Graphes dans le plan et dans l'espace en Python

1 Tracés de surfaces

Soit f une fonction de deux variables x, y donnée au préalable. Pour tracer le graphe de f dans $\mathbb{R}^3$, c'est-à-dire la surface dans l'espace $\mathbb{R}^3$ d'équation $z = f(x, y)$ sur l'ensemble $[a, b] \times [c, d]$, on procèdera de la façon suivante en Python :

1. Tout d'abord, on importe la librairie `matplotlib.pyplot` avec l'instruction :

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

2. Ensuite, on importe la fonction `Axes3D` de la librairie `mpl_toolkits.mplot3d` avec l'instruction :

```
from mpl_toolkits.mplot3d import Axes3D

ax=Axes3D(plt.figure())
```

3. Puis, on utilisera la commande `meshgrid` qui permet, étant donnés deux vecteurs $x = (x_0, ..., x_p)$ et $y = (y_0, ..., y_q)$ donnés au préalable, de créer un maillage, c'est-à-dire de construire la grille dont les composantes sont les couples (x_i, y_j) avec $(i, j) \in [\![0, p]\!] \times [\![0, q]\!]$. Pour cela, on commencera par créer les vecteurs x et y (par exemple avec la commande `np.linspace`), et après on écrira l'instruction :

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

   A noter que, si l'on ne précise pas le nombre de subdivision choisie dans la commande `np.linspace`, c'est-à-dire si l'on écrit juste `x=np.linspace(a,b)` au lieu de `x=np.linspace(a,b,n)`, alors le vecteur x aura par défaut 50 composantes. A noter aussi que l'on a besoin d'un maillage suffisamment fin pour donner l'illusion lors du tracé d'une surface lisse.

4. Enfin, on utilisera la commande `ax.plot_surface`, sous la forme suivante :

```
ax.plot_surface(X,Y,f(X,Y))
```

   A noter qu'évidemment, on aura besoin d'avoir défini la fonction f utilisée au préalable. A noter aussi qu'on fera suivre la commande `ax.plot_surface` de l'instruction `plt.show()`, de façon à pouvoir voir le tracé de la surface. A noter enfin que l'on peut tracer plusieurs surfaces en même temps. Il suffit pour cela de répéter la commande `ax.plot_surface` avec des fonctions différentes. Dans ce cas, les différentes surfaces tracées apparaîtront avec des couleurs distinctes.

Exemple 1.1. *Considérons la fonction $f : (x, y) \longmapsto x^2(1 + y)^3 + 7y^2$. Pour tracer dans l'espace $\mathbb{R}^3$ la surface d'équation $z = f(x, y)$ avec $-1 \leq x \leq 1$ et $-2 \leq y \leq 2$, on utilisera le script suivant :*

