Chapitre 5

Représentations avancées en 2D et 3D (avec Matplotlib)

5.1 Utilisation avancée de Matplotlib

Nous allons revenir sur la bibliothèque graphique Matplotlib (http://matplotlib.org/) qui nous a permis de représenter aisément et avec précision une fonction $y = f(x)$.
Nous allons voir qu'il est également possible :

- d'améliorer encore la **présentation de la figure obtenue** ;
- de représenter des **courbes paramétriques** telles que $(x(t), y(t))$, où t est le paramètre ;
- de tracer des **courbes de niveau**, c'est-à-dire des représentations particulières d'une fonction réelle à deux variables réelles : $z = f(x, y)$.

De surcroît, Matplotlib offre de très intéressantes possibilités en matière de **représentations en trois dimensions**.
Notons que des alternatives existent. On peut citer, en particulier, le paquet Python **Mayavi** (https://docs.enthought.com/mayavi/mayavi/).

5.2 Visualisation d'une ou plusieurs fonctions $f(x)$

Nous avons déjà eu l'occasion de créer un **objet `figure`** et de représenter plusieurs graphes de fonctions dans cette figure :

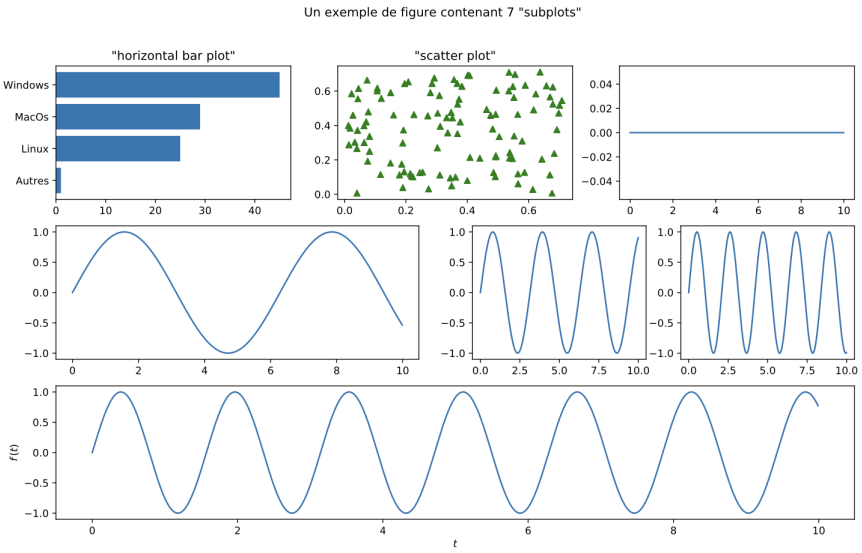
- on crée la figure à l'aide de `plt.figure` ;
- on représente chacune des fonctions grâce à `plt.plot`.

Tous les graphes partagent alors le même **objet `axes`**.
Il est également possible de partager la figure créée en $m \times n$ emplacements (m lignes et n colonnes) destinés à recevoir chacun un objet `axes`. Ce partage peut se faire en invoquant `subplot(m,n,a)`, où `a` sélectionne le $a$ème emplacement pour les instructions graphiques suivantes. La numérotation se fait de gauche à droite et de haut en bas.
L'exemple ci-dessous consiste en une figure possédant 7 `subplot` dont deux sont particuliers : le premier est un "barplot" horizontal et le second un "scatter plot" ("nuage de points"). Dans ce dernier, nous utilisons NumPy pour créer un ensemble de points au hasard.

```
import numpy as np
import matplotlib.pyplot as plt
# preparation du contenu du "barplot"
OS = ('Autres','Linux','MacOs','Windows')
h_pos = np.arange(len(OS))
pourcentage = [1,25,29,45]
# preparation du nuage de points
N = 110
x_s = 0.72*np.random.rand(N)
y_s = 0.72*np.random.rand(N)
# preparation du contenu des autres objets "axes"
def f(t,a):
    return np.sin(a*t)
t = np.arange(0., 10., 0.01)
y = [] # liste de fonctions evaluees
for i in range(5):
    y.append(f(t,i))
# creation de la figure
plt.figure(figsize=(14,8)) # taille de la (grande) figure (en
 ↪  pouces)
fig = plt.figure(1)
fig.suptitle('Un exemple de figure contenant 7 "subplots"')
# 1er objet "axes"
ax1 = plt.subplot(331)
ax1.barh(h_pos,pourcentage)
ax1.set_title('"horizontal bar plot"')
ax1.set_yticks(h_pos)
ax1.set_yticklabels(OS)
# 2eme objet "axes"
ax2 = plt.subplot(332)
ax2.scatter(x_s, y_s, marker='^', color='green')
ax2.set_title('"scatter plot"')
# 3eme objet "axes"
ax3 = plt.subplot(333)
ax3.plot(t, y[0])
# 4eme objet "axes"
ax4 = plt.subplot(323)
ax4.plot(t, y[1])
# 5eme objet "axes"
ax5 = plt.subplot(347)
ax5.plot(t, y[2])
# 6eme objet "axes"
ax6 = plt.subplot(348)
ax6.plot(t, y[3])
# 7eme objet "axes"
ax7 = plt.subplot(313)
```

```
ax7.plot(t, y[4])
ax7.set_xlabel('$t$')
ax7.set_ylabel('$f\,(t)$')
# affichage de la figure
plt.show()
```

