Module numpy (numerical python)

```
import numpy as np
```

- Tableaux d'éléments de même type
- Opérations mathématiques rapides sur des tableaux
  Exemple: `np.sin(tableau_ndarray)`
- Algèbre linéaire, nombres aléatoires, transf. de Fourier

Création de tableaux (type ndarray)

Tableaux à 1d : vecteur de *n* composantes

`np.zeros(n)`	vecteur de composantes 0
`np.ones(n)`	vecteur de composantes 1
`np.empty(n)`	vecteur non initialisé

Tableaux à 2d : *matrice de n lignes et m colonnes*

`np.zeros((n,m))`	matrice nulle
`np.identity(n)`	matrice identité
`np.eye(n, m, k)`	matrice avec 1 sur $k^{ième}$ diagonale
`np.diag([valeurs], k)`	matrice diagonale (cas k=0)

Tableaux à *d* dimensions : taille $N_1 \times N_2 \times ... \times N_d$ (cas général)

```
T = np.zeros(shape, dtype)
```

nb de valeurs dans chaque dimension: tuple (N1, N2, N3, ...)

(optionnel) ***type*** *des éléments:*

`float`	*réel* (`np.float16`, ..., 64)
`complex`	*nb complexe* (`np.complex64`, 128)
`int`	*entier* (`np.int8`, 16, 32, 64)
`np.uint`	*entier positif* (`np.uint8`, ..., 64)

```
T = np.random.rand(N1, N2, ...)
```

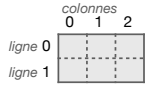
Crée un tableau de forme N_1, N_2, ... rempli de nombre aléatoires (loi de probabilité uniforme sur l'intervalle [0,1[ ).

```
T = np.fromfunction(function, shape)
```

Crée un tableau initialisé avec une fonction (voir help(np.fromfunction)).

Accès aux éléments

Exemple:
```
for i in range(3):
    for j in range(3):
        T[i,j] = ...
```
no de ligne / *colonne*



En général: `T[i,j,k,...]`

Vue sur une portion de tableau: `T[début:fin:incrément]` *(pas de copie)*

`T[k,:]`	ligne *k* du tableau	`T[i:i+h, j:j+l]`	sous-matrice *h* × *l*
`T[:,k]`	colonne *k* du tableau		

Copier un tableau `T2 = T1.copy()`

Fichiers de données

- Format texte

```
T = np.loadtxt('data.txt',options)
```

Options:

`skiprows=n`	ignorer les *n* 1res lignes
`comments='#'`	symbole des commentaires (# par défaut)
`delimiter=str`	séparateur entre les valeurs (esp./tab par défaut)
`unpack=bool`	si True: x, y, z = loadtxt(...)
`usecols=(0,2)`	lit les colonnes *0* et *2* uniquement

Fichier data.txt:

```
# x      y      z
1900    3e3    4e3
1920    2876   3e9
```

```
np.savetxt('data.out', fmt='%.3e', autres_options)
```

- Format binaire .npy

```
np.save('data.out', T)
np.load('data.out')
```

numpy.fft: transformées de Fourier

Voir `help(np.fft)` dans une console ipython3 (ou dans Google).

Attributs d'un tableau T

`T.ndim`	nb de dimensions (axes)
`T.shape`	tuple avec nb d'éléments dans chaque dimension `T.shape[0]` : nombre de lignes
`T.size`	nb total d'éléments
`T.dtype`	type des éléments (data-type)
`T.itemsize`	taille d'un élément (octets)

Conversions

list→array: `np.array([[1,2],[3,4]])` → $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$

array→list: `T.tolist()` → `[[1,2],[3,4]]`

`T.astype(dtype)` Copie du tableau converti au type spécifié

Opérations mathématiques

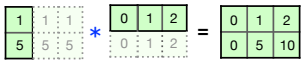
Les fonctions mathématiques de numpy s'appliquent à chaque élémen des tableaux. Ex:
`np.exp(tableau)` ✔ *fonctionne*
`math.exp(tableau)` ⚠ *erreur*

Somme, moyenne, min, max, écart-type (std):
`T.sum()`, `T.mean()`, `T.min()`, `T.max()`, `T.std()`

Somme le long d'un axe:
`T.sum(axis=0)` ⇔ T[0, :] + T[1, :] + ... *= somme des colonnes si 2d*
`T.sum(axis=1)` ⇔ T[:, 0] + T[:, 1] + ... *= somme des lignes si 2d*

Les opérations `+ - * / **` sont effectuées **terme-à-terme.**

Tableau `+` *nombre* ← *effectué sur chaque terme*



Calcul vectoriel et matriciel

- Produit matriciel (tableaux à 2d): `mat1 @ mat2`
- Produit scalaire entre 2 vecteurs (tableaux à 1d): `vec1 @ vec2`
- Produit vectoriel: `np.cross(vec1, vec2)`
- Produit extérieur de 2 vecteurs: `np.outer(vec1, vec2)`
- Norme d'un vecteur: `np.linalg.norm(vec)`
- Trace: `mat.trace()`
- Matrice transposée: `mat.T` ⚠ *sans effet sur un tableau à 1d*

En 1d: *pas de distinction entre vecteur ligne ou colonne.*
Les opérations mat@vec et vec@mat donnent le résultat escompté.

