REPRÉSENTATION GRAPHIQUE

B. Landelle

Table des matières

I Fonctions et tableaux **2**
1 Fonctions 2
2 Tableaux, listes par compréhension 3

II Représentation graphique 2D **3**
1 Graphe d'une fonction 4
2 Graphe d'une suite 8
3 Graphe d'une courbe paramétrée 9
4 Graphe d'une courbe implicite 11
5 Tracé d'un champ de vecteurs 12

III Représentation graphique 3D **13**
1 Tracé d'une courbe paramétrée dans l'espace 13
2 Tracé d'une surface d'équation $z = f(x, y)$ 14
3 Tracé d'une surface paramétrée 15

I Fonctions et tableaux

1 Fonctions

Pour les simulations numériques, on importera le module `numpy` sous l'alias `np`.

```
import numpy as np
```

Pour définir une fonction, on peut utiliser les fonctions anonymes aussi appelées fonctions `lambda` pour les définitions simples ou la syntaxe standard avec `def`, plus générale et qui permet d'écrire des fonctions plus élaborées.

```
f=lambda x:np.sqrt(1-x**2)

def f(x):
    return np.sqrt(1-x**2)
```

Quelques fonctions mathématiques importantes :

`abs`	$x \mapsto \lvert x \rvert$
`np.log`	$x \mapsto \ln(x)$
`np.sqrt`	$x \mapsto \sqrt{x}$
`np.floor`	$x \mapsto \lfloor x \rfloor$

Exemples : 1. Une fonction $f : \mathbb{R} \to \mathbb{R}$ se décompose de manière unique en $f = \check{f} + \widehat{f}$ avec $\check{f}$ paire et $\widehat{f}$ impaire. On peut coder $\check{f} : x \in \mathbb{R} \mapsto \dfrac{f(x)+f(-x)}{2}$ ainsi :

```
def paire(f,x):
    return (f(x)+f(-x))/2

paire=lambda f,x:(f(x)+f(-x))/2
```

On remarque qu'une fonction peut être argument d'une autre fonction.

2. Codage d'une fonction `h(x,n)` d'argument un réel x et un entier n et qui renvoie $h_n(x)$ avec

$$\forall (x,n) \in \mathbb{R} \times \mathbb{N} \qquad h_n(x) = \prod_{k=1}^{n} (1 + x^k)$$

On saisit :

```
def h(x,n):
    p=1
    for k in range(n):
         p*=1+x**(k+1)
    return p
```

2 Tableaux, listes par compréhension

Un tableau, objet de type `ndarray`, est une suite de variables de type flottant associées à des emplacements consécutifs de la mémoire. Ainsi, les éléments du tableau sont accessibles en lecture et écriture en temps constant.

Pour générer des tableaux de valeurs régulièrement espacées, on utilise les instructions `np.linspace` et `np.arange`. L'instruction `np.linspace(a,b,n)` construit un tableau de n valeurs régulièrement espacées de a à b, bornes incluses.

```
>>> a=np.linspace(0,1,4)
>>> a
array([ 0.        ,  0.33333333,  0.66666667,  1.        ])
```

L'instruction `np.arange(a,b,h)` construit une liste de valeurs régulièrement espacées de h de a à b exclus.

```
>>> a=np.arange(0,1,.2)
>>> a
array([ 0. ,  0.2,  0.4,  0.6,  0.8])
```

Avec certaines fonctions *vectorisées*, on peut directement calculer le tableau image d'un tableau par une fonction.

```
>>> tx=np.linspace(0,np.pi,5)
>>> np.sin(tx)
array([  0.00000000e+00,   7.07106781e-01,   1.00000000e+00,
         7.07106781e-01,   1.22464680e-16])
```

On peut aussi utiliser des listes par compréhension :

```
>>> tx=np.linspace(0,np.pi,5)
>>> tf=[np.sin(x) for x in tx]
>>> tf
[0.0, 0.70710678118654746, 1.0, 0.70710678118654757,
1.2246467991473532e-16]
```

Cette syntaxe très simple est à privilégier car elle fonctionne pour toutes les fonctions, y compris celles non vectorisées.

II Représentation graphique 2D

Pour les représentations graphiques dans le plan et l'utilisation de fonctions mathématiques, on importera le module `matplotlib.pyplot` sous l'alias `plt` et le module `numpy` sous l'alias `np`.

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

L'instruction essentielle dans cette partie est la fonction `plt.plot` qu'on utilise selon la syntaxe suivante :

```
plt.plot(tx,ty)
```

où `tx` et `ty` sont des listes ou tableaux d'abscisses et d'ordonnées.