Figure obtenue :



5.3 Représentation d'un champ scalaire

La température est un exemple de **champ scalaire** : il s'agit d'une grandeur scalaire dont la valeur (un nombre) varie selon le point de l'espace considéré : $T = f(x, y, z)$.
Une **fonction scalaire de deux variables** $f(x, y)$, par exemple la température sur une plaque (surface), peut être visualisée :

- soit **en deux dimensions** à l'aide de **courbes de niveau** ($z_0 = f(x, y)$) ;
- soit **en trois dimensions** sous la forme d'une **surface** permettant de visualiser la valeur $z = f(x, y)$ de la fonction.

Nous allons représenter le champ scalaire bidimensionnel suivant :

$$f(x, y) = \frac{10x}{1 + x^2 + y^2} \, .$$

5.3.1 Définition d'une grille de points en 2D (fonction `np.meshgrid`)

Avant de pouvoir représenter la fonction $f(x, y)$, il est nécessaire de créer une **grille rectangulaire de points** dans le plan xy :

```
import numpy as np
x = y = np.linspace(-4.,4.,41)
xv, yv = np.meshgrid(x,y,indexing='ij')
```

A partir de coordonnées `x` et `y` également espacées, la fonction `np.meshgrid` produit les coordonnées "vectorisées" `xv` et `yv` (`xv` et `yv` sont des tableaux 41×41 de dimension 2).
On calcule alors la valeur de f aux 41×41 points de la grille :

```
fv = 10*xv/(1 + xv**2 + yv**2)
```

Le résultat retourné, `fv`, est une matrice 41×41 (c'est-à-dire un tableau de dimension 2).

5.3.2 Représentations en 2D

Pour représenter le champ scalaire **en deux dimensions**, il est nécessaire d'importer Matplotlib, avant de créer une figure et un système d'axes :

```
import matplotlib.pyplot as plt
fig = plt.figure(1) # pour créer la figure
ax = fig.gca()      # pour créer les axes ("get the current axes")
```

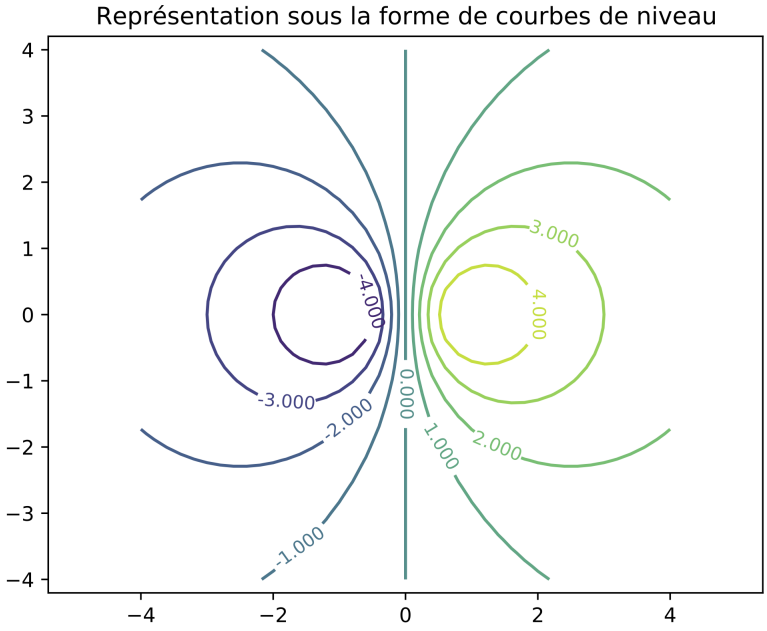
En appelant la fonction `contour`, on obtient alors, par exemple, 9 **courbes de niveau** dont les valeurs sont précisées sur le dessin :

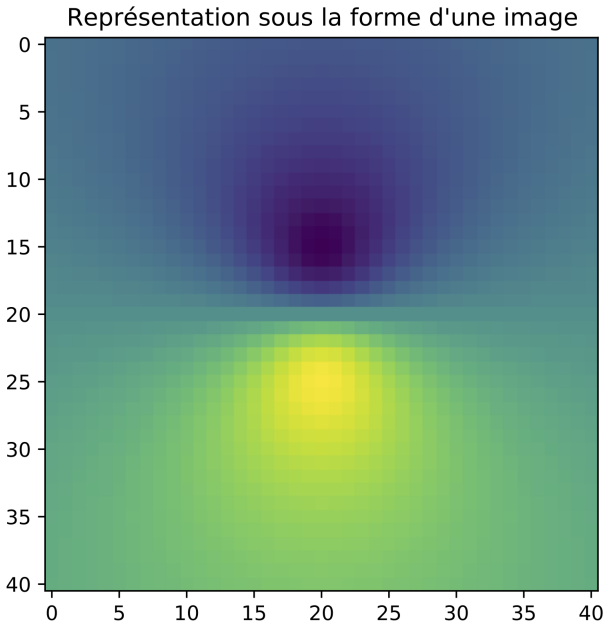
```
plt.title("Représentation sous la forme de courbes de niveau")
cs = ax.contour(xv,yv,fv,9) # "contour set"
plt.clabel(cs, inline=True, fontsize=9)
plt.axis('equal')
plt.show()
```