En 2d:
`vec.reshape((3,1))` *crée un vecteur-colonne (matrice de 3 lignes et 1 col.)*
`vec.reshape((1,3))` *crée un vecteur-ligne (matrice de 1 lignes et 3 col.)*

Autre notation : (Python < 3.5)

```
m1 @ m2 ⇔ np.dot(m1,m2) ⇔ m1.dot(m2)
v1 @ v2 ⇔ np.dot(v1,v2) ⇔ v1.dot(v2)
np.transpose(M) ⇔ m.transpose()
```

numpy.linalg: Méthodes d'algèbre linéaire

```
import numpy.linalg as la
```
← *pour abréger la notation*

`la.det(M)`	déterminant de la matrice *M*
`la.inv(M)`	matrice inverse
`la.eig(M)`	valeurs propres
`la.solve(A, B)`	renvoie *X* tel que $A \cdot X = B$
`la.matrix_rank(M)`	rang de M
`la.matrix_power(M, n)`	M^n

numpy.random: nombres aléatoires

`np.random.rand(N1, N2, ...)`
tableau de forme N_1, N_2, ... rempli de nombre aléatoires distribués selon la loi de probabilité uniforme sur l'intervalle [0,1[.

`np.random.randn(N1, N2, ...)` idem pour la loi normale

`np.random.randint(min, max, nb)`
nb nombres entiers aléatoires dans l'intervalle [min, max[(max est exclu).

`np.random.random_integers(min, max, nb)`
nb nombres entiers aléatoires dans l'intervalle [min, max] (max est inclus).

Module matplotlib: création de graphiques

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

Notebook jupyter: - utiliser `%matplotlib inline` (ou notebook) pour l'importation
- instructions `plt.figure()` et `plt.show()` non obligatoires

Création d'un graphique

```
plt.figure(figsize=(8,5))
plt.plot(X, Y, label='mes_données')
plt.xlabel('légende axe x')
plt.ylabel('légende axe y')
plt.legend()
plt.show()
```

création d'un nouveau graphique
X, Y: ***listes ou tableaux numpy*** *contenant les coordonnées x, resp. y, des points.*
légendes des axes
affiche la légende des courbes (voir l'option 'label' de plot)
affiche le graphique

```
plt.xscale('log')
plt.axis('equal')
plt.xlim(xmin, xmax)
plt.savefig('fichier.pdf')
```

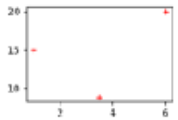
axe x en échelle log
même échelle pour les axes x et y
bornes de l'axe des x
enregistre le graphique

Création de tableaux de valeurs régulièrement espacées

`np.linspace(a, b, n)`	tableau (numpy) contenant *n* valeurs dans l'intervalle [*a*,*b*]
`np.logspace(a, b, n)`	*n* valeurs de 10^{-a} à 10^{-b} régulièrement espacées en échelle log
`np.arange(a, b, dx)`	*n* valeurs de *a* (inclus) à *b* (exclus) par pas *dx*

Graphique de points ou d'une fonction

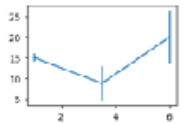
listes ou tableaux numpy



```
X = [1, 3.5, 6]
Y = [15, 8.8, 20]
plt.plot(X, Y, marker='+', ls='', c='r' )
```

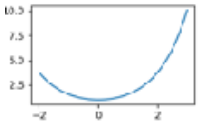
Options de plot():

`marker=`	`'o'`, `'s'`, `'*'`, `'+'`, `'x'`, `'^'`
`linestyle=`	`'-'`, `'--'`, `'_'`, `':'`
`color=`	`'r'`, `'g'`, `'b'`, `'DarkGreen'`, ...



```
X = [1, 3.5, 6]
Y = [15, 8.8, 20]
dY= [1., 4.2, 6.4]
plt.errorbar(X, Y, dY)
```

Graphique d'une fonction

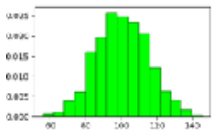


```
X = np.linspace(-2, 3, 100)
plt.plot(X, np.cosh(X))
```

⚠ *utiliser des tableaux numpy et les fonctions mathématiques de numpy*

`np.exp(tableau)`	✔ *correct*
`math.exp(tableau)`	⚠ *erreur*
`math.exp(liste)`	⚠ *erreur*

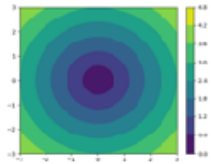
Histogramme



```
X = 100 + 15*np.random.randn(1000)
plt.hist(X, 15, normed=1, facecolor = 'lime', edgecolor = 'green')
```

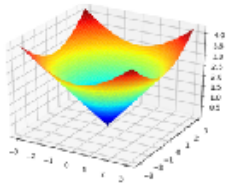
nombre de barres

Graphique en 3d: z = f(x,y)



```
X = np.linspace(-3, 3, 200)
Y = X
XX, YY = np.meshgrid(X, Y)
Z = np.sqrt(XX**2 + YY**2)
plt.contourf(XX, YY, Z)
plt.colorbar()
```

coordonnées selon axe x
coordonnées selon axe y
coordonnées x et y de ***tous les points*** *de la grille*
hauteurs: $z = \sqrt{x^2 + y^2}$
graphique de ***courbes de niveaux*** *(f = filled)*
barre avec légende des couleurs



```
from mpl_toolkits.mplot3d import Axes3D
ax = plt.subplot(111, projection='3d')
ax.plot_surface(XX, YY, Z, cmap='jet')
ax.set_aspect(1)
```

111 signifie graphique numéro 1 dans une grille de 1 x 1 graphiques.
cmap signifie colormap
même échelle pour les axe x, y et z