1 Graphe d'une fonction

Pour tracer le graphe de sin sur $[-4\,;4]$, on saisit :

```
tx=np.linspace(-4,4,100)
ty=[np.sin(x) for x in tx]
plt.plot(tx,ty)
plt.grid();plt.show()    # quadrillage puis affichage
```

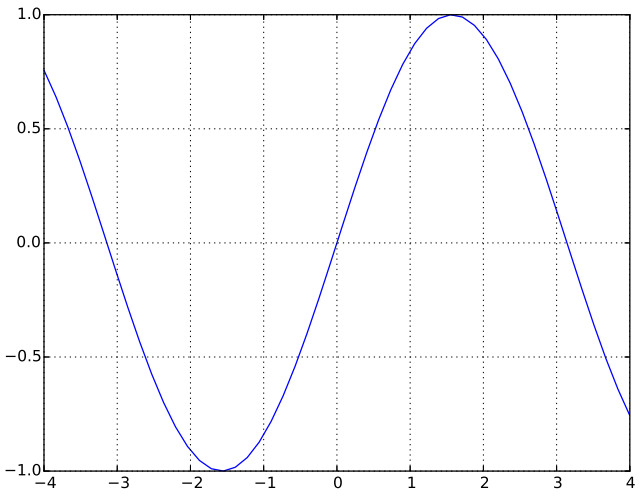

FIGURE 1 – Tracé de $y = \sin(x)$

Remarque : Pour créer le tableau `ty`, on peut aussi procéder vectoriellement avec `ty=np.sin(tx)`. Cette syntaxe est très confortable quand elle fonctionne mais elle suppose que la fonction soit *vectorisée*. Si ce n'est pas le cas, il faut recourir à `np.vectorize`. La construction par liste par compréhension proposée ci-avant fonctionne systématiquement.

Si on veut donner une légende au graphe, on saisit l'argument `label` dans l'appel à `plt.plot` puis l'affichage de cette légende avec `plt.legend()`.

```
plt.plot(tx,ty,label='y=sin(x)');
plt.legend()
plt.grid();plt.show()
```

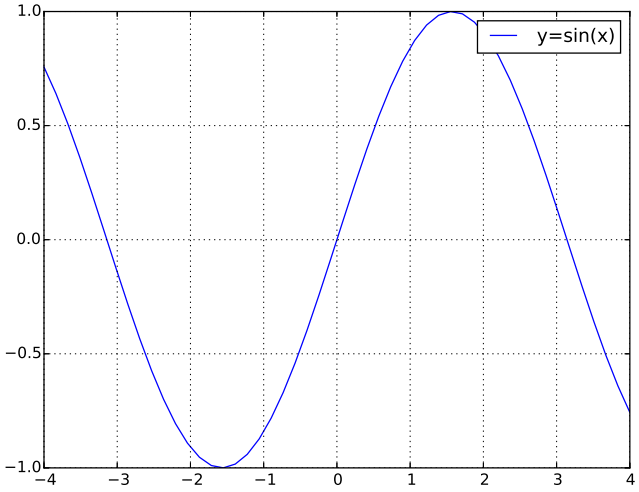

FIGURE 2 – Tracé avec légende

Pour nommer les axes, on utilise les instructions `plt.xlabel` pour l'axe des abscisses et `plt.ylabel` pour l'axe des ordonnées.

```
plt.plot(tx,ty);
plt.xlabel("temps (s)")
plt.ylabel("distance (m)")
plt.grid();plt.show()
```

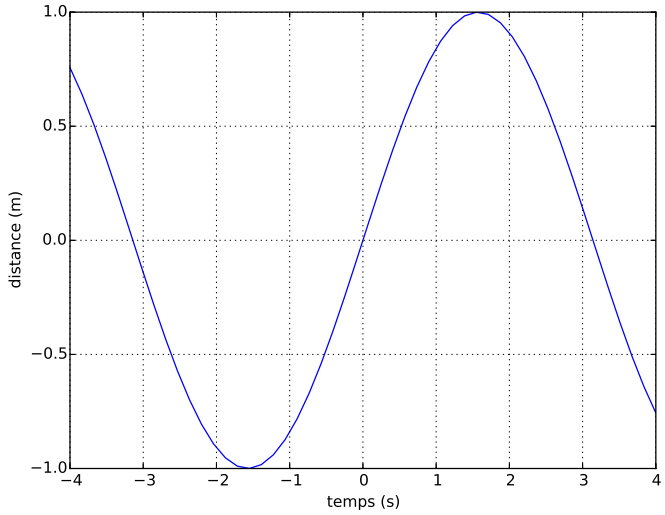

FIGURE 3 – Axes labellisés

Le fenêtrage est réalisé automatiquement par python mais on peut aussi fixer les coordonnées minimales et maximales avec la commande `plt.axis([xmin,xmax,ymin,ymax]`. Les instructions `plt.xlim(xmin,xmax)` et `plt.ylim(ymin,ymax)` permettent de fixer séparément les valeurs minimales et maximales des abscisses ou des ordonnées.

