L2-S3 : UE Méthodes numériques

SEANCE 3

Fonctions à 2 variables; représentation graphique

22 septembre 2021

Introduction

Dans ce cours, on s'intéresse aux tableaux à deux dimensions. Ils peuvent représenter :

- le résultat d'une fonction à deux variables
- une image
- une matrice
- ...

Lorsqu'on utilise NumPy, il faut **éviter d'utiliser des boucles ! Faire directement les calculs sur des tableaux !**

Fonctions mathématiques à deux variables

On s'intéresse aux fonctions du type $z = f(x, y)$, par exemple

$$f(x, y) = x^2 + y^2$$

A 2 nombres en entrée correspond 1 nombre en sortie.

En python, la définition de cette fonction s'écrit :

In [1]:
```
def ma_fonction(x,y):
    return x*x + y*y
```

Fonctions mathématiques à deux variables

Une des forces de Python réside dans la librairie `numpy`, qui permet des calculs très efficaces sur des tableaux. Peut-on donner des tableaux à `ma_fonction` ?

Oui si on utilise les fonctions mathématiques de `numpy` :

In [2]:
```
def ma_fonction(x,y):
    return np.sin(x*y)   # utiliser numpy
```

mais si `x` et `y` sont des tableaux 1D, alors le résultat sera 1D aussi...

In [3]:
```
import numpy as np
ma_fonction(np.array([1,2,3]),np.array([4,5,6])
    )
```

Out[3]:
```
array([-0.7568025 , -0.54402111, -0.75098725])
```

La sortie de la fonction n'est pas le résultat de la fonction sur toutes les paires (x, y) de $\mathbb{R}^2$ (ou de $[1, 2, 3] \times [4, 5, 6]$ dans l'exemple ci-dessus).

La fonction meshgrid

In [4]:
```
x = np.linspace(0, 2, 5)
y = np.linspace(0, 1, 3)
print("x =",x)
print("y =",y)
```

Out[4]:
```
x = [0.  0.5 1.  1.5 2. ]
y = [0.  0.5 1. ]
```

In [5]:
```
ma_fonction(x,y)
```

Out[5]:
```
---------------------------------------------------------------

ValueError  Traceback (most recent call last)
      1 def ma_fonction(x,y):
----> 2         return np.sin(x*y)
ValueError: operands could not be broadcast
    together
with shapes (5,) (3,)
```

L'exécution ne fonctionne pas car les tableaux `x` et `y` n'ont pas la même taille. Il faut transformer les 2 listes 1D en 2 tableaux 2D contenant toutes les paires de points possibles entre `x` et `y`.

La fonction meshgrid

Ceci est réalisé par la fonction `numpy.meshgrid()`

In [6]:
```
1 xx, yy = np.meshgrid(x, y)
2 print("xx =",xx)
3 print("yy =",yy)
```

Out[6]:
```
xx = [[0.   0.5 1.   1.5 2. ]
 [0.   0.5 1.   1.5 2. ]
 [0.   0.5 1.   1.5 2. ]]
yy = [[0.   0.   0.   0.   0. ]
 [0.5 0.5 0.5 0.5 0.5]
 [1.   1.   1.   1.   1. ]]
```

Le résultat de `meshgrid` contient 2 grilles 2D des coordonnées `x` et `y`.

La fonction meshgrid

La fonction va s'appliquer à toutes les paires de points des tableaux 2D xx et yy :

In [7]:
```
res = ma_fonction(xx,yy)
print(res)
```

Out[7]:
```
array([[ 0., 0.        , 0.        , 0.
       , 0.      ],
          [ 0., 0.24740396, 0.47942554,
       0.68163876, 0.84147098],
          [ 0., 0.47942554, 0.84147098,
       0.99749499, 0.90929743]])
```

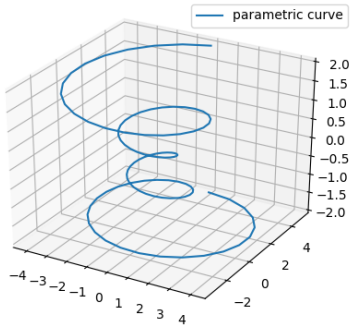
Attention à l'ordre des indices :

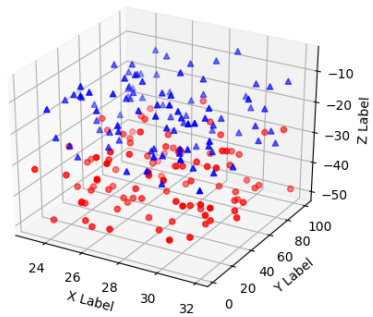
In [8]:
```
ma_fonction(x[3], y[1]) == res[1, 3]
```