On peut également afficher la fonction $f(x, y)$ comme une **image** dont les couleurs reflètent les valeurs que prend la fonction :

```
import numpy as np
x = y = np.linspace(-4.,4.,41)
xv, yv = np.meshgrid(x,y,indexing='ij')
#
fv = 10*xv/(1 + xv**2 + yv**2)
#
import matplotlib.pyplot as plt
fig = plt.figure(1) # pour creer la figure
ax = fig.gca()      # pour creer les axes ("get the current axes")
#
plt.title("Représentation sous la forme d'une image")
ax.imshow(fv)
plt.show()
```

Types de représentations bidimensionnelles envisageables pour un champ scalaire :





Remarque : la fonction `plt.axis('equal')` permet de faire en sorte que l'échelle soit identique sur les deux axes. En remplaçant `'equal'` par `'scaled'`, on contraint de plus le cadre du graphe à être carré.

5.3.3 Représentations en 3D

La fonction `ax.contour` s'applique également en trois dimensions :

```
%matplotlib notebook
# La ligne ci-dessus permet en principe de modifier
# l'angle de vue de la représentation.
# Si vous utilisez noto, il est possible que vous deviez
# installer "ipympl" (commande: pip install ipympl) puis
# faire précéder le code de la ligne %matplotlib ipympl
# (au lieu de %matplotlib notebook)
#
import numpy as np
x = y = np.linspace(-4.,4.,41)
xv, yv = np.meshgrid(x,y,indexing='ij')
#
fv = 10*xv/(1 + xv**2 + yv**2)
#
import matplotlib.pyplot as plt
fig = plt.figure(1)
ax = plt.axes(projection='3d')
ax.contour(xv,yv,fv,9)
ax.set_title("Representation sous la forme de courbes de niveau
 ↪  (en 3D)")
plt.show()

```

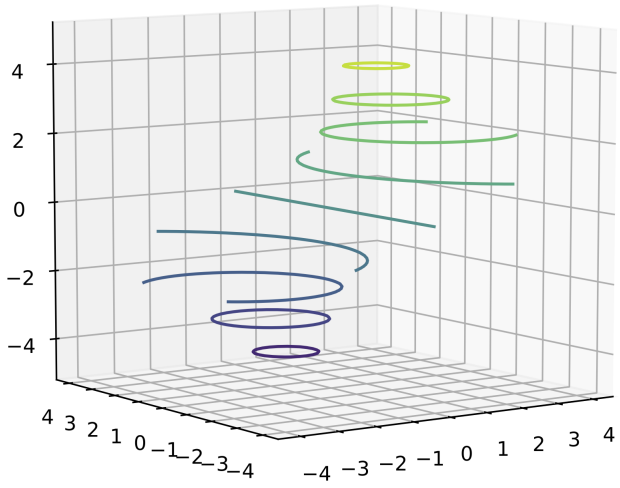
Pour visualiser le champ scalaire sous forme d'une **surface en trois dimensions**, les lignes suivantes s'avèrent nécessaires :

```
%matplotlib notebook
import numpy as np
x = y = np.linspace(-4.,4.,41)
xv, yv = np.meshgrid(x,y,indexing='ij')
#
fv = 10*xv/(1 + xv**2 + yv**2)
#
import matplotlib.pyplot as plt
from matplotlib import cm
fig = plt.figure(1)
ax = plt.axes(projection='3d')
ax.plot_surface(xv,yv,fv,cmap = cm.coolwarm)
ax.set_title("Représentation 3D de la surface $z=f(x,y)$")
plt.show()
```

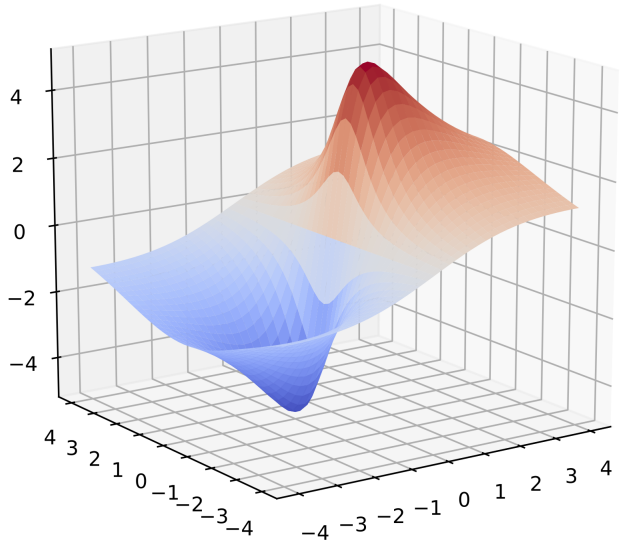
La fonction `ax.plot_surface` utilise ici une carte des couleurs (`cmap`) qui associe une couleur "chaude" (rouge) aux grandes valeurs de f et une couleur "froide" (bleue) aux valeurs les plus petites.