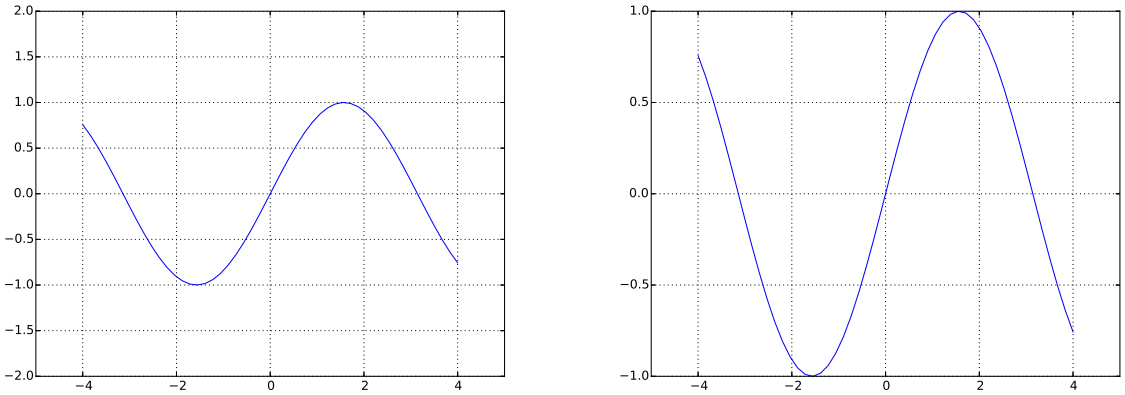

FIGURE 4 – Fenêtrage paramétré

```
...
plt.plot(tx,ty)
plt.axis([-5,5,-2,2])     # fenêtre xmin=-5,xmax=5, ymin=-2, ymax=2
plt.grid();plt.show()

plt.plot(tx,ty)
plt.xlim(-5,5)            # fenêtre xmin=-5, xmax=5
plt.grid();plt.show()
```

Des options de `plt.plot` permettent de personnaliser certains tracés :

- `color='red','blue','cyan', ...,`
- `marker='s','+','*','.','1','2',...,`
- `linestyle='-','--','-.',':',...`

```
...
tx=np.linspace(-4,4,50)
ty=[ np.sin(x) for x in tx]
plt.plot(tx,ty,color='red',marker='s',linestyle='-.')
# tracé en rouge, points repérés par des carrés, reliés par des pointillés
plt.grid();plt.show()

plt.plot(tx,ty,'rs-.')   # syntaxe courte pour le même graphique
plt.grid();plt.show()
```

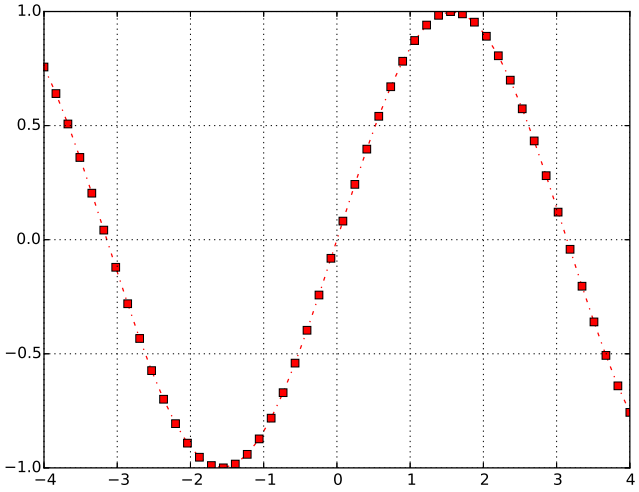

FIGURE 5 – Tracé de $y = \sin(x)$

On peut tracer plusieurs courbes sur un même graphique, soit en les plaçant à la suite comme arguments d'un même appel à `plt.plot`, soit lors de plusieurs appels de `plt.plot` avant de produire l'affichage avec `plt.show()`.

