Représentations graphiques sous Python

Pour représenter des suites ou des fonctions nous utiliserons le module graphique `matplotlib` de Python. Il peut être chargé directement, mais pour plus de commodité, nous chargerons la bibliothèque `pylab` qui le contient ainsi que de nombreuses autres fonctions mathématiques.

1 Prise en main

1) Tracer une suite

La commande **plot**(`x,y`) prend en argument deux listes de même longueur et trace l'ensemble des points de coordonnées $(x[i], y[i])_{i\in[\![0,\mathtt{len(x)-1}]\!]}$.

En particulier, pour ne tracer qu'un seul point, on pourra écrire **plot**(`x,y,'o'`), avec x, y deux variables représentant des flottants[1].

Remarques :

- Le graphique apparaît normalement **dans une autre fenêtre** en arrière plan. S'il ne s'affiche pas, utiliser la commande `show()`.
- Par défaut la commande relie les points par un trait, cela peut être modifié.

```
from pylab import *
x = [n+1 for n in range(100)]
y = [1/i for i in x]
plot(x,y)
plot(range(100),y)         # idem
plot(range(100),y,'ro')    # rouge avec points ronds
plot(range(100),y,'ko-')   # noir avec points ronds reliés par des traits.
```

(a) Tracer la suite $(\cos(n))_{n\in\mathbb{N}}$, sans relier les points. Tracer les termes de la suite un à un, sans les sauvegarder dans une liste.

(b) Émettre une conjecture sur la nature de la suite $(\cos n)_{n\in\mathbb{N}}$.
Ne pas hésiter à augmenter le nombre de points pour admirer la vue ! (avec 100, 1000 *puis* 5000*). On pourra utiliser* `plot(x,y,'o')`.
La (bonne) conjecture peut être démontrée en utilisant un raisonnement par l'absurde.

(c) Tracer avec Python les n premiers termes de la suite définie par récurrence par $u_0 = 1$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \frac{u_n}{n+1}$.

2) Tracer une fonction

La commande **linspace**(`0,10,100`) crée un tableau de 100 points uniformément répartis entre 0 et 10.

(a) Tracer la courbe $x \mapsto \cos x$ sur $[0, 2\pi]$ avec 200 points.

(b) Quelques commandes supplémentaires en exemple (pas à connaître par cœur) :

```
x = linspace(-5,5,200)
y1 = exp(x)
y2 = x+1
plot(x, y1, color='blue', label='exp(x)')
plot(x, y2, color='red', label='x+1')
xlabel("abscisses")
ylabel("ordonnees")
legend()
xlim(-5, 5)
ylim(-2, 10)
```

Pour davantage de commandes, se reporter à l'aide en tapant `help(plot)`.

3) Tracer une réciproque

Sans calculs, tracer la courbe de l'application réciproque de $x \mapsto (1+x^2)^x$.

[1] On n'a pas besoin de créer une liste à un élément.

2 EXERCICES D'APPROFONDISSEMENT

EXERCICE 1

1) Fixer une variable globale `xmax`.
Programmer une fonction python **ordre1**(`a,b,x0,y0`) qui trace sur [`x0`,`xmax`] la solution du problème de Cauchy

$$y' = ay + b, \qquad \text{avec } y(\texttt{x0}) = \texttt{y0}$$

2) Tracer une fonction **champVecteurs**(`a,b,xmin,xmax,ymin,ymax`) qui trace le champ de vecteurs associé à l'équation différentielle $y' = ay + b$ dans la fenêtre définie par $\{(x,y)) \in \mathbb{R}^2,\ \texttt{xmin} \leq x \leq \texttt{xmax},\ \texttt{ymin} \leq y \leq \texttt{ymax}\}$

3) Programmer une fonction **ordre2**(`omega,sigma,y0,y0prime`) qui trace sur [0,`xmax`], la solution du problème de Cauchy

$$y'' + 2\sigma\omega y' + \omega^2 y = 0, \qquad \text{avec } y(0) = \texttt{y0} \text{ et } y'(0) = \texttt{y0prime}$$

4) Tracer une fonction **champVecteursF**(`f,xmin,xmax,ymin,ymax`) qui trace le champ de vecteurs associé à l'équation différentielle $y' = f(x,y)$ dans la fenêtre définie par $\{(x,y)) \in \mathbb{R}^2,\ \texttt{xmin} \leq x \leq \texttt{xmax},\ \texttt{ymin} \leq y \leq \texttt{ymax}\}$
Essayer avec $y' = y^2 - x$.

5) En déduire le tracé d'une solution approchée de l'équation différentielle $y' = f(x,y)$.
Attention : le résultat devient vite très grand !
Superposer avec le champ de vecteurs.

EXERCICE 2 (Pour ceux qui ont terminé)

Adapter l'exercice précédent pour donner le champ de vecteurs associé au système différentiel :

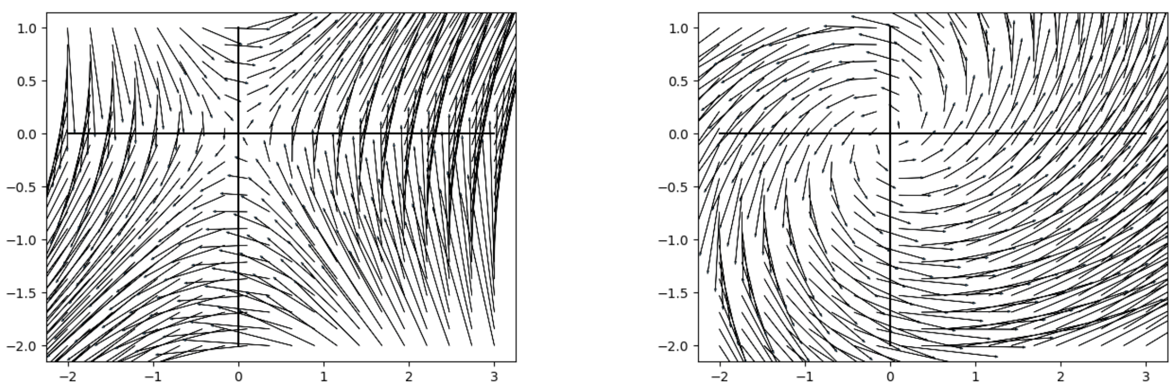
$$\begin{cases} x'(t) &= ax + by \\ y'(t) &= cx + dy \end{cases}$$

(le champ de vecteur correspond aux vecteurs (x', y') pour chaque position (x,y))
On verra des comportements très différents selon les valeurs de a, b, c, d.

1) Essayer avec $(a,b,c,d) = (0.86, 2.03, 1.97, 0.28)$

2) Essayer avec $(a,b,c,d) = (0.86, -2.1, 1.28, 3.10)$

3) Essayer avec $(a,b,c,d) = (3.42, 2.33, -2.06, 0.69)$



Pour les premières valeurs de (a,b,c,d)

Remarque : On pourra utiliser la commande **arrow**(`x,y,dx,dy,**kwargs`) qui trace une flèche entre `(x,y)` et `(x+dx,y+dy)`.