Types de représentations tridimensionnelles envisageables pour un champ scalaire :

Représentation sous la forme de courbes de niveau (en 3D)



Représentation tridimensionnelle de la surface $z = f(x, y)$



5.4 Représentation d'une courbe paramétrée

5.4.1 Représentation en 2D

En **deux dimensions**, une **courbe paramétrée** par le paramètre t est donnée par $\vec{r}(t) = x(t)\vec{e}_x + y(t)\vec{e}_y$, où les vecteurs $\vec{e}_x$ et $\vec{e}_y$ sont des vecteurs unitaires dans les directions x et y, respectivement.
En guise d'exemple, nous allons considérer une courbe appelée **trochoïde** définie par :

$$\begin{cases} x(t) &= Rt - d\sin t\,, \\ y(t) &= R - d\cos t\,. \end{cases}$$

Remarque 5.1. La **cycloïde** est un cas particulier de cette courbe que l'on obtient en posant $d = R$. Si vous vous intéressez aux courbes mathématiques en deux et trois dimensions, ne manquez pas de visiter le site suivant : https://mathcurve.com/. ◇

Les coordonnées du vecteur position $\vec{r}(t)$ peuvent être définies de la manière suivante :

```
import numpy as np
import matplotlib.pyplot as plt
R = 3
d = 4
t = np.linspace(-2*np.pi,2*np.pi,1000)
courbe_x = R*t - d*np.sin(t)
courbe_y = R - d*np.cos(t)
```

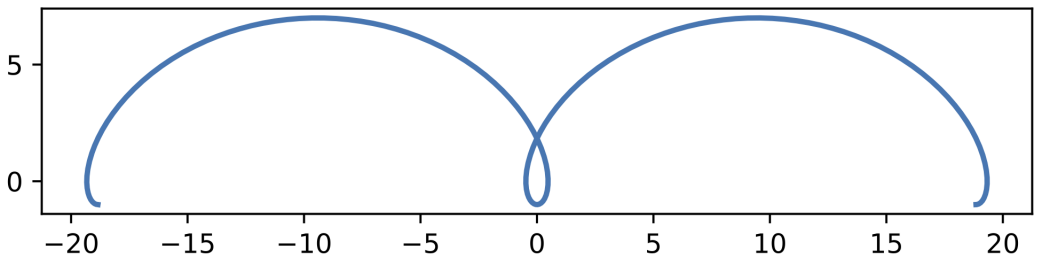
Après avoir créé les axes, on peut afficher la courbe :

```
fig = plt.figure(1)
ax = fig.gca()
ax.plot(courbe_x, courbe_y, linewidth=2)
plt.axis('scaled')
plt.show()
```

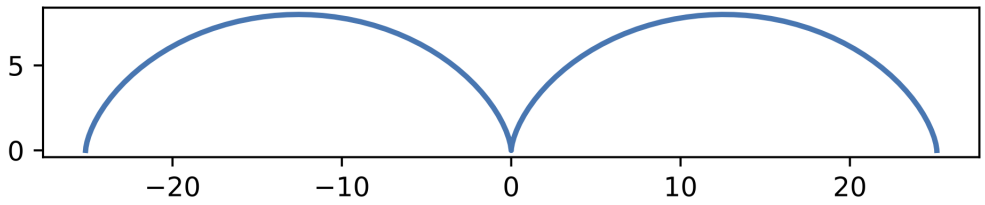
Remarque 5.2. On pourrait ici également simplement utiliser le code suivant :

```
plt.plot(courbe_x, courbe_y, linewidth=2)
plt.axis('scaled')
plt.show()
```

◇



Cas particulier de la cycloïde :



5.4.2 Représentation en 3D

En **trois dimensions**, une **courbe paramétrée** par le paramètre t est donnée par :

$$\vec{r}\,(t) = x(t)\vec{e}_x + y(t)\vec{e}_y + z(t)\vec{e}_z\,,$$

où les vecteurs $\vec{e}_x$, $\vec{e}_y$ et $\vec{e}_z$ dénotent les vecteurs unitaires dans les directions x, y, et z, respectivement.
Après le préambule habituel,

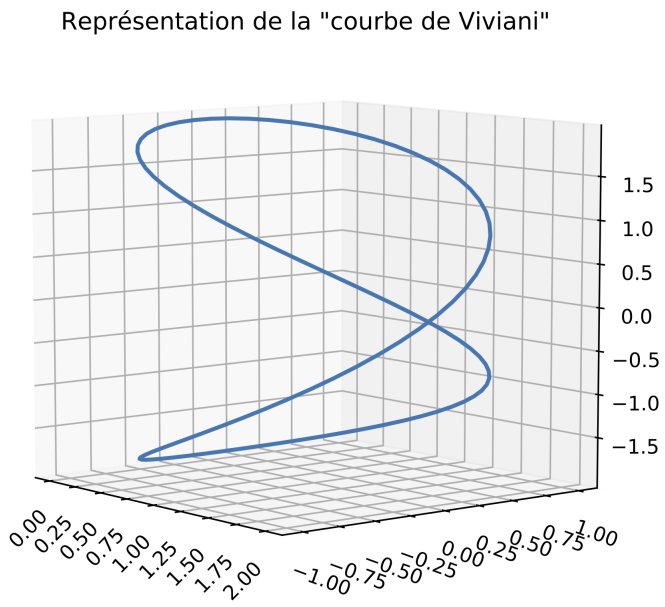
```
import numpy as np
import matplotlib.pyplot as plt
```

les coordonnées du vecteur position $\vec{r}\,(t)$ peuvent être définies de la manière suivante (cas d'une courbe appelée "courbe de Viviani") :

