Ingénierie numérique et simulation avec Python

Tous les scripts contiendront les lignes d'import :

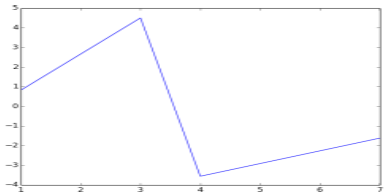
```
import numpy as np
import matplotlib.pyplot as plt
```

ATTENTION : les indentations n'ont pas toujours pu être respectées pour les lignes très longues !

Graphiques

Des points en dimension 2

```
x = [1,3,4,7]
y = 10 * np.random.rand(4) – 5
print(y)
plt.plot(x, y)
plt.show()
```



y contient

```
[ 0.822  4.500 -3.559 -1.619]
```

Les points(x[0],y[0]), …, (x[3], y[3])) sont reliés.

On peut **sauvegarder la figure1 au format désiré.**

```
plt.savefig("ma_fig.jpeg" )
```

Effacer le contenu de la fenêtre courante pour éviter la superposition :

```
plt.clf()
```

Par défaut, les graphiques successifs se superposent dans la même fenêtre (au début : fenêtre 1).

Fermer la fenêtre :

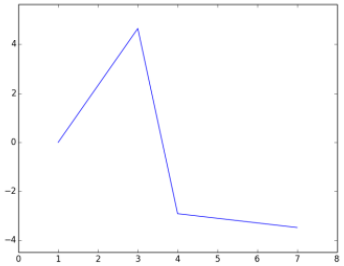
```
plt.close()
```

Label sur les axes :

```
plt.xlabel('Les $x$')
plt.ylabel('Les $y$')
```

Choisir les valeurs extrémales des axes :

```
plt.axis([xmin, xmax, ymin, ymax])
```



```
plt.axis([0, 8, -4.5, 5])
```

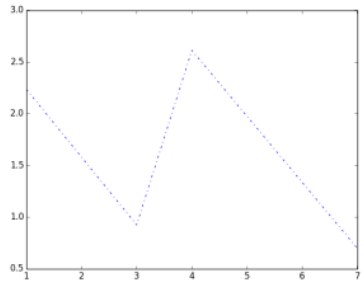
Couleur du trait :

```
plt.plot(x,y, 'g') ➔ en vert
plt.plot(x, y, color = (1,0.5,0))
plt.plot(x, y, color = '#FF99AA')
```

'b' : bleu, 'g' : vert, 'r' : rouge, 'c' : cyan, 'm' : magenta, 'y' : jaune, 'k' : noir, 'w' : blanc

Style du trait : linestyle

```
plt.plot(x, y, linestyle = '-.')
```



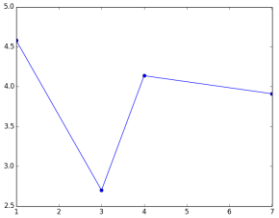
'-' : continu, '- -' : tirets, '-.' : points-tirets, ' :' : pointillé.

Mettre des symboles aux points tracés : marker

options :

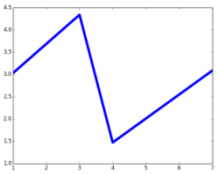
'+' | '*' | ',' |'.' | '1' | '2' | '3' | '4' | '<' | '>' | 'D' | 'H' | '^' | '_' |'d' | 'h' | 'o' | 'p' | 's' | 'v' | 'x' | '|' | TICKUP | TICKDOWN |TICKLEFT | TICKRIGHT

```
plt.plot(x, y, marker = 'o')
```



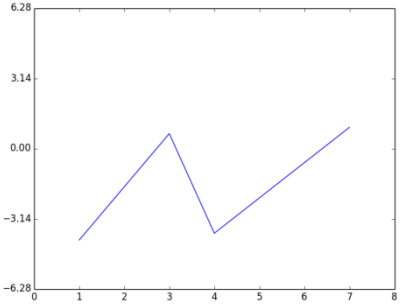
Épaisseur du trait : linewidth

```
plt.plot(x, y, linewidth = 5.6)
```



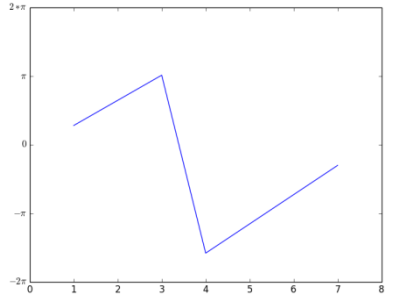
Position des marqueurs des axes :

```
plt.xticks(list(range(9)))
plt.yticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
```



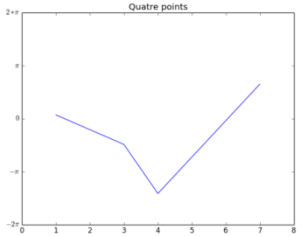
Changer le nom des marqueurs des axes:

```
plt.xticks(list(range(9)))
plt.yticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi],
 [r'$-2\pi$', r'$-\pi$', '$0$',r'$\pi$', r'$2\pi$' ])
```



Titre :

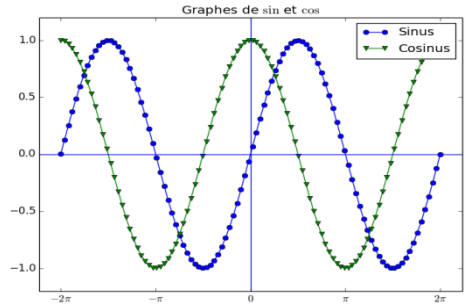
```
plt.title('Mon Titre')
```




```
plt.title('Quatre points')
```