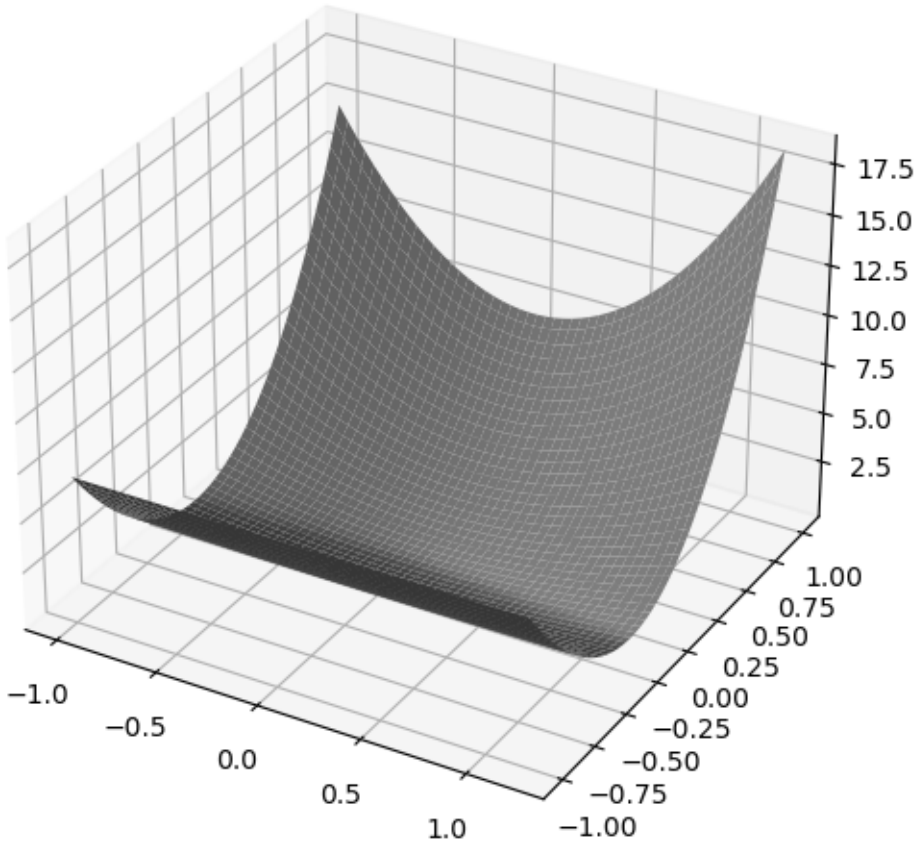
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

ax=Axes3D(plt.figure())

def f(x,y):
    return (x**2)*((1+y)**3)+7*(y**2)

x=np.linspace(-1,1)
y=np.linspace(-2,2)
X,Y=np.meshgrid(x,y)
ax.plot_surface(X,Y,f(X,Y),color='grey')
plt.show()
```

Dans ce cas, le résultat obtenu par Python sera le suivant :



2 Tracé de lignes de niveau

Soit f une fonction de deux variables x, y donnée au préalable et définie sur un ensemble U. Rappelons que la ligne de niveau r de f est l'ensemble :

$$f^{-1}(\{r\}) = \{(x, y) \in U |\ f(x, y) = r\}\,.$$

Pour tracer la ligne de niveau r de la fonction f à l'intérieur du rectangle $[a, b] \times [c, d]$, on utilisera la commande `plt.contour` en Python, sous la forme `plt.contour(X,Y,f(X,Y),[r])`, après avoir réalisé un maillage du rectangle $[a, b] \times [c, d]$ avec la commande `np.meshgrid`. Plus précisément, on écrira la liste d'instructions suivante en Python :

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

def f(x,y):
    .......
    return ....

x=np.linspace(a,b)
y=np.linspace(c,d)
X,Y=np.meshgrid(x,y)
plt.contour(X,Y,f(X,Y),[r])
plt.show()
```

A noter qu'ici, on n'aura pas besoin des commandes du paragraphe précédent, puisque le tracé des lignes de niveau se fera dans le plan. A noter aussi que, pour tracer plusieurs lignes de niveau de f, à savoir les lignes de niveau $r_1, ..., r_n$, on pourra utiliser la commande `plt.contour(X,Y,f(X,Y),[r1,...,rn])`. A noter enfin que la commande `plt.contour(X,Y,f(X,Y),n)` permet de tracer n lignes de niveau équiréparties entre la plus petite et la plus grande valeur de f sur le rectangle $[a, b] \times [c, d]$. Dans tous les cas, si l'on trace plusieurs lignes de niveau, alors elles apparaîtront dans différentes couleurs.