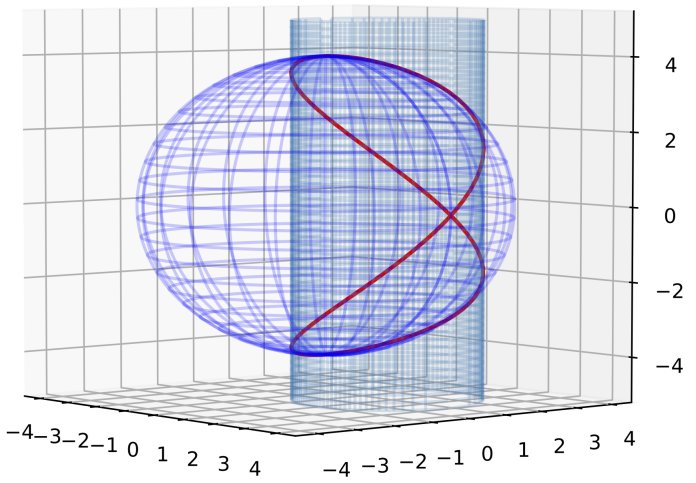
```
a = 1
t = np.linspace(-2*np.pi,2*np.pi,100)
courbe_x = a + a*np.cos(t)
courbe_y = a*np.sin(t)
courbe_z = 2*a*np.sin(t/2)
```

Il est alors possible d'afficher la courbe en prenant soin de faire tourner quelque peu les graduations des axes x et y pour améliorer la lisibilité de la figure :

```
fig = plt.figure(1)
ax = fig.add_subplot(projection='3d')
plt.setp(ax.get_xticklabels(), rotation=45)
plt.setp(ax.get_yticklabels(), rotation=-20)
ax.plot(courbe_x, courbe_y, courbe_z, linewidth=2)
ax.set_title('Représentation de la "courbe de Viviani"')
plt.show()
```



Remarque 5.3. La courbe de Viviani (appelée aussi "fenêtre de Viviani") est définie comme l'**intersection d'une sphère et d'un cylindre** qui est tangent à la sphère et passe par le centre de cette dernière :



◇

5.5 Représentation d'un champ vectoriel (en 2D)

5.5.1 Champ vectoriel et potentiel scalaire

En physique, certaines grandeurs sont décrites par des **champs vectoriels**. Ainsi, le champ électrique $\vec{E}$ est une grandeur vectorielle dont les composantes varient selon le point de l'espace considéré :

$$\vec{E} = \begin{pmatrix} E_x(x,y,z) \\ E_y(x,y,z) \\ E_z(x,y,z) \end{pmatrix} .$$

De plus, certains de ces champs vectoriels "dérivent d'un potentiel", c'est-à-dire d'un champ scalaire. Dans le cas de $\vec{E}$, ce champ scalaire est le potentiel électrique $\phi = f(x,y,z)$. Plus précisément,

$$\vec{E} = -\vec{\nabla}\phi = -\begin{pmatrix} \partial\phi/\partial x \\ \partial\phi/\partial y \\ \partial\phi/\partial z \end{pmatrix} ,$$

où $\vec{\nabla}\phi$ est appelé le **gradient** de ϕ et où $\partial/\partial x$, $\partial/\partial y$ et $\partial/\partial z$ sont des **dérivées partielles**.

5.5.2 La fonction `quiver`

Le gradient d'un champ scalaire en un point est un vecteur pointant dans la direction où le champ augmente le plus (direction de "plus grande pente"). Il est donc **perpendiculaire à la courbe de niveau** qui passe par ce point. Ainsi, le champ électrique est toujours **orthogonal aux courbes sur lesquels le potentiel est le même** (**équipotentielles**). Nous allons l'illustrer en deux dimensions à l'aide de la fonction scalaire :