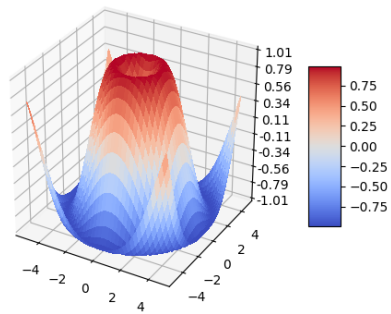
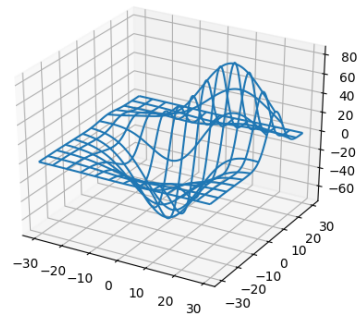
Out[8]:
```
True
```

Représentation graphique de tableaux 2D

Il existe plein de façon se représenter un tableau 2D, en particulier par des surfaces 3D :
`https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html`



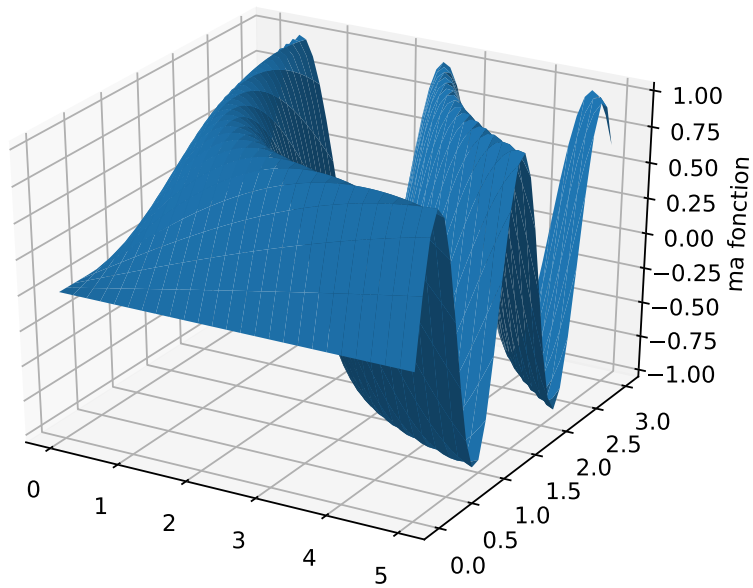


ou bien à l'aide d'échelles de couleurs.

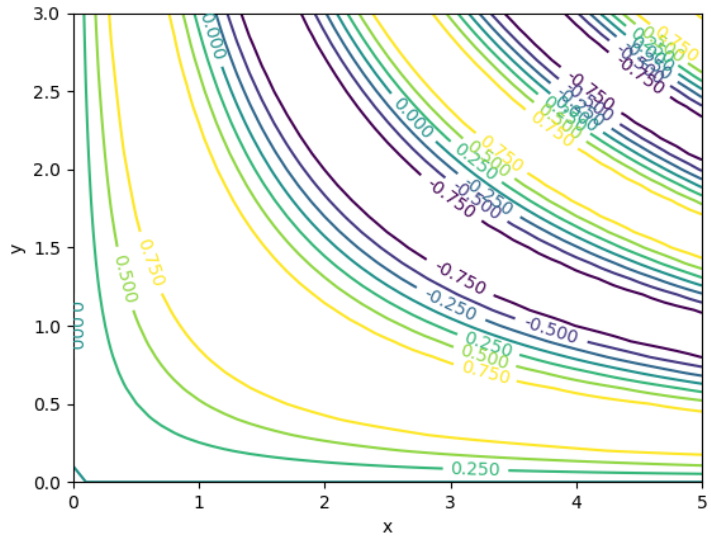
Surfaces 3D

```
1  # charge les librairies
2  import matplotlib.pyplot as plt
3  from mpl_toolkits.mplot3d import Axes3D #  3D
4  import numpy as np
5
6  # initialise les donnees
7  x = np.linspace(0,5,51)
8  y = np.linspace(0,3,31)
9  xx, yy = np.meshgrid(x,y)
10 z = ma_fonction(xx,yy)
11
12 # trace le graphique
13 fig = plt.figure()
14 ax=plt.axes(projection='3d')  # permet la 3D
15 surf = ax.plot_surface(xx, yy, z)
16 # fixe les limites de l'axe z
17 ax.set_zlim(-1.01, 1.01)
18 # titre de l'axe z
19 ax.set_zlabel('ma fonction')
20 plt.show()
```