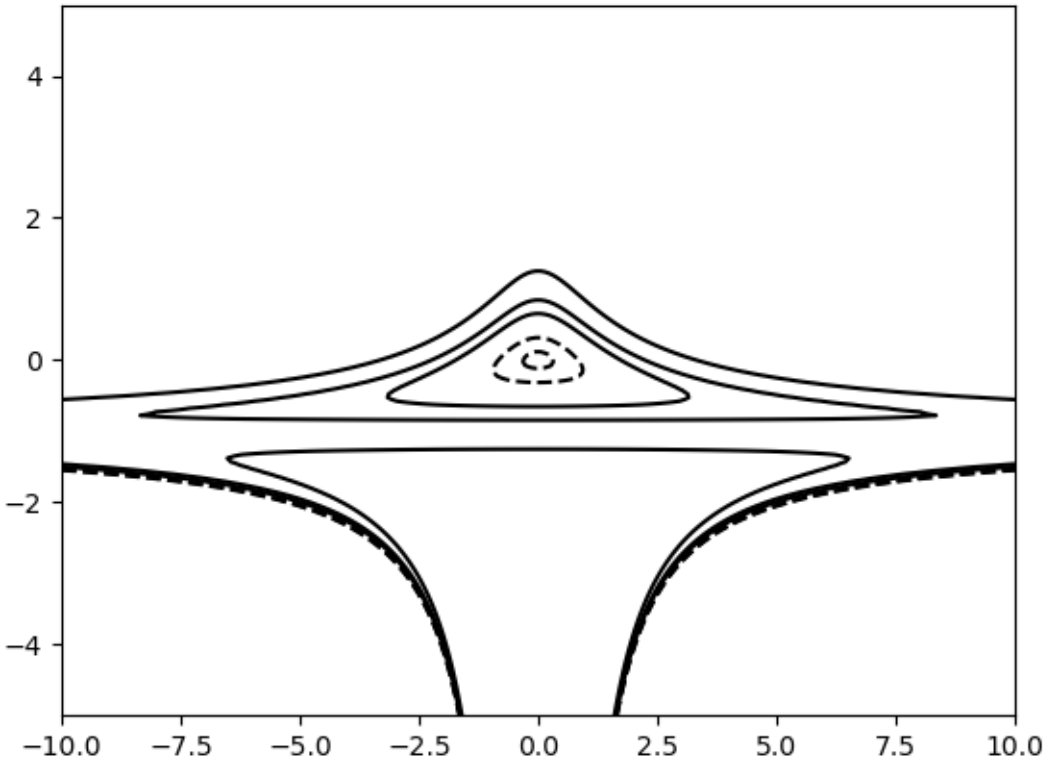
Exemple 2.1. *Considérons la fonction* $f : (x, y) \longmapsto x^2(1 + y)^3 + 7y^2 - 1$. *Pour tracer dans le plan* $\mathbb{R}^2$ *les lignes de niveau* $-0.9, -0.3, 2, 4, 10$ *de la fonction* f, *et ce à l'intérieur du rectangle* $[-10, 10] \times [-5, 5]$, *on utilisera le script suivant :*

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

def f(x,y):
    return (x**2)*((1+y)**3)+7*(y**2)-1

x=np.linspace(-10,10,200)
y=np.linspace(-5,5,200)
X,Y=np.meshgrid(x,y)
plt.contour(X,Y,f(X,Y),[-0.9,-0.3,2,4,10],colors='black')
plt.show()
```

Dans ce cas, le résultat obtenu par Python sera le suivant :



A noter que, dans l'exemple ci-dessus, on a augmenté la précision dans les commandes `np.linspace` de façon à avoir des lignes de niveau les plus lisses possibles. A noter aussi que, si l'on utilise comme dans l'exemple ci-dessus la couleur noire pour représenter différentes lignes de niveau, alors celles qui correspondent à des valeurs négatives du niveau r seront représentées par des lignes en pointillés. En particulier, on peut conjecturer dans l'exemple ci-dessus que f présente un minimum local en $(0, 0)$.

3 Tracé de champs de vecteurs et de gradients

Pour représenter une famille de vecteurs du plan $\mathbb{R}^2$, lesquels sont placés en des points donnés de ce plan (ce que l'on appelle un *champ de vecteurs*), on dispose en Python de la commande `plt.quiver`. Plus précisément, étant donnés des vecteurs $x = (x_1, ..., x_n)$, $y = (y_1, ..., y_n)$, $u = (u_1, ..., u_n)$ et $v = (v_1, ..., v_n)$, la commande `plt.quiver` permet de tracer n^2 flèches dont les origines sont les points (x_i, y_j) et dont les directions sont données par les vecteurs (u_i, v_j), et ce pour tout $(i, j) \in [\![1, n]\!]^2$. Elle s'utilise comme suit :

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

x=np.array([x1,...,xn])
y=np.array([y1,...,yn])
u=np.array([u1,...,un])
v=np.array([v1,...,vn])
X,Y=np.meshgrid(x,y)
U,V=np.meshgrid(u,v)
plt.quiver(X,Y,U,V)
plt.show()
```