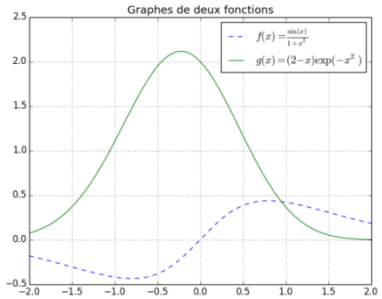
Tracé simultané de plusieurs nuées – Titre – Légende - Axes

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

x = np.linspace(0,4*np.pi,100)
y = np.sin(x)
z = np.cos(x)
plt.plot(x, y, marker = 'o')
plt.plot(x, z, marker = 'v')
plt.xticks([0,np.pi, 2*np.pi, 3*np.pi,4*np.pi],
   [r'$0$', r'$\pi$', r'$2\pi$', r'$3\pi$', r'$4\pi$'])
plt.title('Graphes de $\sin$ et $\cos$')
plt.axis([-1,14, -1.2, 1.2])
plt.axhline()
plt.axvline()
plt.legend(['Sinus', 'Cosinus'])
plt.show()
```



Graphes de fonctions (2D)

1. **Créer le vecteur d'abscisses**
2. **Lui appliquer la fonction vectorisée pour créer le vecteur des ordonnées**

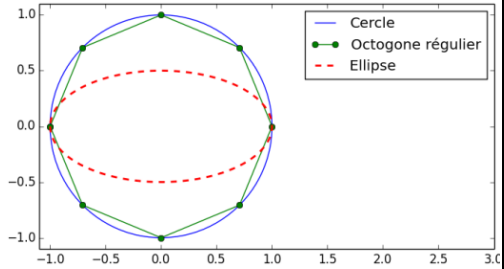



```
import numpy as np
import matplotlib.pyplot as plt
f1 = lambda x: np.sin(x) / (1 + x**2)

f1_v = np.vectorize(f1)
f2 = lambda x : (2 - x) * np.exp(-x**2)
f2_v = np.vectorize(f2)
x = np.linspace(-2,2,100)
y = f1_v(x)
z = f2_v(x)
plt.plot(x, y, linestyle = '--')
plt.plot(x, z, linestyle = '-')
plt.title('Graphes de deux fonctions')
plt.axis([-2,2, -0.5, 2.5])
plt.grid(True)
plt.legend([r'$f(x)=\frac{\sin(x)}{1+x^2}$',
            r'$g(x)=(2-x)\exp(-x^2)$'])
plt.show()
```

Courbes paramétrées

Repère orthonormé :

```
plt.axis('scaled')
```



Échelles logarithmiques

```
plt.semilogx()
plt.semilogy()
plt.loglog()
```

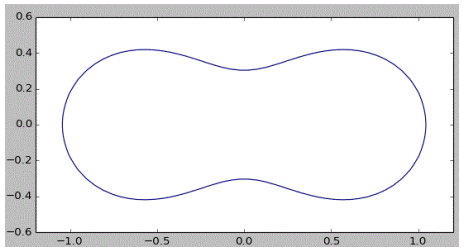
Courbe cartésienne
Courbe de niveau - Géodésique

Courbe de niveau k de la function f :
courbe d'équation cartésienne f(x,y) = k

```
def f(x,y):
    return (x**2+y**2)**2-x**2+y**2

x = np.linspace(-1.2,1.2,50)
y = np.linspace(-0.6,0.6,50)
X,Y = np.meshgrid(x,y)
```

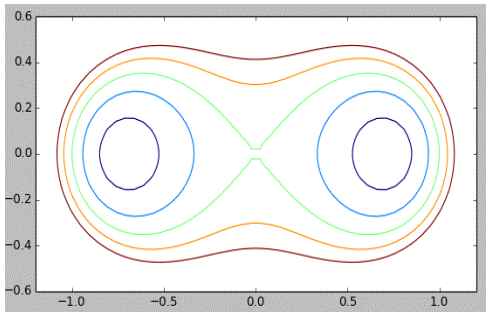
Courbe d'équation f(y,y) = 0.1 :

```
plt.contour(X,Y,f(X,Y), levels = [0.1])
plt.show()
```



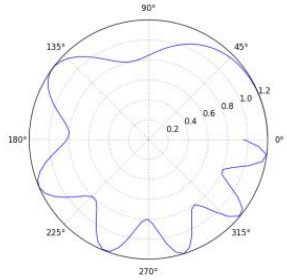
Plusieurs courbes de niveau :

```
plt.contour(X,Y,f(X,Y),
      levels = [-0.2, -0.1, 0, 0.1, 0.2])
```



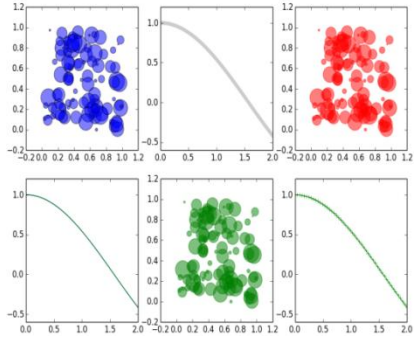
Courbes polaires