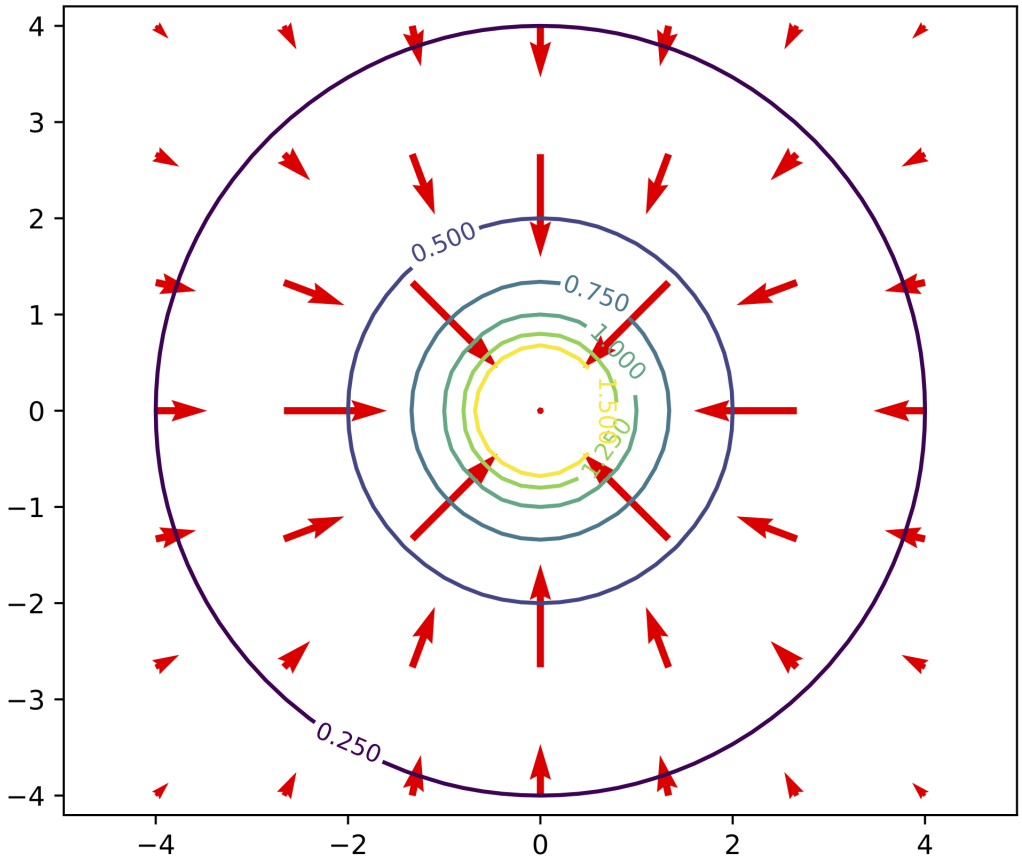
$$\Psi(x,y) = \frac{1}{\sqrt{x^2+y^2}}$$

dont nous allons représenter des **courbes de niveaux** accompagnées de **vecteurs gradients**. Pour ce faire, nous utilisons la fonction `np.gradient` qui retourne le gradient d'un champ donné comme un tableau à N dimensions, ainsi que la fonction `ax.quiver` qui permet de représenter un champ bidimensionnel de flèches.

```
1  import numpy as np
2  x = y = np.linspace(-4.,4.,41)
3  xv, yv = np.meshgrid(x,y,indexing='ij')
4  Psiv = 1/(np.sqrt(xv**2 + yv**2))
5  #
6  x2 = np.linspace(-4.,4.,7)
7  y2 = np.linspace(-4.,4.,7)
8  x2v, y2v = np.meshgrid(x2,y2,indexing='ij')
9  Psi2v = 1/(np.sqrt(x2v**2 + y2v**2))
10 dPsidx, dPsidy = np.gradient(Psi2v)
11 #
```

```
12 import matplotlib.pyplot as plt
13 fig = plt.figure(1)
14 ax = fig.gca()
15 niveaux = [0.25, 0.5, 0.75, 1.0, 1.25, 1.5]
16 cs = ax.contour(xv,yv,Psiv,levels=niveaux)
17 plt.clabel(cs, inline=True, fontsize=9)
18 ax.quiver(x2v,y2v,dPsidx,dPsidy, color='r')
19 plt.axis('equal')
20 plt.show()
```

Représentation de quelques courbes $\Psi = \Psi_0 =$ constante (**cercles de rayon** $r = \sqrt{x^2 + y^2}$ **centrés à l'origine**) et de quelques vecteurs gradient $\vec{\nabla}\Psi$:



5.5.3 La fonction `streamplot`

Matplotlib permet également de représenter un champ vectoriel sous la forme de **lignes de champ**. Ces lignes sont **tangentes**, en tout point, à la direction des vecteurs et permettent d'aisément visualiser l'allure du champ. En revanche, elles ne fournissent pas d'information sur l'**intensité du champ**, sauf si leurs couleurs, leurs épaisseurs ou leur densité sont significatives.
La fonction `streamplot` demande :