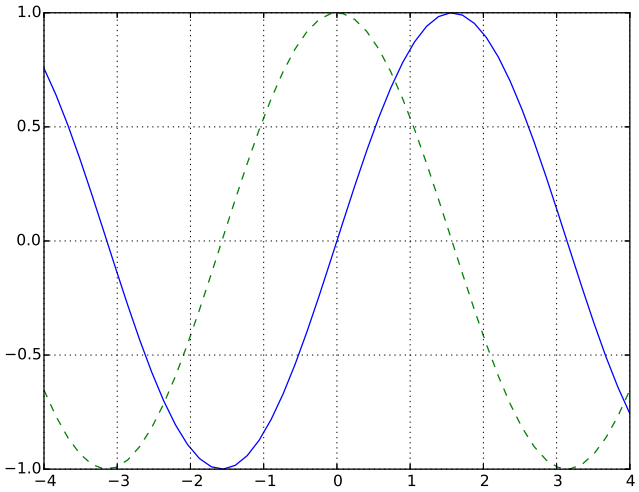

FIGURE 6 – Tracé de $y = \sin(x)$ et $y = \cos(x)$

```
tz=[np.cos(x) for x in tx]
plt.plot(tx,ty)           # tracé des points (x,y)
plt.plot(tx,tz,'--')      # tracé des points (x,z)
plt.grid();plt.show()     # affichage des tracés sur un même graphique
```

ou avec

```
plt.plot(tx,ty,tx,tz,'--')  # tracé des points (x,y) et (x,z)
plt.grid();plt.show()   # affichage des tracés sur un même graphique
```

Si on veut par exemple des légendes pour chaque graphe et le tracé plein plus épais que le tracé en pointillé, on saisit :

```
plt.plot(tx,ty,label='y=sin(x)',linewidth=2)
plt.plot(tx,tz,'--',label='y=cos(x)')
plt.legend()
plt.grid();plt.show()
```

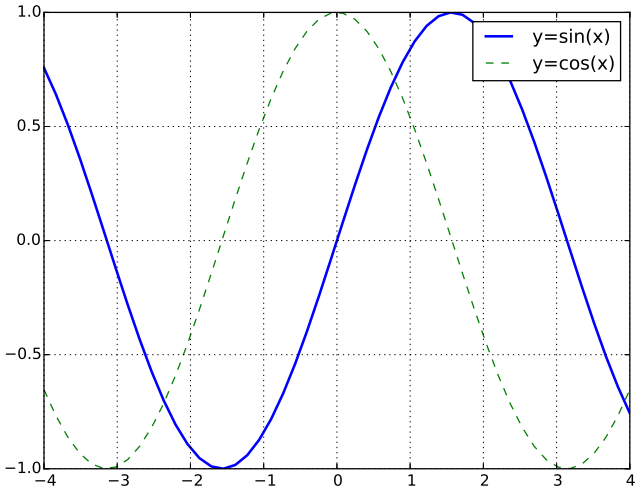

FIGURE 7 – Tracé de $y = \sin(x)$ et $y = \cos(x)$

2 Graphe d'une suite

Soit $(u_n)_n$ la suite définie par

$$\forall n \in \mathbb{N} \qquad u_n = \sin(\pi\sqrt{1+n^2})$$

On trace les 20 premiers termes de la suite avec :

```
f=lambda n:np.sin(np.pi*np.sqrt(1+n**2))
tn=range(20)
tu=[f(n) for n in tn]
plt.plot(tn,tu,'bo')   # ou aussi : plt.scatter(tn,tu)
plt.grid();plt.show()
```

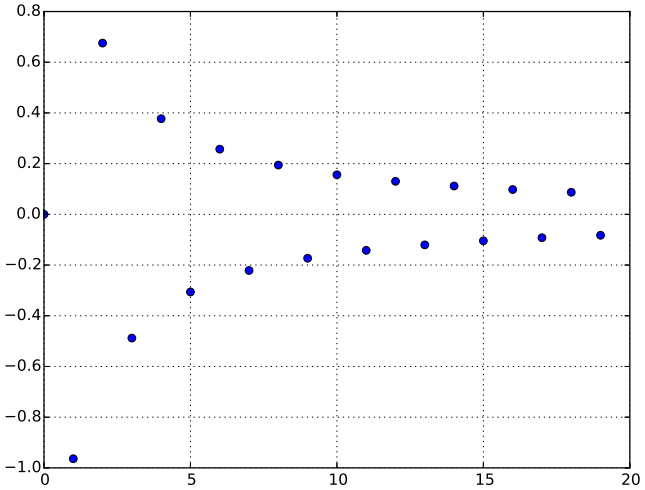

FIGURE 8 – Tracé de la suite $(u_n)_n$

Les options `'bo'` signifient `color='blue'` et `'marker='o'` ce qui produit des points bleus non reliés.