```
theta = np.linspace(0,2*np.pi,50)
rho = 1 + 0.2 * np.cos(theta ** 2)
plt.polar(theta, rho)
```



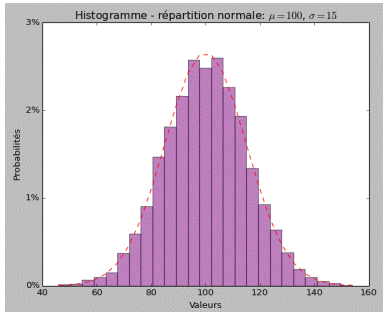
Sous-fenêtres

```
plt.subplot(n_ligne,n_col,n_fenetre)

x = np.linspace(0,2,50)
y = np.cos(x)
X = np.random.rand(50)
Y = np.random.rand(50)
area = np.pi*(15*np.random.rand(50))**2
plt.subplot(2,3,1)
plt.scatter(X, Y, s=area, alpha=0.5)
plt.subplot(2,3,2)
plt.plot(x,y, linewidth = 5, color = '0.8')
plt.subplot(2,3,3)
plt.scatter(X,Y, s=area, alpha=0.5, color='r')
plt.subplot(2,3,4)
plt.plot(x,y)
plt.subplot(2,3,5)
plt.scatter(X, Y, s=area, alpha=0.5, color='g')
plt.subplot(2,3,6)
plt.plot(x,y, linestyle = '-', marker = '+')
```



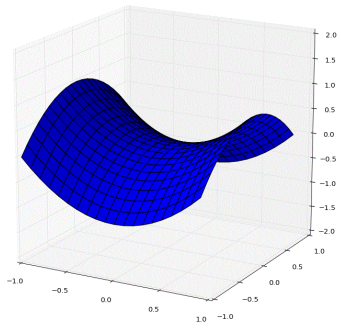
Histogramme

```
import matplotlib.mlab as mlab # probas

# données aléatoires - répartition normale
mu = 100        # moyenne de la distribution
sigma = 15      # écart type de la distribution
x = mu + sigma * np.random.randn(10000)

num_bins = 25
# histogramme des données
n, bins, patches = plt.hist(x, num_bins,
       normed=1, facecolor='purple', alpha=0.5)
# ajouter le tracé de la courbe de la loi
normale
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Valeurs')
plt.yticks([0, 0.01, 0.02, 0.03],
           ['0%', '1%', '2%', '3%'])
plt.ylabel('Probabilités')
plt.title(r'Histogramme - répartition normale:
           $\mu=100$, $\sigma=15$')

plt.show()
```



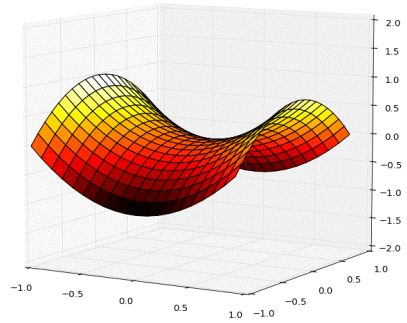
Graphes de fonctions 3D

```
from mpl_toolkits.mplot3d import Axes3D

ax = Axes3D(plt.figure())
X = np.arange(-1, 1, 0.1)
Y = np.arange(-1, 1, 0.1)
X, Y = np.meshgrid(X, Y)
Z = X**2 - Y**2
ax.plot_surface(X, Y, Z, rstride=1, cstride=1)
ax.set_zlim(-2,2)
plt.show()
```



Choix de couleur : *cmap = colormap*

```
ax.plot_surface(X, Y, Z, rstride=1,
                cstride=1,cmap=cm.hot)
```

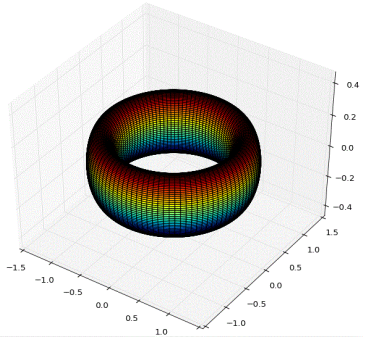


Nappes paramétriques 3D

Tore de Représentation Paramétrique

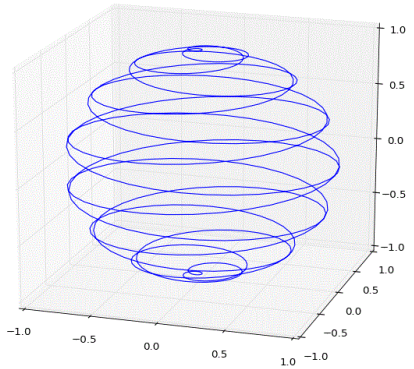
*x = (1 + 0.2 * cos(φ)) * cos(θ)*
*y = (1 + 0.2 * cos(φ)) * sin(θ)*
*z= 0.2 * sin(φ))*

```
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(plt.figure())
theta = np.linspace(0, 2 * np.pi, 100)
phi = np.linspace(0, 2 * np.pi, 100)

X = np.outer((1 + 0.2 * np.cos(phi)),
             np.cos(theta))
Y = np.outer((1 + 0.2 * np.cos(phi)),
             np.sin(theta))
Z = np.outer(np.sin(phi),
         0.2 * np.ones(np.size(theta)))

ax.plot_surface(X, Y, Z, rstride=1,
                cstride=1,cmap=cm.jet)
ax.set_zlim3d(-0.45,0.45)
plt.show()
```



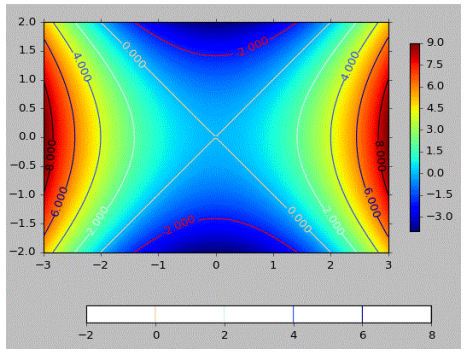
Courbes paramétriques 3D