Lignes de niveaux (contours) 2D

```
1  # charge les librairies
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # initialise les donnees
6  x = np.linspace(0,5,51)
7  y = np.linspace(0,3,31)
8  xx, yy = np.meshgrid(x,y)
9  z = ma_fonction(xx,yy)
10
11 # trace le graphique
12 fig, ax = plt.subplots()
13 CS = ax.contour(xx, yy, z)
14 ax.clabel(CS, inline=1, fontsize=10)
15 ax.set_xlabel('x')
16 ax.set_ylabel('y')
17 plt.show()
```



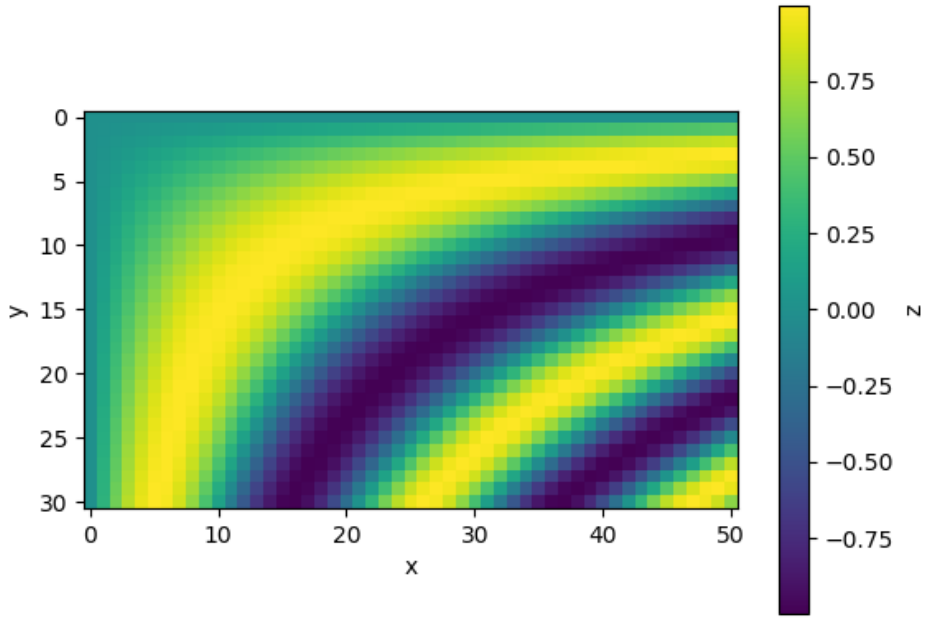
Le nombre de niveaux ou directement les niveaux des contours peuvent être préciser manuellement avec l'argument `levels` :

- `ax.contour(xx, yy, z, levels=6)` impose d'avoir 6 contours ;
- `ax.contour(xx, yy, z, levels=[-1,0,1])` impose les contours $-1, 0, 1$.

Carte colorée

`imshow` est adapté pour représenter des images ou des tableaux 2D.

```
1  # charge les librairies
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # initialise les donnees
6  x = np.linspace(0,5,51)
7  y = np.linspace(0,3,31)
8  xx, yy = np.meshgrid(x,y)
9  z = ma_fonction(xx,yy)
10
11 # trace le graphique
12 fig = plt.figure()
13 im = plt.imshow(z)
14 plt.xlabel('x')
15 plt.ylabel('y')
16 # creation barre de couleur
17 c = fig.colorbar(im)
18 c.set_label('z')
19 plt.show()
```