Pour relier les points par des pointillés :

```
plt.plot(tn,tu,'bo--')
```

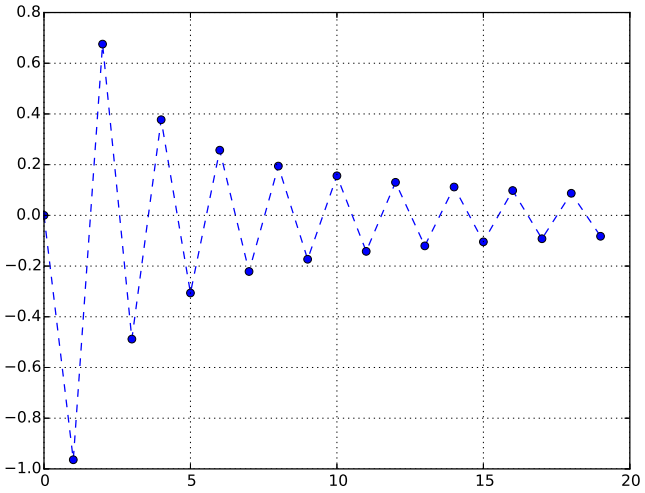

FIGURE 9 – Tracé de la suite $(u_n)_n$

3 Graphe d'une courbe paramétrée

Une *courbe paramétrée* de paramétrage l'application $\mathrm{I} \to \mathbb{R}^2, t \mapsto (x(t), y(t))$ avec I un intervalle de $\mathbb{R}$ est l'ensemble des points

$$\{(x(t), y(t)),\ t \in \mathrm{I}\}$$

```
tt=np.linspace(0,2*np.pi,100)
tx=[np.cos(t) for t in tt]
ty=[np.sin(t) for t in tt]
plt.plot(tx,ty)                 # tracé de t->(x(t),y(t))
plt.grid();plt.show()
```

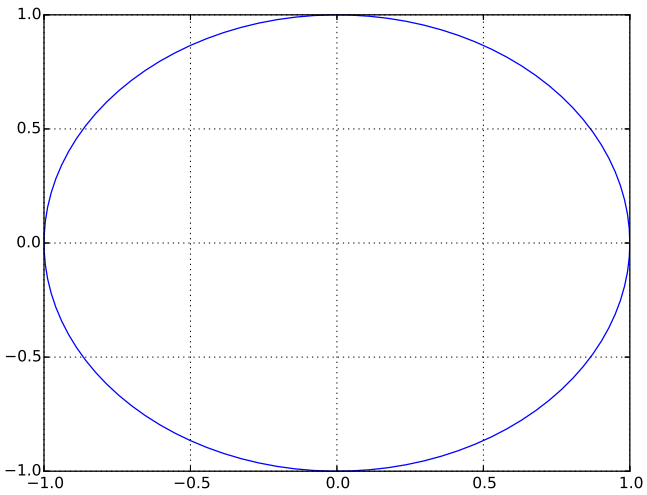

FIGURE 10 – Tracé de courbe paramétrée par $t \mapsto (\cos(t), \sin(t))$

Sur la figure précédente, les proportions ne sont pas respectées puisqu'on est censé observer le cercle unité et pourtant, le dessin ressemble davantage à celui d'une ellipse qu'à celui d'un cercle. Pour conserver les proportions, on utilise l'option `plt.axis('equal')` :

```
...
plt.axis('equal');plt.show()
```

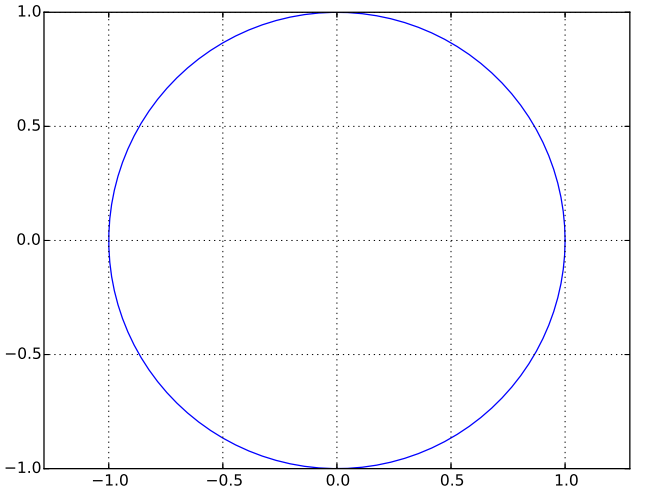

FIGURE 11 – Tracé de courbe paramétrée par $t \mapsto (\cos(t), \sin(t))$

Selon le même principe que pour une courbe paramétrée, on peut représenter un nuage de points. Pour observer 1000 points dont les coordonnées suivent des lois normales centrées réduites, on saisit :

```
import numpy.random as rd
n=1000
tx=rd.normal(0,1,n);ty=rd.normal(0,1,n)
plt.plot(tx,ty,'bo');plt.grid();plt.show()
```