```
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(plt.figure())
t = np.linspace(0, 2 * np.pi, 500)
x = np.cos(t) * np.cos(15*t)
y = np.cos(t) * np.sin(15*t)
z = np.sin(t)
ax.plot(x,y,z)
plt.show()
```



Courbes de niveau

```
X = np.linspace(-3, 3, 100)
Y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(X, Y)
Z = X**2 - Y**2

# lignes de niveau
CS = plt.contour(X, Y, Z)
plt.clabel(CS, inline=1, fontsize=12)

# coloration de la surface
im = plt.imshow(Z,
              interpolation='bilinear',
              origin='lower',cmap=cm.jet,
              extent=(-3,3,-2,2))

# colorbar pour les lignes de niveau
plt.colorbar(CS,
     orientation='horizontal',shrink=0.8,
     extend='both')

# colorbar pour l'image
plt.colorbar(im, shrink=0.8)

plt.show()
```



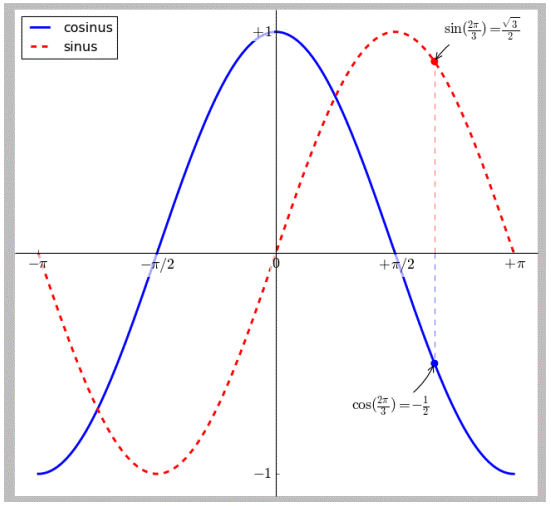
Animations

Animation selon t du graphe de la fonction x→f(x,t) f(x,t)=t*sin(x) ici

```
from matplotlib import animation


fig = plt.figure()
ax = plt.axes( xlim=(0, 4*np.pi), ylim=(-1.1, 1.1))
line, = ax.plot([], [], lw=2)


# tableau des valeurs du parametre t (theta)
t = np.linspace(-2* np.pi, 3 * np.pi, 1000)

# initialisation
def init():
    line.set_data([], [])
    return line,

# animation appelée séquentiellement
def animate(i):
    x = np.linspace(0,4*np.pi,50)
    y = i*np.sin(x)
    line.set_data(x, y)
    return line,

# Tracé de l'animation.
# "blit=True" signifie que seules les parties changées sont tracées
anim = animation.FuncAnimation(fig,
         animate,np.linspace(0, 1, 20),
         init_func=init,  blit=True)

plt.show()
```

Utilisation de pylab

```
# -*- coding: utf-8 -*-
import numpy as np
from pylab import *

figure(figsize=(10,9), dpi=80)
subplot(111)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosinus")
plot(X, S, color="red", linewidth=2.5, linestyle="--",  label="sinus")

ax = gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

xlim(X.min()*1.1, X.max()*1.1)
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

ylim(C.min()*1.1,C.max()*1.1)
yticks([-1, +1], [r'$-1$', r'$+1$'])

legend(loc='upper left')

t = 2*np.pi/3
plot([t,t],[0,np.cos(t)], color ='blue', linewidth=.5, linestyle="--")
scatter([t,],[np.cos(t),], 50, color ='blue')
annotate(r'$\sin(\frac{2\pi}{3}) =\frac{\sqrt{3}}{2}$', xy=(t, np.sin(t)),  xycoords='data',
     xytext=(+10, +30), textcoords='offset points', fontsize=16,
     arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plot([t,t],[0,np.sin(t)], color ='red', linewidth=.5, linestyle="--")
scatter([t,],[np.sin(t),], 50, color ='red')
annotate(r'$\cos(\frac{2\pi}{3})= -\frac{1}{2}$', xy=(t, np.cos(t)), xycoords='data',
          xytext=(-90, -50), textcoords='offset points', fontsize=16,
          arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(16)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.65 ))

show()
```



Algèbre linéaire (numpy)

Matrices et vecteurs

```
a = np.array([1,4,5,7], dtype = float)
b = np.array([[1,2],[3,4]], dtype = int)
```

➔Tous les éléments d'un "array" doivent être de même type ; array ≠ liste.

```
>>> type(a)
< type 'numpy.ndarray'>
```

➔Modifier une donnée d'une extraction d'un tableau entraîne aussi une modification du tableau initial.

➔*np.copy(a)* ou *a.copy()* permettent de faire une copie d'un tableau *a*.

```
>>> a = np.array([1,2,3,4,5])
>>> aa = np.array([1,2,3,4,5])
>>> b = a[1 :3]
>>> b[1] = 0
>>>a
array([1,2,0,4,5]) # a est modifié !
>>>c = np.copy(aa[1 :4]) # méthode 1
>>>c[3] = 10
>>>d = aa[1:4].copy()  # méthode 2
>>>d[3] = 11
>>>aa
array([1,2,3,4,5]) # aa non modifié
```

➔ slicing

```
>>>a = np.eye(5,5)
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]

>>>a[3,4] = 5
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  5.]
 [ 0.  0.  0.  0.  1.]]

