Fiche 6 - Représentations graphiques - équation différentielle

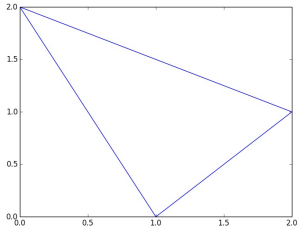
Un graphe est la ligne brisée reliant une suite de points définis par leurs coordonnées. Ainsi, il convient de définir deux vecteurs : un vecteur des abscisses, x, et un vecteur des ordonnées, y.

La fonction plot est dans le sous-module `pyplot` du module `matplotlib`.

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

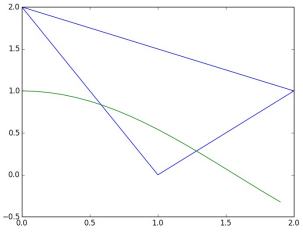
Exemple de triangle :

```
>>> plt.plot([0,1,2,0],[2,0,1,2])
```



La courbe de cosinus :

```
>>> x=np.arange(0,2,0.1)
>>> y=np.cos(x)
>>> plt.plot(x,y)
```



Attention ! Les graphes se superposent dans la fenêtre graphique ouverte. Utiliser `close()` pour fermer la fenêtre.

I. Quelques options et autres instructions

I.1. Vecteur de coordonnées

Il y a deux fonction dans la **module numpy** qui génèrent un tableau de nombres équi-répartis :

Notation	Explication
`linspace(a,b,n)`	n points équi-répartis sur [a,b]
`arange(a,b,pas)`	[a,a+pas,a+2pas , . . .] sur [a,b[

```
>>>np.linspace(2,3,5)
array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])
>>>np.arange(2,3.1,0.25)
array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])
```

Attention ! Les fonctions déjà programmées comme cos, exp, . . . peuvent s'appliquer à un nombre ou à un vecteur. Ce n'est pas le cas des fonctions que l'on peut définir soi-même.

⇒ Pour cela, il existe l'instruction `vectorize` du module `numpy`.

```
def f(t):
    if t>0.5:
        return t
    else:
        return 1-t
```

```
>>> x=np.linspace(0,1,5);x
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
>>> f(x)
ValueError: ... value of an array ... is ambiguous.
>>> h=np.vectorize(f)
>>> h(x)
array([ 1.  ,  0.75,  0.5 ,  0.75,  1.  ]
```

Une autre approche serait :

```
>>> [f(a) for a in x]
[1.0, 0.75, 0.5, 0.75, 1.0]
```

I.2. Les options de la fonction plot

Pour plus de détail, aller dans l'inspecteur d'objet.

➜ Les couleurs

Notation	Explication
'b'	blue
'g'	green
'r'	red
'm'	magenta
'k'	black
'y'	yellow

➜ Les styles de lignes

Notation	Explication
'-'	ligne pleine
'--'	ligne découpée
'-.'	ligne tiret-point
':'	ligne de pointillés

Lycée Saint-Exupery – maffre.saintexupery@gmail.com – MPSI

➜ Les styles de marques

Notation	Explication
'.'	point
'o'	cercle
','	pixel
's'	carré
'D'	diamant
'+'	plus

➜ Autres options

Notation	Explication
label='...'	associe une légende à la courbe
color='blue'	couleur
linewidth=3	épaisseur du trait

I.3. L'environnement d'un graphe

Notation	Explication
close()	ferme les figures précédentes ; par défaut, les graphes successifs se superposent.
show()	ouvre une fenêtre pour afficher une image
figure('nom')	donne un nom à la fenêtre
title('nom')	donne un titre à la figure
savefig('nom.pdf')	enregistre la figure dans un fichier image (.pdf, .png, ...)
xlabel('...')	étiquette sur l'axe
xlim(a,b)	bornes de la fenêtre axis([xmin,xmax,ymin,ymax])
xscale('log')	échelle des graduations : 'linear'
grid(True)	ajoute une grille
legend()	affiche une légende
axis()	nature du repère 'scaled' : orthonormé 'off' : sans
text(a,b,'message')	ajout de texte au point (a,b)
arrow(x,y,dx,dy)	flèche

Remarque : Précisions sur l'instruction `text`

➜ Les options :

Notation	Explication
fontsize=12	taille du texte
color='red'	couleur
ha='left'	alignement horizontal : point à gauche 'right' ou 'center'
va='bottom'	alignement vertical : point au dessous 'center' ou 'top'

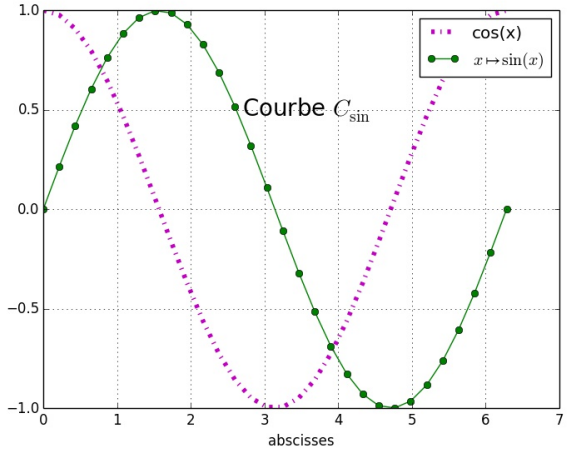
➜ Le texte peut contenir des expressions mathématiques en utilisant des commandes Tex entre $:

Notation	Expression
M^{12}_{i,j}	$M^{12}_{i,j}$
\mapsto	$\mapsto$
\rightarrow	$\rightarrow$
\cos	$\cos$

➜ Exemple avec légende :

plot1.py

```
plt.close()
x=np.linspace(0,2*pi,30)
y1=np.cos(x)
y2=np.sin(x)
plt.plot(x, y1, 'm-.', label='cos(x)', linewidth=4)
plt.plot(x, y2, 'g-o', label='$x\mapsto \sin (x)$')
plt.text(2.7,0.5,'Courbe $C_\sin$ ',ha='left',va='cente
plt.xlabel("abscisses")
plt.grid(True)
plt.legend()
plt.show()
```



II. Figures multiples

Cet instruction permet de créer un tableau de figures : elle positionne le graphe numéroté "nb" dans un tableau de "n" lignes et "p" colonne :

```
plt.subplot(n,p,nb)
```

➜ Exemple

plot2.py

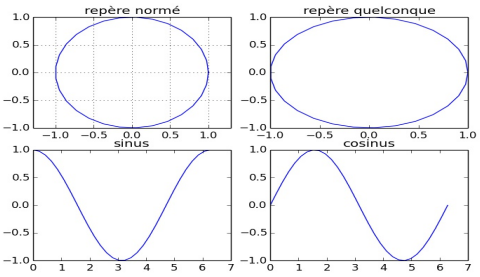
```
plt.close()
t=np.linspace(0,2*pi,30)
x=np.cos(t)
y=np.sin(t)

plt.subplot(2,2,1)
plt.plot(x, y)
plt.axis('equal')
plt.title('repere norme')
plt.grid(True)

plt.subplot(2,2,2)
plt.plot(x, y)
plt.title('repere quelconque')
```

```
plt.subplot(2,2,3)
plt.plot(t,x)
plt.title('sinus')

plt.subplot(2,2,4)
plt.plot(t,y)
plt.title('cosinus')
```



III. Equation différentielle

III.1. L'instruction odeint()

La fonction odeint() calcule une valeur approchée d'une solution au problème de Cauchy pour un système d'équations différentielles.

Cette fonction est dans le module scipy :

```
import scipy.integrate as spi
```

Considérons le problème :

$$y' = f(y,t)$$
$$y(t_0) = y_0$$

L'instruction y=spi.odeint(f,y0,t) calcule la valeur de la solution approchée en les valeurs de t où t est une liste de points commençant par t_0.

La variable de sortie y est un tableau de taille $n \times p$ où la k-ième colonne contient la k-ième composante de y :

```
y[i,k]= yk(t[i])
```

Equation d'ordre 1

Considérons l'équation $y' = y$ de condition initiale $y(0) = 1$. Cela nous donne une approximation de exp(1).

exp(1) voir exp1.py

```
def f(y,t):
    return y
y0=1
t=np.linspace(0,1,2)
y=spi.odeint(f,y0,t)
print('valeur par defaut :',e)
print('valeur approximee :',y[1,0])
```

On trouve :

```
valeur par defaut : 2.718281828459045
valeur approxime : 2.71828193307
```

Tracé de la courbe de la solution approchée :

Tracé de exp voir courbe_exp.py

```
def f(y,t):
    return y
y0=1
t=np.linspace(0,3,20)
y=spi.odeint(f,y0,t)
plt.plot(t,y[:,0],'ro')
plt.plot(t,exp(t),'b-')
```

Système d'équations d'ordre 1

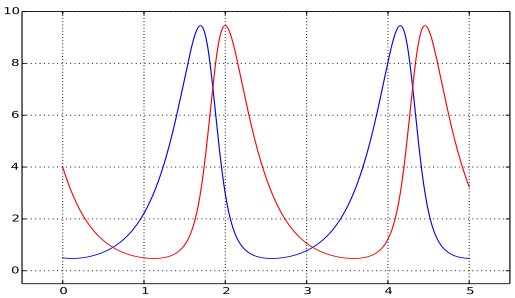
Considérons le système de Lotka-Volterra :

$$x'(t) = ax(t) - bx(t)y(t)$$
$$y'(t) = -cy(t) + dx(t)y(t)$$

Le tracé en fonction du temps est : les proies en bleu et les prédateurs en rouge.