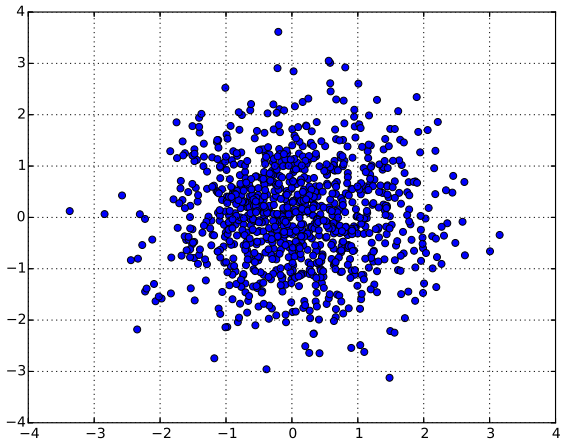

FIGURE 12 – Tracé d'un nuage de points

4 Graphe d'une courbe implicite

Étant donnée une fonction $f : \mathrm{D} \to \mathbb{R}$ avec $\mathrm{D} \subset \mathbb{R}^2$, la *courbe implicite* d'équation $f(x, y) = 0$ est l'ensemble des points

$$\{(x, y) \in \mathrm{D} \quad | \quad f(x, y) = 0\}$$

En sciences physiques, le tracé de lignes de champ ou d'équipotentielles est une situation typique de représentation de courbes implicites d'équations de la forme

$$f(x, y) = \lambda \quad \text{avec} \quad \lambda \text{ un paramètre}$$

L'équation peut s'écrire $f(x, y) - \lambda = 0$ ce qui équivaut donc à la situation générique d'une courbe implicite. L'importation du module `pylab` est nécessaire. La fonction f doit être vectorisée.

```
import pylab as pl

def f(x,y):
    return ((x-1)**2+y**2)*((x+1)**2+y**2)

x=np.linspace(-2,2,100)          # tableau des valeurs de x
y=np.linspace(-2,2,100)          # tableau des valeurs de y
X,Y=np.meshgrid(x,y)             # grille des points (x,y)
Z=f(X,Y)
```

```
levels=[.1,.2,.5,.6,.9,1,2,3]     # différentes valeurs de paramètre
pl.contour(X,Y,Z,levels)          # tracé des courbes implicites
plt.show()
```

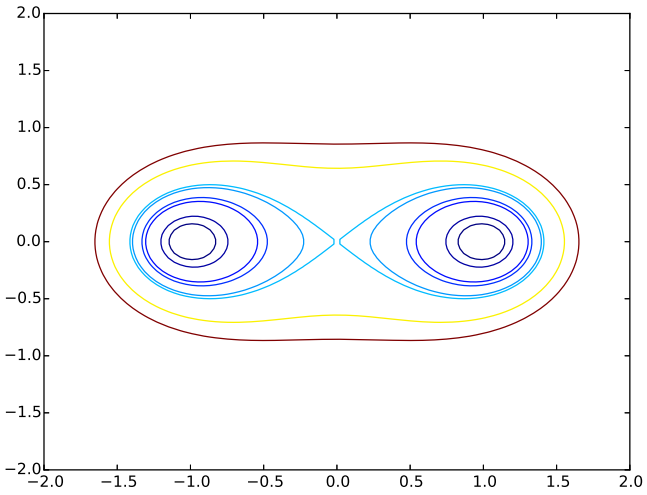

FIGURE 13 – Tracé de courbes implicites

5 Tracé d'un champ de vecteurs

Étant donnée une fonction $f : \mathrm{D} \to \mathbb{R}^2$ avec $\mathrm{D} \subset \mathbb{R}^2$, le tracé du *champ de vecteurs* associé à f consiste à représenter l'ensemble des vecteurs suivants

$$\{f(x,y),\ (x,y) \in \mathrm{D}\}$$

Les éléments de cet ensemble sont des vecteurs de $\mathbb{R}^2$ et sont représentés par des flèches dont le sens, la direction et la longueur correspondent au caractéristique du vecteur en question. La fonction f doit être vectorisée.

```
def f(x,y):
    a=1;b=2;c=1;d=3;
    return a*x*y-b*x,-c*x*y+d*y

tx=np.linspace(0,10,20)     # tableau des valeurs de x
ty=np.linspace(0,10,20)     # tableau des valeurs de y
X,Y=np.meshgrid(tx,ty)      # grille des points (x,y)
U,V=f(X,Y)
plt.quiver(X,Y,U,V)         # Tracé du champ de vecteurs
plt.show()
```