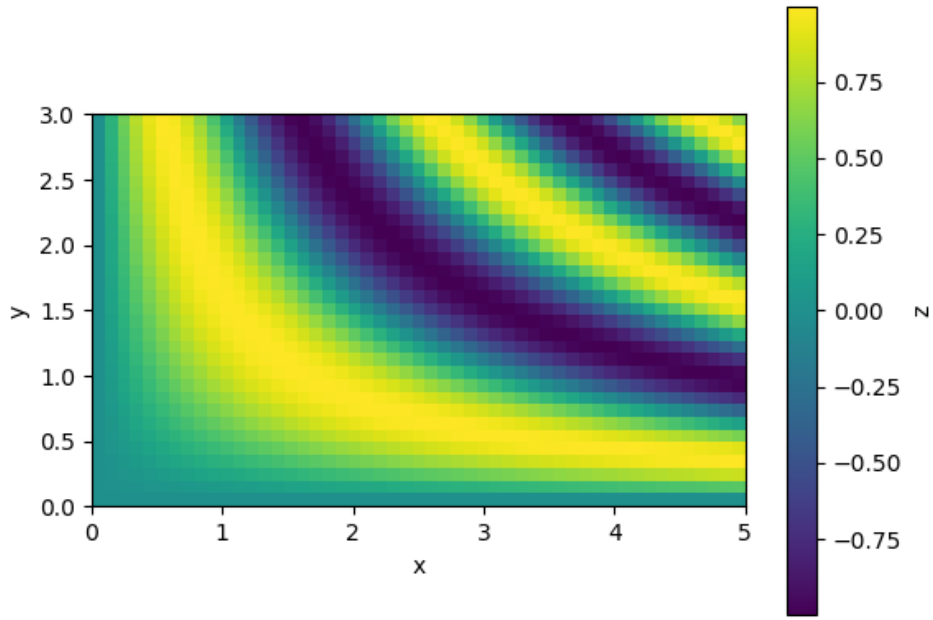
On remarque deux problèmes :

1. l'axe des y est inversé par rapport au sens habituel ;
2. les axes sont numérotés en pixels et non avec les valeurs de x et y.

Carte colorée

`imshow` est adapté pour représenter des images ou des tableaux 2D.

```
1  # charge les librairies
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # initialise les donnees
6  x = np.linspace(0,5,51)
7  y = np.linspace(0,3,31)
8  xx, yy = np.meshgrid(x,y)
9  z = ma_fonction(xx,yy)
10
11 # trace le graphique
12 fig = plt.figure()
13 im = plt.imshow(z, origin='lower',
14                 extent=[0,5,0,3])
15 plt.xlabel('x')
16 plt.ylabel('y')
17 c = fig.colorbar(im)
18 c.set_label('z')
19 plt.show()
```



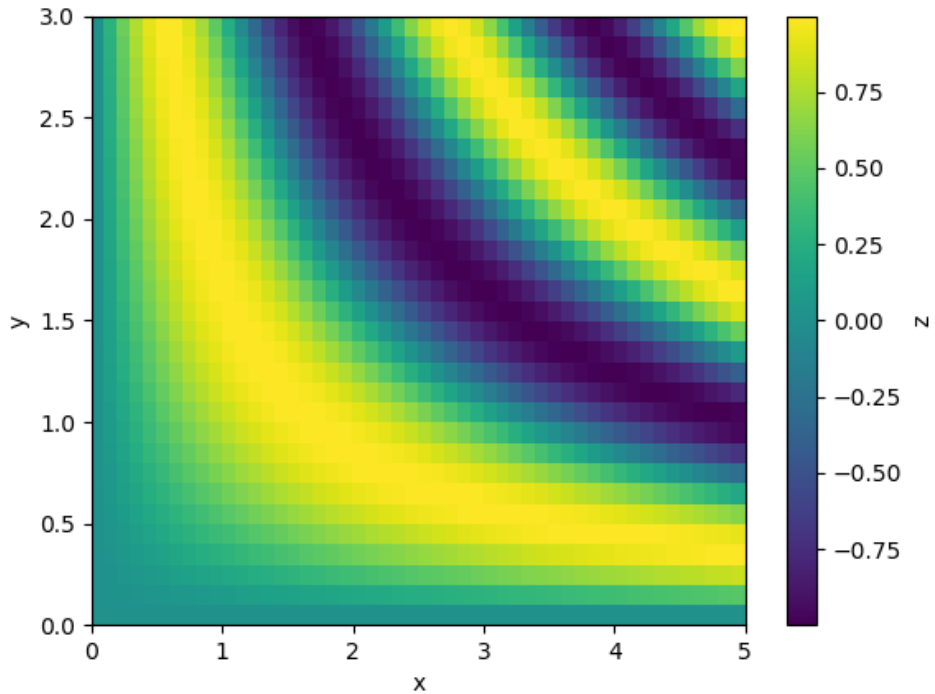
Deux mots clés sont importants avec `imshow` :

1. `origin='lower'` permet de mettre l'origine 0 en bas ;
2. `extent` donne les valeurs min et max des axes x et y entre lesquelles numéroter les axes.