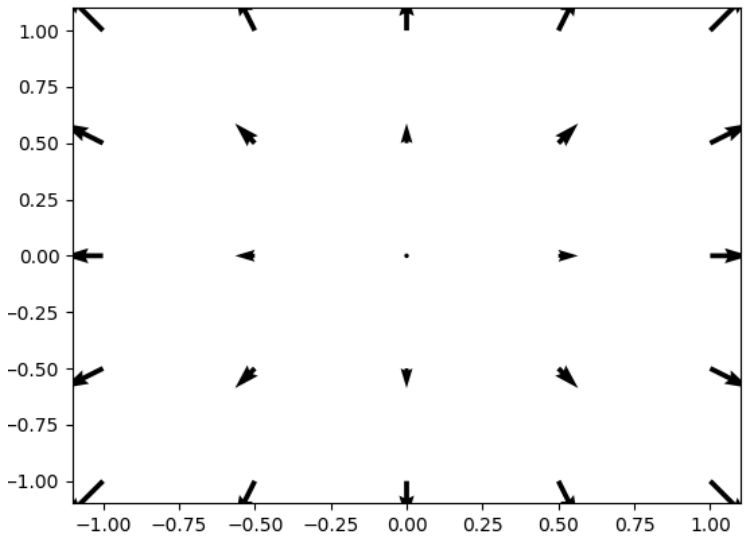
A noter qu'on a besoin ici d'utiliser la commande `np.meshgrid` au préalable pour pouvoir créer toutes les familles de points (x_i, y_j) et de vecteurs (u_i, v_j) lorsque le couple d'indices (i, j) parcourt $[\![1, n]\!]^2$. A noter aussi que Python ne trace pas exactement les vecteurs (u_i, v_j). Plus précisément, Python applique à tous ces vecteurs un coefficient de proportionnalité qui en modifie les longueurs mais pas les directions, et ce afin que le graphe obtenu soit plus clair.

Exemple 3.1. *Considérons les vecteurs* $x = (-1, -0.5, 0, 0.5, 1)$, $y = x$, $u = (-2, -1, 0, 1, 2)$ *et* $v = u$. *Pour appliquer la commande* `plt.quiver` *à ces quatre vecteurs, on utilisera le script suivant :*