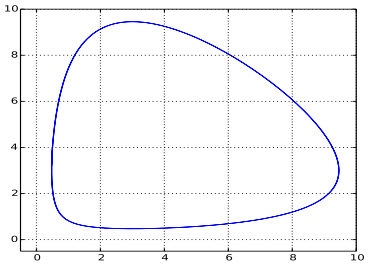
Système de Lotka-Volterra

```
a,b,c,d=3,1,3,1
def f(y,t):
   return array([a*y[0]-b*y[0]*y[1],
                 -c*y[1]+d*y[0]*y[1]])
"""Graphes de t->x(t) et t->y(t)"""
plt.figure("Equation de Lotka Volterra")
plt.grid(True)
plt.xlim(-0.5,5.5)
plt.ylim(-0.5,10)
y0=np.array([0.5,4])
t=np.linspace(0,5,10**3)
y=spi.odeint(f,y0,t)
plt.plot(t,y[:,0],color='blue')
plt.plot(t,y[:,1],color='red')
plt.savefig('lotka_volterra_temporel.pdf')
```



Le tracé des prédateurs en fonction des proies.

```
"""Courbe t->(x(t),y(t))"""
plt.axis('scaled')
plt.grid(True)
plt.xlim(-0.5,10)
plt.ylim(-0.5,10)
plt.plot(y[:,0],y[:,1],color='blue')
plt.savefig('lotka_volterra.pdf')
```



Equation d'ordre 2

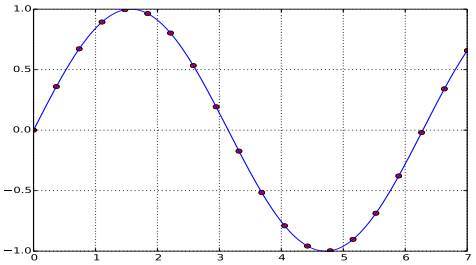
Etudions le cas de la fonction sin solution de $x'' + x = 0$ avec pour conditions initiales $(x(0) = 0, x'(0) = 1)$.
Il convient de réécrire cette équation différentielle d'ordre 2 en un système d'équations différentielles d'ordre 1 :

$$x'(t) = y(x)$$
$$y'(t) = -x(t)$$

On trace en pointillés rouges la courbe de sin et en trait bleu celle de la solution approchée au problème de Cauchy :

Fonction *sinus*

```
def f(y,t):
    return array([y[1],-y[0]])
plt.close()
plt.figure("Equation differentielle x''+x=0")
"""Graphe de sin"""
u=np.linspace(0,7,20)
plt.plot(u,sin(u),'ro')
t=np.linspace(0,7,10**3)
y0=np.array([0,1])
y=spi.odeint(f,y0,t)
plt.plot(t,y[:,0],'b-')
plt.grid(True)
plt.show()
plt.savefig('sin.pdf')
```



III.2. Champ de vecteurs : quiver()

Le **module `matplotlib.pyplot`** propose deux fonctions :

- `quiver(X,Y,U,V)` affiche un champ de vecteur.
- `streamplot(X,Y,U,V)` affiche les lignes de champ.

Les quatre variables sont des tableaux de même taille où `X` contient les abscisses des point, `Y` contient les ordonnées des point, `U` contient les abscisses des vecteurs et `V` contient les ordonnées des vecteurs.

➜ l'instruction `meshgrid` du module `numpy` facilite la construction de `X` et `Y` en donnant seulement les listes `x` et `y` considérées :

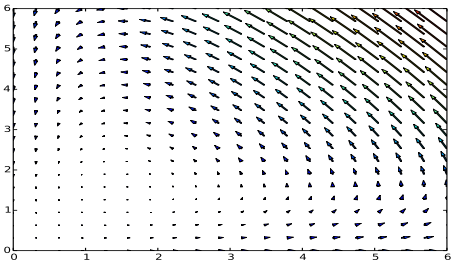
```
>>> x=[0,1];y=[2,3,4];
>>> X,Y=np.meshgrid(x,y)
>>> X
array([[0, 1],
       [0, 1],
       [0, 1]])
>>> Y
array([[2, 2],
       [3, 3],
       [4, 4]])
```

Ainsi, il ne reste plus qu'à appliquer la fonction f.
➜ Exemple du système de Lotka-Volterra :

Champ de vecteurs

```
a,b,c,d=3,2,3,2
x,y=np.linspace(0,6,20),np.linspace(0,6,20)
X,Y=np.meshgrid(x,y)
U,V=a*X-b*X*Y,-c*Y+d*X*Y
N=np.sqrt(U*U+V*V)
plt.quiver(X,Y,U,V,N)
```

Remarque : Le 5ième argument est optionnel, c'est un tableau fixant la color de chaque vecteur, il est de même taille que les autres arguments.



Lignes de champ

```
a,b,c,d=3,2,3,2
x,y=np.linspace(0,6,20),np.linspace(0,6,20)
X,Y=np.meshgrid(x,y)
U,V=a*X-b*X*Y,-c*Y+d*X*Y
N=np.sqrt(U*U+V*V)
plt.streamplot(X,Y,U,V,color=N)
```