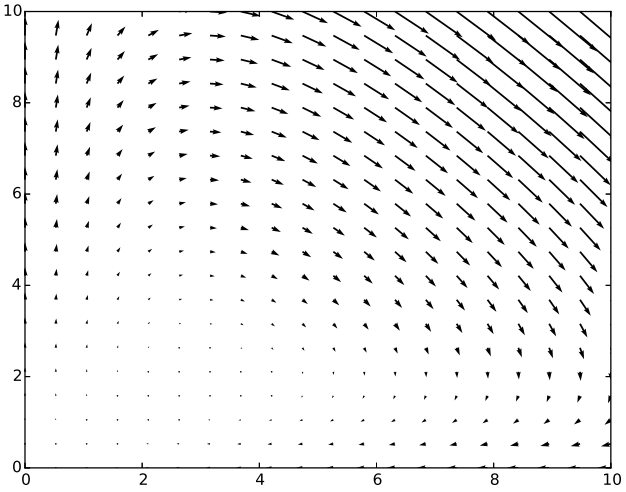

FIGURE 14 – Tracé d'un champ de vecteurs

III Représentation graphique 3D

Pour les représentations graphiques dans l'espace et l'utilisation de fonctions mathématiques, on réalisera les importations suivantes :

```
import numpy as np, matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
```

1 Tracé d'une courbe paramétrée dans l'espace

Une *courbe paramétrée* dans l'espace de paramétrage $\mathrm{I} \to \mathbb{R}^3, t \mapsto (x(t), y(t), z(t))$ avec I un intervalle de $\mathbb{R}$ est l'ensemble des points

$$\{(x(t), y(t), z(t)), t \in \mathrm{I}\}$$

```
tt=np.linspace(0,20,100)
tx=tt*np.cos(tt)
ty=tt*np.sin(tt)
tz=tt
fig=plt.figure()
ax=fig.gca(projection='3d')
ax.plot(tx,ty,tz)
plt.show()
```

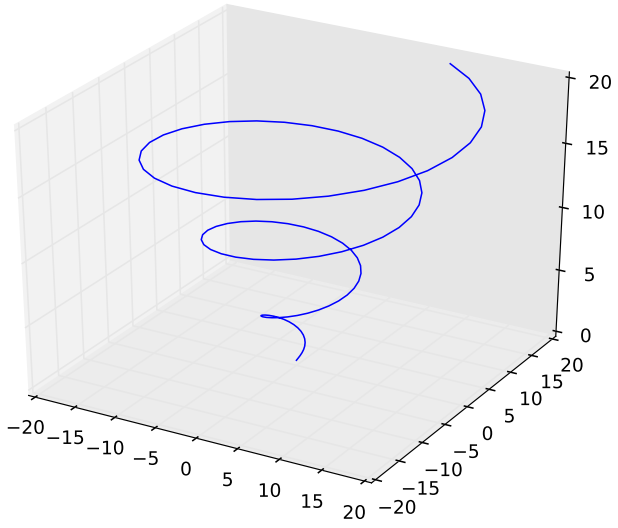

FIGURE 15 – Hélice tracée sur un cône

2 Tracé d'une surface d'équation $z = f(x, y)$

Pour représenter une *surface* d'équation $z = f(x, y)$, on représente l'ensemble des points

$$\{(x, y, f(x, y)),\ (x, y) \in \mathrm{D}\}$$

avec D un pavé de $\mathbb{R}^2$. La fonction f doit être vectorisée. Par exemple, pour la fonction f définie par

$$\forall (x, y) \in \mathbb{R}^2 \qquad f(x, y) = \begin{cases} \dfrac{(xy)^2}{x^2 + y^2} & \text{si } (x, y) \neq (0, 0) \\ 0 & \text{sinon} \end{cases}$$

on saisit :

```
def f(x,y):
    if x==0 and y==0:
        return 0
    else:
        return (x*y)**2/(x**2+y**2)

f=np.vectorize(f)
tx=np.linspace(-1,1,200)    # tableau des valeurs de x
ty=np.linspace(-1,1,200)    # tableau des valeurs de x
X,Y=np.meshgrid(tx,ty)      # grille des points (x,y)
Z=f(X,Y)
fig=plt.figure()
ax=fig.gca(projection='3d')
ax.plot_surface(X,Y,Z,cmap=cm.coolwarm)
plt.show()
```