Carte colorée

`pcolor` est plus adapté pour représenter $z = f(x, y)$.

```
1  # charge les librairies
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # initialise les donnees
6  x = np.linspace(0,5,51)
7  y = np.linspace(0,3,31)
8  xx, yy = np.meshgrid(x,y)
9  z = ma_fonction(xx,yy)
10
11 # trace le graphique
12 fig = plt.figure()
13 im = plt.pcolor(x,y,z,shading='auto')
14 plt.xlabel('x')
15 plt.ylabel('y')
16 # creation barre de couleur
17 c = fig.colorbar(im)
18 c.set_label('z')
19 plt.show()
```

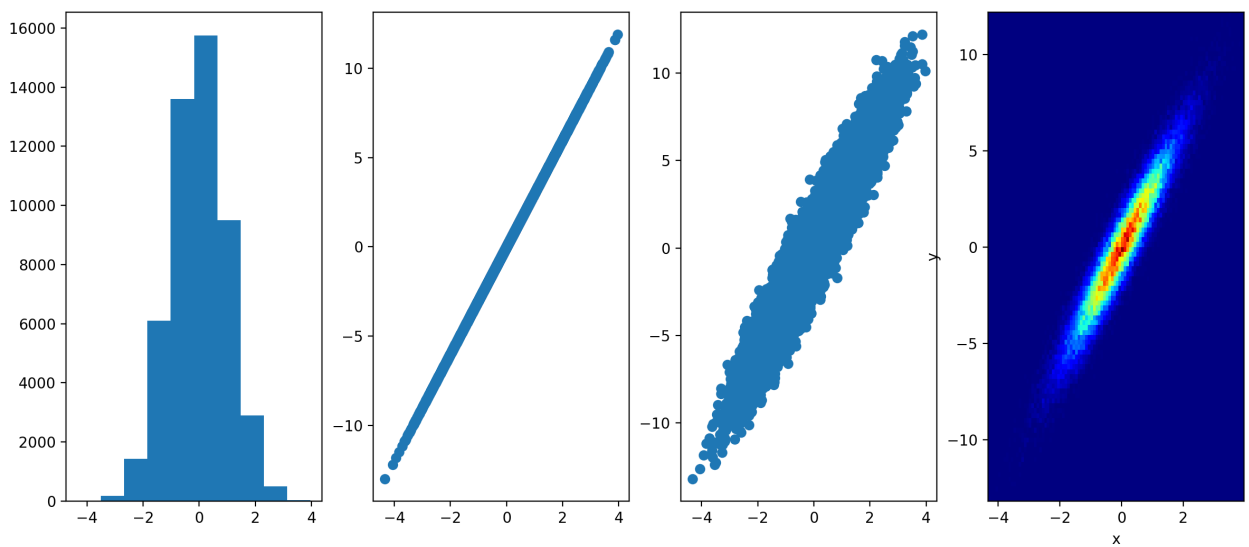


Histogrammes 2D

L'objectif est de tracer un histogramme 2D de réalisations aléatoires de points (x, y).

```
# initialise les donnees
x = np.random.normal(size=50000)
y = x * 3 + np.random.normal(size=50000)

# trace le graphique
fig,ax = plt.subplots(1,4,figsize=(15,6))
ax[0].hist(x)
ax[1].plot(x,3*x,'o')
ax[2].plot(x,y,'o')
ax[3].hist2d(x, y, bins=(100, 100), cmap=plt.cm.jet)
ax[3].set_xlabel('x')
ax[3].set_ylabel('y')
plt.show()
```



À vos TPs !

1. Ouvrir un terminal :
   - soit sur https ://jupyterhub.ijclab.in2p3.fr
   - soit sur un ordinateur du 336
2. Télécharger la séance d'aujourd'hui :

```
methnum fetch L2/Seance3 TONGROUPE
```

   en remplaçant TONGROUPE par ton numéro de groupe.
3. Sur un ordinateur du bâtiment 336 uniquement, lancer le jupyter :

```
methnum jupyter notebook
```

4. Pour soumettre la séance, dans le terminal taper :

```
methnum submit L2/Seance3 TONGROUPE
```

À vos TPs !

Le cours est disponible en ligne ici :
`https://methnum.gitlab.dsi.universite-paris-saclay.fr/L2/`.

Rappel : votre gitlab personnel sert de sauvegarde pour passer vos documents d'une plateforme à l'autre via les commandes methnum/fetch.