>>>a[1:, 3] = 6
 [[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  6.  0.]
 [ 0.  0.  1.  6.  0.]
 [ 0.  0.  0.  6.  5.]
 [ 0.  0.  0.  6.  1.]]
>>>a[0,:] = 9
[[ 9.  9.  9.  9.  9.]
 [ 0.  1.  0.  6.  0.]
 [ 0.  0.  1.  6.  0.]
 [ 0.  0.  0.  6.  5.]
 [ 0.  0.  0.  6.  1.]]
```

➔extraction par tableaux d'entiers (indices)

```
>>>a = np.array(range(2,14), float)
[2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13.]
>>>b = np.array([0,0,0,1,1,2,0], int)
```

➔ matrice des a[b[i], i=0 à len(b) - 1]

```
>>>print(a[b])
[ 2.  2.  2.  3.  3.  4.  2.]
>>>a = a.reshape(4,3)
```

➔ changer la taille

```
[[  2.   3.   4.]
 [  5.   6.   7.]
 [  8.   9.  10.]
 [ 11.  12.  13.]]
>>>c = b -1
[-1 -1 -1  0  0  1 -1]
```

➔ matrice des a[b[i],c[i]] avec len(b) = len(c)

```
>>>print(a[b, c])
[ 4.  4.  4.  5.  5.  9.  4.]
```

➔ matrices spéciales

```
>>>a = np.eye(2,3)
>>>print(a)
[[ 1.  0.  0.]
 [ 0.  1.  0.]]
>>>a = np.ones((2,3))
```

➔ np.ones(t-uple) !

```
>>>print(a)
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
```

➔ np.zeros(t-uple) !

```
>>>a = np.zeros((2,3))
 [[ 0.  0.  0.]
 [ 0.  0.  0.]]
```

➔ vecteurs spéciaux

```
>>> a = np.zeros(6)
[ 0.  0.  0.  0.  0.  0.]
>>> a = np.ones(6)
[ 1.  1.  1.  1.  1.  1.]
>>>a = np.arange(5)
>>>print(a)
[0 1 2 3 4]
>>>a = np.arange(5.)
>>>print(a)
[ 0.  1.  2.  3.  4.]
>>>a = np.arange(5, dtype = np.float)
>>>print(a)
[ 0.  1.  2.  3.  4.]
>>>a = np.arange(0,1,0.2)
>>>print(a)
[ 0. 0.2  0.4  0.6  0.8]
```

➔pas le dernier !

```
>>>a = np.linspace(0,1,6)
```

➔ nb d'éléments

```
>>>print(a)
[ 0.   0.2  0.4  0.6  0.8  1. ]
```

➔ np.diag(vecteur, numSurdiagonale)

```
>>>v = np.arange(4)
>>>a = np.diag(v)
>>>print(a)
[[0 0 0 0]
 [0 1 0 0]
 [0 0 2 0]
 [0 0 0 3]]
>>>b = np.diag(v, 2)
>>>print(b)
[[0 0 0 0 0 0]
 [0 0 0 1 0 0]
 [0 0 0 0 2 0]
 [0 0 0 0 0 3]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
>>>c = np.diag(v, -1)
>>>print(c)
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 0]]
```

➔matrice aléatoire (loi uniforme dans [0,1])

```
>>>a = np.random.rand(2,3)
>>>print(a)
[[0.23084823  0.8247565   0.92051542]
 [0.62923654  0.99323584 0.49146271]]
```

➔reshape, flatten ne modifient pas le 'array'

```
>>>a = np.linspace(0,5,6)
>>>b =a.reshape(2,3)
>>>print(a)
[ 0.  1.  2.  3.  4.  5.]
>>>print(b)
[[ 0.  1.  2.]
 [ 3.  4.  5.]]
>>>b.flatten()
>>>print(b.flatten())
[ 0.  1.  2.  3.  4.  5.]
>>>print(b)
[[ 0.  1.  2.]
 [ 3.  4.  5.]]
```

Opérations matricielles

➔ Concaténer deux matrices

Par défaut : concaténation verticale (axis = 0)

Pour une concaténation horizontale : axis = 1

```
>>>a = np.linspace(0,5,6)
>>>b = a.reshape(2,3)
>>>c = 4 + b
>>>print(b)
[[ 0.  1.  2.]
 [ 3.  4.  5.]]
>>>print(c)
[[ 4.  5.  6.]
 [ 7.  8.  9.]]
>>>print(np.concatenate((c,b)))
[[ 4.  5.  6.]
 [ 7.  8.  9.]
 [ 0.  1.  2.]
 [ 3.  4.  5.]]
>>>print(np.concatenate((c, b), axis = 0))
[[ 4.  5.  6.]
 [ 7.  8.  9.]
 [ 0.  1.  2.]
 [ 3.  4.  5.]]
>>>print(np.concatenate((c, b), axis = 1))
[[ 4.  5.  6.  0.  1.  2.]
 [ 7.  8.  9.  3.  4.  5.]]
```

➔ **Pour concaténer un vecteur et une matrice, il est nécessaire de convertir le vecteur en matrice à 1 ligne (ou 1 colonne).**

np.newaxis **permet d'ajouter une dimension à un tableau (si b est un tableau unidimensionnel,** ***b[np.newaxis,:]*** **retournera une matrice à 1 ligne tandis que** ***b[:,np.newaxis]*** **retournera une matrice à 1 colonne).**

➔ **take() : alternative au slicing pour l'extraction**