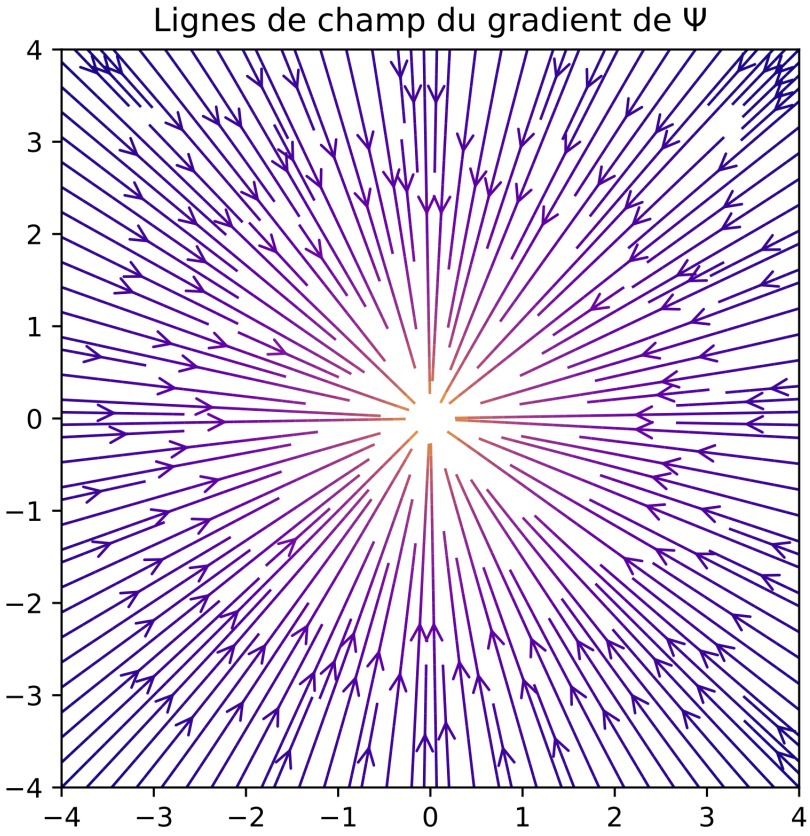
- une grille de points également espacés (c'est-à-dire deux tableaux à 1D `x` et `y`) ;

- deux tableaux 2D fournissant les composantes des vecteurs du champ en chaque point de la grille.

Le paramètre `density` contrôle l'espacement des lignes du champ.

La couleur des lignes (gérée par le paramètre `color` qui doit alors être un tableau) permet quant à elle de visualiser la longueur (norme) des vecteurs du champ.

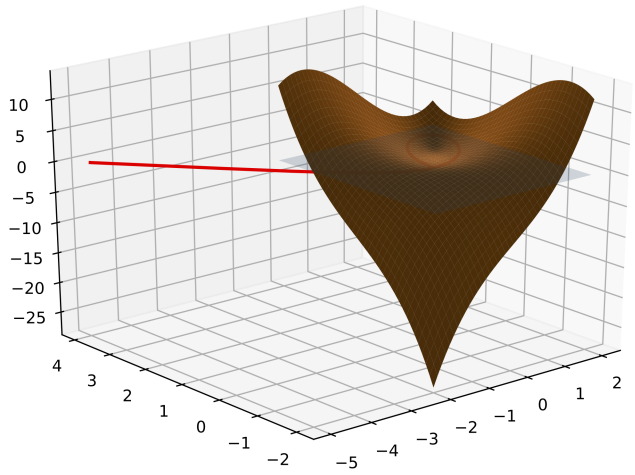
```
import numpy as np
x = np.linspace(-4.,4.,101)
y = np.linspace(-4.,4.,101)
xv, yv = np.meshgrid(x,y,indexing='ij')
Psiv = 1/(np.sqrt(xv**2 + yv**2))
dPsidx, dPsidy = np.gradient(Psiv)
import matplotlib.pyplot as plt
fig = plt.figure(1)
ax = fig.gca()
t_couleurs = 36*np.log(dPsidx**2+dPsidy**2) # tableau des couleurs
ax.streamplot(x, y, dPsidy, dPsidx, color =
 ↪  t_couleurs,linewidth=1,cmap = plt.cm.plasma, density =
 ↪  2,arrowstyle = '->', arrowsize = 1.5)
ax.set_title("Lignes de champ du gradient de $\Psi$")
plt.axis('scaled')
plt.show()
```



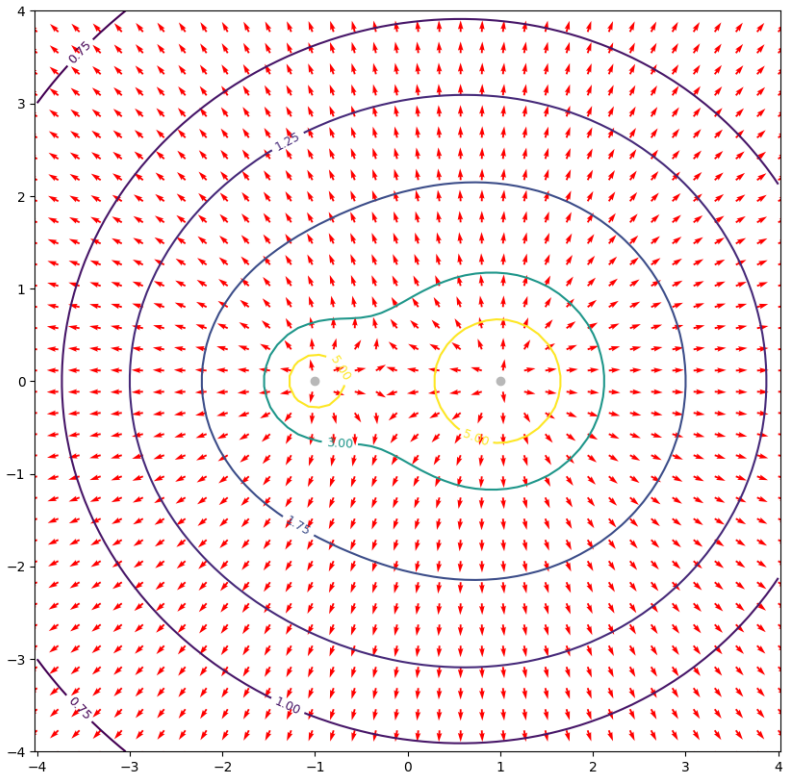
Remarque 5.4. les lignes de champ ne se croisent jamais. ◇