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

x=np.array([-1,-0.5,0,0.5,1])
y=np.array([-1,-0.5,0,0.5,1])
u=np.array([-2,-1,0,1,2])
v=np.array([-2,-1,0,1,2])
X,Y=np.meshgrid(x,y)
U,V=np.meshgrid(u,v)
plt.quiver(X,Y,U,V)
plt.show()
```

Dans ce cas, le résultat obtenu par Python sera le suivant :



Considérons à présent une fonction f de classe $\mathcal{C}^1$ sur un ouvert non vide Ω de $\mathbb{R}^2$. On rappelle que le gradient de f en tout point (x, y) de Ω est défini par :

$$\nabla(f)(x, y) = (\partial_1(f)(x, y), \partial_2(f)(x, y)),$$

où $\partial_1(f)(x,y)$ et $\partial_2(f)(x,y)$ désignent les dérivées partielles d'ordre 1 de f en (x,y). Pour obtenir une représentation graphique du gradient de f, on commencera par calculer explicitement ce gradient. Ensuite, on choisira deux vecteurs $x = (x_1, ..., x_n)$ et $y = (y_1, ..., y_n)$ qui seront les listes des abscisses et des ordonnées des points (x_i, y_j) où on va calculer le gradient. Enfin, on utilisera la commande `plt.quiver` pour la représentation graphique.

Exemple 3.2. *Considérons la fonction* $f : (x,y) \longmapsto xye^{-\frac{x^2+y^2}{2}}$. *On souhaite représenter dans le plan le gradient de* f, *et ce à l'intérieur du rectangle* $[-2,2] \times [-2,2]$. *Pour ce faire, on commence par remarquer à l'aide de calculs simples que, pour tout* $(x,y) \in \mathbb{R}^2$ *:*

$$\nabla(f)(x,y) = \left(y(1-x^2)e^{-\frac{x^2+y^2}{2}}, x(1-y^2)e^{-\frac{x^2+y^2}{2}} \right).$$

Dès lors, on pourra utiliser le script suivant :

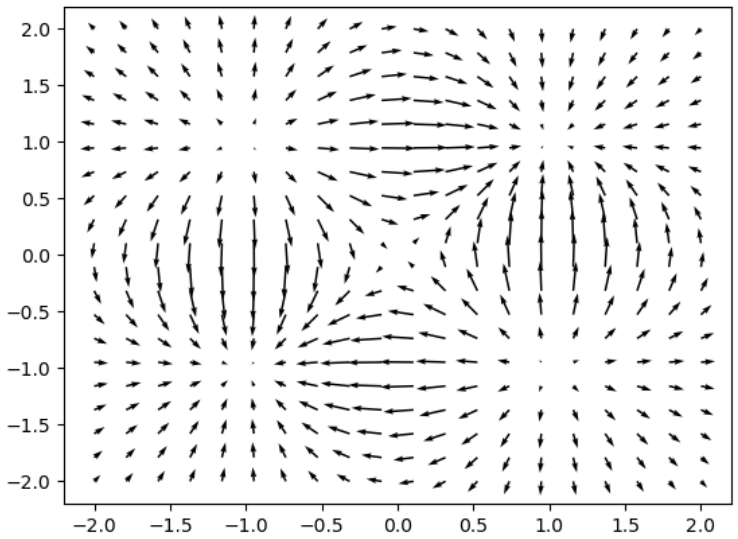
```
import numpy as np
import matplotlib.pyplot as plt

def df1(x,y):
    return y*(1-x**2)*np.exp(-0.5*(x**2+y**2))

def df2(x,y):
    return x*(1-y**2)*np.exp(-0.5*(x**2+y**2))

x=np.linspace(-2,2,20)
y=np.linspace(-2,2,20)
X,Y=np.meshgrid(x,y)
plt.quiver(X,Y,df1(X,Y),df2(X,Y))
plt.show()
```

Dans ce cas, le résultat obtenu par Python sera le suivant :



A noter que, dans l'exemple ci-dessus, on détecte cinq zones où les vecteurs semblent diminuer jusqu'à s'annuler. Ces zones correspondent aux points critiques de la fonction f. De plus, lorsque l'on représente un gradient, les vecteurs semblent se suivre selon des lignes, que l'on appelle des *lignes de champ*. Comme le gradient nous donne la direction selon laquelle la fonction f augmente le plus vite, les minima de f correspondent aux points d'où partent les lignes de champ, et les maxima de f correspondent aux points où convergent les lignes de champ. On en repère 2 de chaque type ici, ce qui laisse présager que la fonction présente 2 minima et 2 maxima, ce que l'on pourra vérifier à l'aide des résultats sur la recherche d'extrema.

Reste un dernier point, au centre de la figure, où les lignes de champ semblent se partager (certaines y convergent, d'autres s'en éloignent) : il s'agit d'un point-selle (ou col), ce que l'on pourra aussi vérifier à l'aide des résultats sur la recherche d'extrema.