```
>>>a = np.arange(16).reshape(4,4)
>>>print(a)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
>>>print(a.take([0,3], axis = 0))
[[ 0  1  2  3]
 [12 13 14 15]]
>>>print(a.take([0,3], axis = 1))
[[ 0  3]
 [ 4  7]
 [ 8 11]
 [12 15]]
>>>print(a.take([0,3,2,2], axis = 1))
[[ 0  3  2  2]
 [ 4  7  6  6]
 [ 8 11 10 10]
 [12 15 14 14]]
```

➔ **addition terme à terme de deux tableaux de même taille : +**

➔ **produit terme à terme de matrices de même taille : * (≠ produit matriciel !)**

```
>>>print(4 * np.ones((4,4)) *
            np.diag([1,2,3,4]))
[[  4.   0.   0.   0.]
 [  0.   8.   0.   0.]
 [  0.   0.  12.   0.]
 [  0.   0.   0.  16.]]
```

➔ **a**3 : matrice des cubes des coefficients de a (≠ élévation à la puissance 3!)**

➔ **1./a : matrice des inverses des coefficients de a (≠ inverse de a!)**

➔ **produit matriciel : np.dot(a, b)**
si v est un vecteur et a une matrice, np.dot(a, v) renvoie le produit matriciel a.v
np.dot(v, a) renvoie le produit matriciel transposee(v).a si les tailles sont compatibles.

➔ **produit scalaire de deux vecteurs v et w: np.vdot(v, w)**

➔ **np.outer(vecteur1, vecteur2)**

```
a = np.linspace(0,1,5)
b = np.linspace(10, 18, 3)

c = np.outer(a, b)

print('a = ', a)
print('b = ', b)
print('np.outer(a, b) = ', c)
-----------
a =  [ 0.    0.25  0.5   0.75  1.  ]
b =  [ 10.  14.  18.]
np.outer(a, b) =  [[ 0.    0.    0. ]
 [  2.5   3.5   4.5]
 [  5.    7.    9. ]
 [  7.5  10.5  13.5]
 [ 10.   14.   18. ]]
```

➔**produit de Kronecker de deux matrices a et b : np.kron(a, b)**

➔ **transposée de la matrice a : a.transpose()**

➔ **matrices booléennes de tests**

```
>>>a = 0.5 * np.ones((3,3))
>>>print(a)
[[ 0.5  0.5  0.5]
 [ 0.5  0.5  0.5]
 [ 0.5  0.5  0.5]]
>>>b = np.random.rand(3,3)
>>>print(b)
[[0.78543367  0.44928458  0.40077645]
 [0.04094524  0.32991854  0.71835905]
 [0.18977257  0.83001948
0.79173084]]
>>>print(a < b)
[[ True False False]
 [False False  True]
 [False  True  True]]
>>>print(b < 0.5)
[[False  True  True]
 [ True  True False]
 [ True False False]]
>>>print(b[b < 0.5])
[ 0.44928458  0.40077645  0.04094524
  0.32991854  0.18977257]
```

Appliquer une fonction aux coefficients d'une matrice

➔ **fonctions de numpy : np.fonction(matrice)**

```
>>>a = 10 * np.random.rand(3,3)
>>>b = np.cos(a)
>>>print(b)
[[ 0.11086734 -0.3377801  -0.42096733]
 [-0.37049461 -0.62513203 -0.38531808]
 [-0.72177202 -0.82130018 -0.74233932]]
```

➔ **fonctions personnelles (vectorialisées) : np.vectorize(ma_fonction)**

```
def mafonc(x):
    return np.exp(x) + x**2
mafoncV = np.vectorize(mafonc)
a = 10 * np.random.rand(3,3)
b = mafoncV(a)
```

Extrema, moyenne, …

```
>>>a = 10 * np.random.rand(2,3)
>>>print(a)
[[ 4.179929   2.6558287  5.96364962]
 [ 6.462623   8.4977509  8.33924147]]
```

➔ **minimum, maximum**

```
>>>print(np.amin(a))
2.65582870384
>>>print(np.argmin(a))
```

➔ **indice (comme si ligne)**

```
1
>>>print(np.amax(a))
8.49775092918
>>>print(np.argmax(a))
4
>>>print(np.amin(a, axis = 0))
```

➔**par col.**

```
[ 4.17992979  2.6558287   5.96364962]
>>>print(np.argmin(a, axis = 0))
[0 0 0]
>>>print(np.amin(a, axis = 1))
[ 2.6558287   6.46262388]
```

➔ **par ligne**

```
>>>print(np.argmin(a, axis = 1))
[1 0]
```

Somme, produit, moyenne

```
>>>print(np.sum(a))
36.0990243977
>>>print(np.sum(a, axis = 0))
[10.64255367 11.15357963 14.3028911 ]
>>>print(np.sum(a, axis = 1))
[ 12.79940812  23.29961628]
>>>print(np.prod(a))
30319.3994979
>>>print(np.prod(a, axis = 0))
[27.013314  22.568570  49.732314]
>>>print(np.prod(a, axis = 1))
[  66.20353314  457.97252895]
>>>print(np.mean(a))
6.01650406629
>>>print(np.mean(a, axis = 0))
[ 5.32127683  5.57678982  7.15144555]
>>>print(np.mean(a, axis = 1))
[ 4.26646937  7.76653876]
```

Rang d'une matrice a