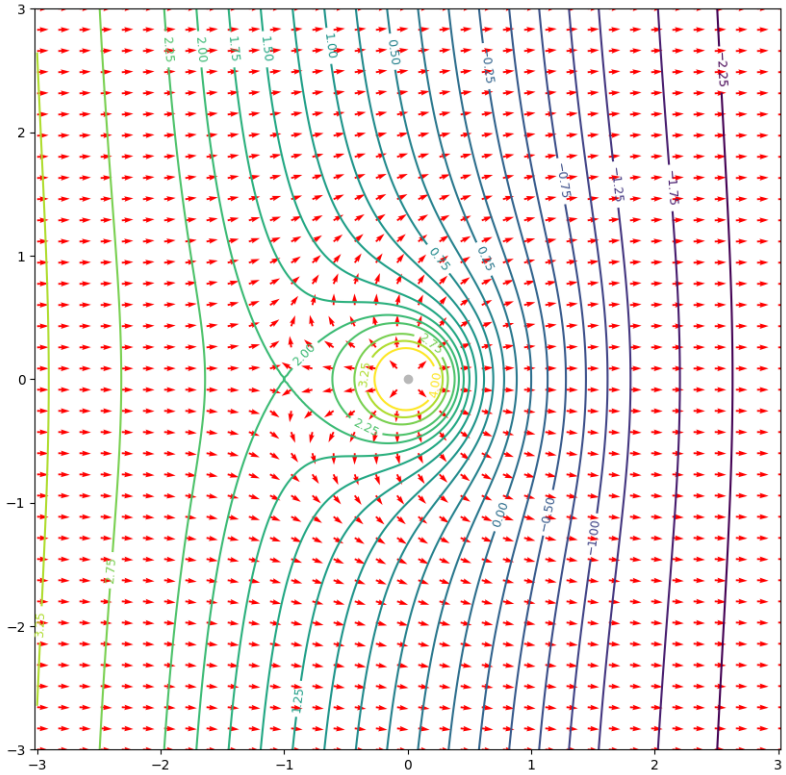
5.6 Exemples de représentations avancées

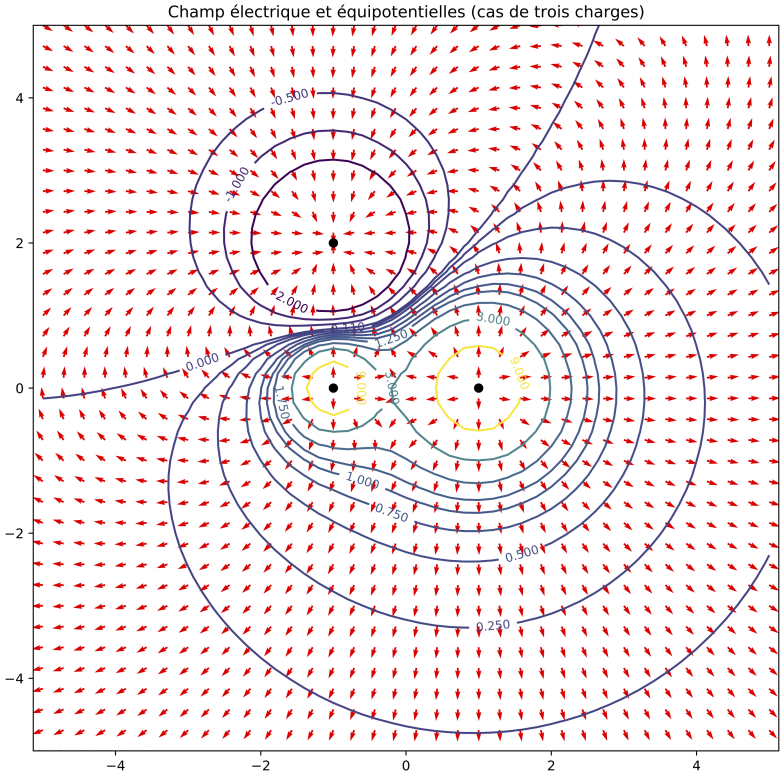
Cette section contient quelques exemples de représentations avancées qui sont, pour certains, discutés en exercices dans la série 18.
Folium de Descartes :



Champ et potentiel électriques :






Champ électrique et équipotentielles (cas de trois charges)
0.500
1.000
2.000
3.000
0.000
1.500
0.750
0.250
4
2
0
−2
−4
−4
−2
0
2
4