a.shape[0]):
    somme+=a[i]
print(somme/a.shape[0])
```

```
In [ ]: Tab = 5+np.random.random((3, 4))*(10-5)
print(Tab)

#?np.random.random
```

```
In [ ]: print('La somme des éléments du tableau est :', y.cumsum())
        #print(Tab.cumsum())
        print('taille y \n ', y.shape[0])
        b=y.cumsum()/y.shape[0]
        plt.plot(a,b,'r-')
```

```
In [ ]: print(Tab.shape)
        print(Tab.min(axis=0))#max, mean, sum... par ligne ou par colonne
        np.sum(Tab.ravel())
```

Opérations booléennes

```
In [ ]: Tab1=np.ones((3,4))#tableau de 0 utile pour initialiser
        print(Tab1)

        print(Tab1.any()) #test de tous les éléments
        print('-----')

        Tab2=np.ones((2,3))#tableau de 1 utile pour initialiser
        print(Tab2.any())
        print(Tab2.all())

        print('-----')
        Tab3=np.random.randint(5,size=(2,3))
        print(Tab3,'\n')

        print(Tab3.any())
        print(Tab3.all())
```

Un petit mot sur l'affichage

il s'agit de la librairie matplotlib qui ressemble à celle de scilab mais encore plus puissante

En voilà un avant-goût

```
In [ ]: #charger la librairie comme plus haut (rappel)
        import matplotlib.pyplot as plt

        theta = np.linspace(-np.pi,np.pi,100)
        y=np.cos(theta)
        plt.plot(theta,y,'red',label='cos')

        y=np.sin(theta)
        plt.plot(theta,y,color='green', linestyle='dashed', marker='o',
                 markerfacecolor='blue', markersize=2, label='sin')

        plt.ylim([-1.1,1.4])
```

```
plt.legend(loc='upper right',ncol=2, mode="expand", shadow=True, fancybox=True)
plt.title('cos & sin')
```

```
In [ ]: # un sondage sur la taille
nb_simulations=100000
mu=2.5 #mean
scale=0.75#std

#on génère un échantillon selon la loi normale
y=np.random.normal(mu,scale,nb_simulations)
#on affiche l'histogramme
plt.hist(y,250,density=True,color='m')
plt.title('Distribution de la taille')
plt.xlabel('Taille')
plt.ylabel('densité')
```

```
In [ ]: ?np.sin
```

Créer une fonction et l'appliquer sur tout le tableau (sans boucle)

Affichage avec subplot

```
In [ ]: def fct_test(t):
        return np.sin(t**2)+5*np.exp(-t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.subplot(224) # 2 lignes et 1 colonnes : position 1
plt.plot(t1, fct_test(t1), "ko")
plt.plot(t2, fct_test(t2), "b")

plt.subplot(221) # 2 lignes et 1 colonnes : position 2
plt.plot(t2, t2*np.cos(2*np.pi*t2), "r+")
plt.show()
```

```
In [ ]: fct_test(5)
```

```
In [ ]: xx=np.linspace(-10,10,1000)
plt.plot(fct_test(xx))
```

```
In [ ]: plt.plot(t2, np.cos(2*np.pi*t2), "r--")
```

```
In [ ]: #?np.sum
```

pour avoir plus de détails ou besoin d'aide:

`?np.random.normal`

`?plt.hist`

Pratique

Exercice 1

- 1- Générer un tableau aléatoire d'entiers entre 0 et 15 et de dimension (1,100)
- 2- Calculer et afficher : moyenne, max, min, écart type
- 3- Générer un tableau aléatoire de réels entre -1 et 1 et de dimension (1,10000)
- 4- Calculer et afficher : moyenne, max, min, écart type

Exercice 2

- 1- Créer une matrice aléatoire de taille (5x6)
- 2- Remplacer une colonne sur deux, sauf la première colonne, par sa valeur moins le double de la colonne suivante
- 3- Remplacer les valeurs négatives par 0 en utilisant un test booléen sans boucle

Exercice 3

- 1- Choisir une image de niveaux de gris à récupérer sur votre home ou en ligne
- 2- Charger cette image dans une variable *I* et Afficher *I* avec l'aide de ce code :

```
*import matplotlib.image as im
*I=im.imread('votre-image.format')
*import matplotlib.pyplot as plt
*plt.imshow(I)
```

- 4- Afficher la taille de *I* et vérifier qu'elle est en niveaux de gris
- 5- Transformer *I* en un tableau d'une seule colonne
- 6- Calculer et afficher l'histogramme de *I*
- 7- Soustraire la moyenne de *I* et afficher le résultat
- 6- Calculer le nouveau histogramme
- 8- Commenter

Exercice 4

- 1- Créer 9 matrices aléatoires de taille (128x128)
- 2- Afficher les matrices comme des images avec l'aide de subplot
- 3- Créer et tester une fonction qui permet de calculer l'image moyenne de toutes ces images
- 4- Lire l'image de Lena dans un tableau I (dispo sur l'ent et sur le net).
- 5- Créer 10 matrices M1, M2, .. M10 aléatoire de 0 et 255 et qui ont la même taille que I
- 6- Ajouter chaque image aléatoire à I tel que $J_i = I + M_i$, $i=1:10$
- 7- Calculer et afficher l'image moyenne de J1, J2, .., J10
- 8- Commenter

Calcul de l'intégrale

```
In [ ]: xmin = 0
xmax = np.pi
nbx = 30
nbi = nbx - 1 # nombre d'intervalles

x = np.linspace(xmin, xmax, nbx)
y = 1 + np.cos(x)
plt.plot(x,y,"bo-")

integrale = 0
for i in range(nbi):
    integrale = integrale + .5*(y[i+1]+y[i])*(x[i+1]-x[i])
    # dessin du rectangle
    x_rect = [x[i], x[i], x[i+1], x[i+1], x[i]] # abscisses des sommets
    y_rect = [0, y[i], y[i], 0, 0] # ordonnees des sommets
    plt.plot(x_rect, y_rect,"r")
print("integrale =", integrale, 'alors que la valeur exacte est ', np.pi)

plt.show()
```

Sauvegarde d'un tableau

```
In [ ]: # Pour des tableaux de dimension trois ou plus,
#il faut utiliser np.save

x = np.linspace(0, 1, 100)
a=1/np.sqrt(2*np.pi*20)*np.exp(-20*(x-.5)**2)
np.savetxt('tableau.txt', a)# en fichier texte à vérifier avec un éditeur
```

```
np.save('tableau.npy', a) # en fichier numpy rapide et facile à charger

import os
#charger les fichier
#os.path.getsize('tableau.npy') pour avoir la taille
var=np.load('tableau.npy') #charger dans une variable var
print('La taille du tableau est : ',var.shape)
ax=np.linspace(0,1,var.shape[0])
plt.plot(ax,var,'g-o')
```

Application

A l'aide des fonction *sin* et *cos* créer une figure qui ressemble à celle donnée en dessous

```
In [ ]: import matplotlib.image as im
I=im.imread('Figure_diff.png')
plt.imshow(I)
```

```
In [ ]: I=im.imread('test.png')
plt.imshow(I)
```

Surprise me !!