```
>>>np.rank(a)
```

Transposée d'une matrice a

```
>>>a.transpose()
```

Inverse d'une matrice a

```
>>>np.linalg.inv(a)
```

Résolution d'un système linéaire a.X = b avec a et b matrices de tailles compatibles, a inversible

```
>>>np.linalg.solve(a, b)
```

Déterminant d'une matrice a

```
>>>np.linalg.det(a)
```

Valeurs propres, vecteurs propres d'une matrice

```
>>>a = np.array([[2,1,1 ],
[1,2,1],[1,1,2]])
>>>val_p, vect_p = np.linalg.eig(a)
>>>print(val_p)
[ 1.  4.  1.]
>>>print(vect_p)
[[-0.8164965  0.57735027 -0.32444284]
 [ 0.4082482  0.57735027 -0.48666426]
 [ 0.4082482  0.57735027  0.8111071]]
```

Norme euclidienne d'une matrice a

```
>>>np.linalg.norm(a)
```

Intégration

Calcul de $\int_{-10}^{0} \exp(x)\, dx$:

```
import numpy as np
import scipy as sp
import scipy.integrate as integ

valeur, err = integ.quad(np.exp, -10,0)
print('Valeur :', valeur)
print('Erreur :', err)

--------------------------
Valeur : 0.9999546000702375
Erreur : 2.8326146575791917e-14
```

Calcul de $\int_{-\infty}^{0} \exp(x)\, dx$:

```
valeur, err = integ.quad(np.exp,-np.inf,0)
print('Valeur :', valeur)
print('Erreur :', err)

--------------------------

Valeur : 1.0000000000000002
Erreur : 5.842607038578007e-11
```

Résolution de systèmes NON linéaires

➔Racines d'un polynôme donné par la liste de ses coefficients

```
import numpy as np
print(np.roots([1,3,3,1]))
-----------------------------------
[-0.99998950 +0.00000000e+00j
 -1.00000525 +9.09542286e-06j
 -1.00000525 -9.09542286e-06j]
```

(Mais -1 est racine triple de $1+3X+3X^2+X^3$!)

➔Zéro d'une fonction numérique

→ Méthode dichotomique

```
import scipy.optimize as spo

print(spo.bisect(np.sin, 3, 4))
---------
3.141592653589214
```

→ Méthode de Newton

(si la valeur de f' n'est pas donnée, la méthode de la sécante est appliquée ;

Si f' et f'' sont données, la méthode de Halley est appliquée)

```
print(spo.newton(np.sin, 3))
print(spo.newton(np.sin, 3, np.cos))
--------
3.14159265359
3.14159265359
```

➔Résolution du système $\begin{cases} x^2 = 1 \\ x - y = 2 \end{cases}$

```
import scipy.optimize as spo
z1 = spo.fsolve(lambda x :
      (x[0]**2 -1, x[0] - x[1] - 2),
       (1,1))
print(z1)
--------
[ 1. -1.]
```

```
z2 = spo.fsolve(lambda x :
      (x[0]**2 -1, x[0] - x[1] - 2),
      (-5,0))
print(z2)
--------
[-1. -3.]
```

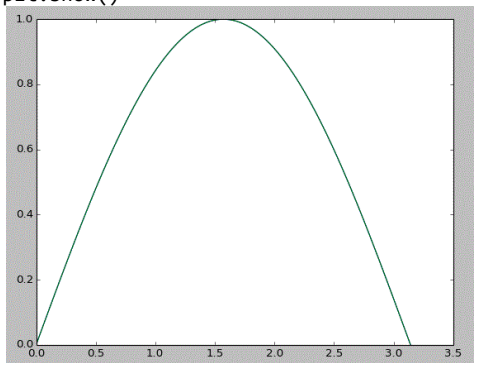
Équations et systèmes différentiels

odeint de **scipy.integrate** résout numériquement des systèmes différentiels (équations avec conditions initiales).

odeint(f, y0, t) résout y'(u) = f(y(u), u) avec y(a) = y0, t=[a, t_1, …, t_{n-1}, b]

Résolution de $y'(t) = y(t)^2 - y(t) * \sin(t) + \cos(t)$ avec y(0) = 0 :

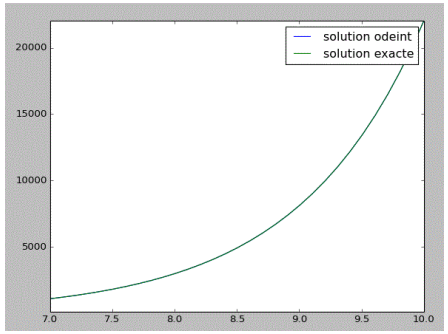
```
from scipy.integrate import odeint

def f(y, u):
    return y**2 - y*np.sin(u) + np.cos(u)

t = np.linspace(0,np.pi, 100)
y0 = 0
y = odeint(f,y0, t)

plt.plot(t, y)
plt.show()
```



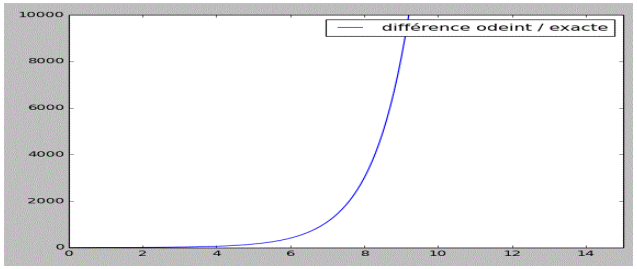
Résolution de $y'(t) = y(t)$ avec y(0) = 1 (solution exacte $y(t) = \exp(t)$):

t doit figurer dans la définition de la fonction f