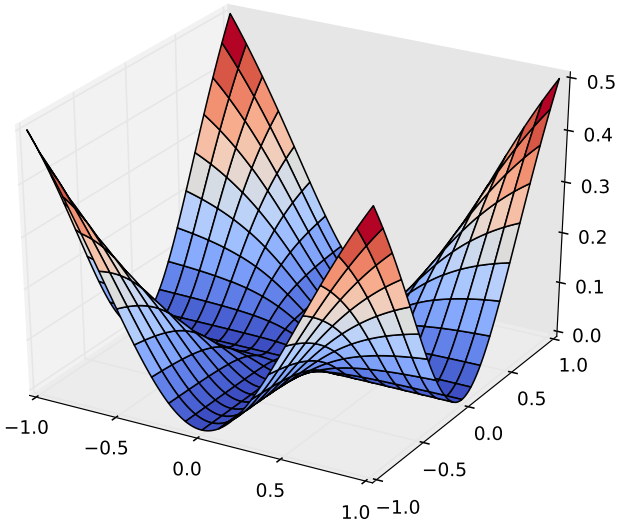

FIGURE 16 – Graphe de $z = f(x, y)$

3 Tracé d'une surface paramétrée

Une *surface paramétrée* de paramétrage $\mathrm{D} \to \mathbb{R}^3, (u, v) \mapsto (x(u, v), y(u, v), z(u, v))$ avec $\mathrm{D} \subset \mathbb{R}^2$ est l'ensemble des points

$$\{(x(u, v), y(u, v), z(u, v)), (u, v) \in \mathrm{D}\}$$

On peut paramétrer un tore de révolution par

$$\forall (\theta, t) \in [\,0\,;2\pi\,]^2 \qquad \begin{cases} x(t, \theta) = \cos(\theta)(\mathrm{R} + r\cos(t)) \\ y(t, \theta) = \sin(\theta)(\mathrm{R} + r\cos(t)) \\ z(t, \theta) = r\sin(t) \end{cases}$$

```
R,r=3,1
tu=np.linspace(0,2*np.pi,30)
tt=np.linspace(0,2*np.pi,30)
T,U=np.meshgrid(tt,tu)        # grille des points (t,u)
X=np.cos(U)*(R+r*np.cos(T))
Y=np.sin(U)*(R+r*np.cos(T))
Z=r*np.sin(T)
fig=plt.figure()
ax=fig.gca(projection='3d')
ax.plot_surface(X,Y,Z,color='w',cstride=1,rstride=1)
ax.set_xlim3d(-3,3)           # plage de valeurs en x
ax.set_ylim3d(-3,3)           # plage de valeurs en y
ax.set_zlim3d(-3,3)           # plage de valeurs en z
plt.show()
```

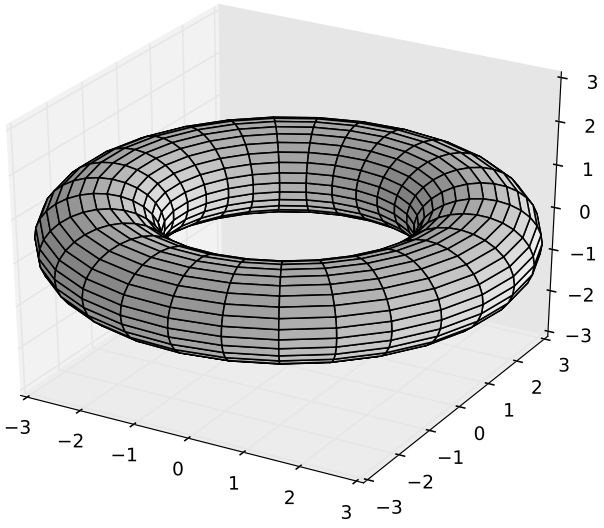

FIGURE 17 – Tore de révolution

Ce document ne dévoile qu'un bref aperçu des possibilités graphiques sous python. Le lecteur curieux pourra consulter le site [1] ou l'ouvrage [2] pour découvrir des fonctionnalités avancées.

Références

[1] Matplotlib : Visualization with Python, `http://matplotlib.org`

[2] Robert Johansson, *Numerical Python*, Apress, 2019