```
from scipy.integrate import odeint

plt.axes( xlim=(7, 10),
          ylim=(2**7, np.exp(10)))

t = np.linspace(0,100,1000)
y = odeint(lambda y, t : y, 1, t)
z = np.exp(t)

plt.plot(t, y)
plt.plot(t, z)
plt.legend(['solution odeint',
            'solution exacte'])
plt.show()
```




```
plt.axes( xlim=(0, 15),
          ylim=(0, 10000))

t = np.linspace(0,100,1000)
y = odeint(lambda y, t : y, 1, t)
z = np.exp(t)

plt.plot(t, abs(z-y[0,:]))

plt.legend(['différence odeint / exacte'])
plt.show()
```

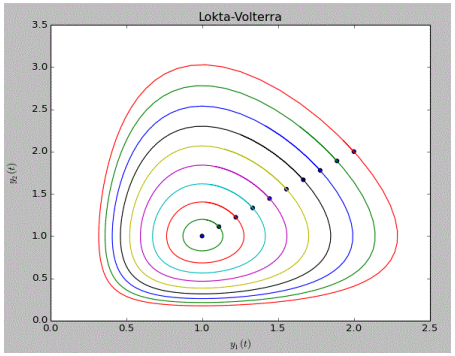


Résolution du système de Lokta-Volterra

$$\begin{cases} y1'(t) = y1(t) - y1(t) * y2(t) \\ y2'(t) = -2 * y2(t) + 2 * y1(t) * y2(t) \end{cases}$$

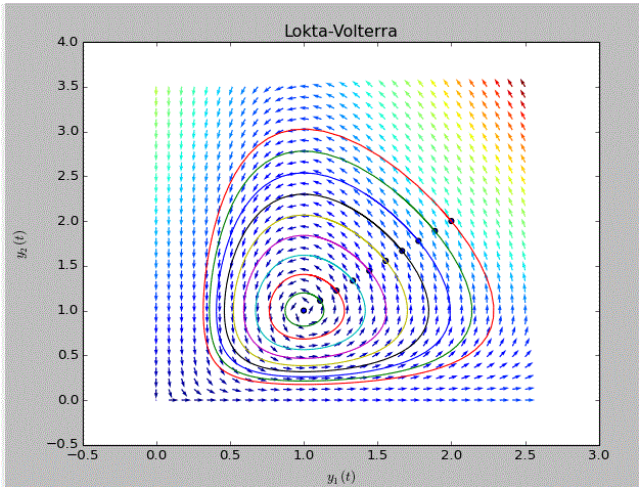
avec y1(0) = 2 et y2(0) = 2.

```
from scipy.integrate import odeint

def f(X, t):
    return (X[0]-X[0]*X[1], -2*X[1]+2* X[0]*X[1])

t = np.linspace(0, 5, 100)
condInit = np.linspace(1, 2, 10)

for y0 in condInit:
    Y = odeint(f, [y0, y0], t)
    plt.plot(Y[:, 0], Y[:, 1])

plt.scatter(condInit, condInit)
plt.xlabel('$y_1(t)$')
plt.ylabel('$y_2(t)$')
plt.title('Lokta-Volterra')

plt.show()
```



➔ champ de vecteurs

```
x = np.linspace(0, 2.5 ,30)
y = np.linspace(0, 3.5, 30)

X, Y = np.meshgrid(x,y)

dX, dY = f([X,Y], 0)

# les normes des vecteurs
normes = np.hypot(dX, dY)

# remplacer les normes nulles par 1
normes[normes == 0] = 1

# normalisation des vecteurs
dX /= normes
dY /= normes

# tracé du champ (option 5 : normes
# permet d'avoir un champ coloré en
# fonction de la valeur de la norme)

plt.quiver(X, Y, dX, dY, normes)
```



Gestion de fichiers

Exemple : gestion de fichier .csv :

```
# -*- coding: utf-8 -*-

# utilisation de fichier au format csv

import csv

pointVirgule = open('python_mail.csv', 'r')

lecturePointVirgule = csv.reader(pointVirgule,
                                 dialect = 'excel')

print('Contenu brut:')
print(lecturePointVirgule)

print('Liste : ')
listePV = list(lecturePointVirgule)
print(listePV)

print('Liste travaillée : ')
listeTravaillee = [x[0].split(';') for x in listePV]
print(listeTravaillee)

pointVirgule.close()
```

Contenu brut:
<_csv.reader object at 0x000000000618B9A0>
Liste :
[['Nom;Prénom;Ville;Courriel;Billes'],
['Katia;Barré;Vannes;katia.barre@laposte.net;2'],
['Pierre;Kiroule;Lyon;pierre.kiroule@gmail.com;4000'],
['Joe;La Pétanque;Poitiers;joe.la_Petanque@perso.solo;50']]
Liste travaillée :
[['Nom', 'Prénom', 'Ville', 'Courriel', 'Billes'], [
'Katia', 'Barré', 'Vannes', 'katia.barre@laposte.net', '2'],
['Pierre', 'Kiroule', 'Lyon', 'pierre.kiroule@gmail.com', '4000'],
['Joe', 'La Pétanque', 'Poitiers', 'joe.la_Petanque@perso.solo', '50']]