L2-S3 : UE Méthodes numériques

SEANCE 4

Fonctions à 2 variables; Représentation graphique, Intégration, Ajustement

28 septembre 2024

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

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

A 2 nombres en entrée correspond 1 nombre en sortie.

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

In [1]:
```
1 def ma_somme(x,y):
2     return x + 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):
            # fonction sin de numpy (boucle
      implicite si x et y sont des tableaux)
  return np.sin(x*y)
```

mais si `x` et `y` sont des tableaux 1D, alors le résultat sera 1D aussi... (et pas 2D)

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

Out[3]:
```
array([5 , 7, 9])
```

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

La fonction meshgrid

In [4]:
```
1 x = np.linspace(0, 2, 5)
2 y = np.linspace(0, 20, 3)
3 print("x =",x)
4 print("y =",y)
```

Out[4]:
```
x = [0.  0.5 1.  1.5 2. ]
y = [0.  10. 20. ]
```

In [5]:
```
1 ma_somme(x,y)
```

Out[5]:
```
ValueError
    Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11076
    /2409487626.py in <module>
----> 1 ma_somme(x,y)
~\AppData\Local\Temp/ipykernel_11076
    /4291362385.py in ma_somme(x, y)
      1 def ma_somme(x,y):
----> 2     return x + y
ValueError: operands could not be broadcast
    together with shapes (5,) (3,)
```

L'exécution ne fonctionne pas car x et y n'ont pas la même taille

La fonction meshgrid

On veut en fait calculer la valeur de f(x,y) **pour toutes les paires (x,y) possibles**. On veut calculer f(M) pour tous les points M(x,y).
Ceci est réalisé par la fonction `numpy.meshgrid()`
A partir de **deux tableaux 1D** contenant respectivement des coordonnées `x` et `y`, produit **deux tableaux 2D** contenant respectivement la coordonnée x et la coordonnée y pour chacune des paires de valeurs formées à partir des deux tableaux 1D initiaux

In [6]:

```
1 xx, yy = np.meshgrid(x, y)
2 print("xx =\n",xx)  # la coordonnee x (axe
      horizontal, "numero de colonne")
3 print("yy =\n",yy)  # correspond au second
      indice des tableaux xx et yy, et
4                     #la coordonnee y correspond au
      premier indice ("numero de
5                     #ligne": conforme a la
      convention habituelle des matrices)
```

La fonction meshgrid

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.]
 [10. 10. 10. 10. 10.]
 [20. 20. 20. 20. 20.]]
```

La fonction meshgrid

Le résultat de `meshgrid` contient 2 grilles 2D des coordonnées `x` et `y`, chacune de dimention N_y x N_x. La fonction va s'appliquer à toutes les paires de points des tableaux 2D `xx` et `yy` :

In [7]:
```
res = ma_somme(xx,yy)
print("x+y=",res)
```

Out[7]:
```
x+y=
 [[ 0.   0.5  1.   1.5  2. ]
 [10.  10.5 11.  11.5 12. ]
 [20.  20.5 21.  21.5 22. ]]
```

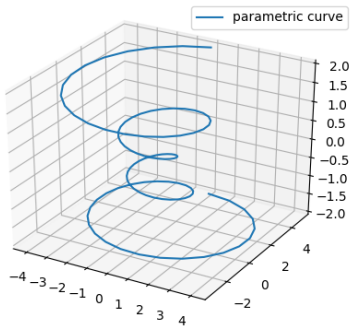
Attention à l'ordre des indices (tableaux xx, yy et res) :
premier indice = numéro de ligne : **second** tableau 1D en argument de meshgrid (ici : y, axe vertical)
second indice = numéro de colonne : **premier** tableau 1D en argument de meshgrid (ici : x, axe horizontal)

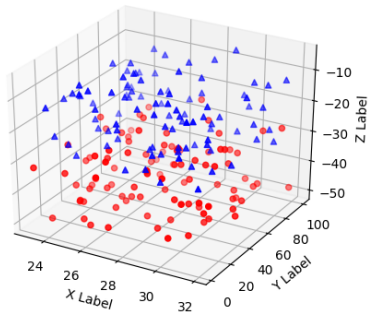
In [8]:
```
ma_somme(x[3], y[1]) == res[1, 3]
```

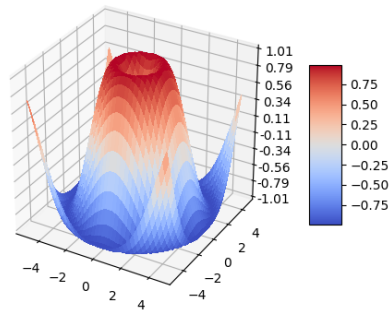
Out[8]:
```
True
```

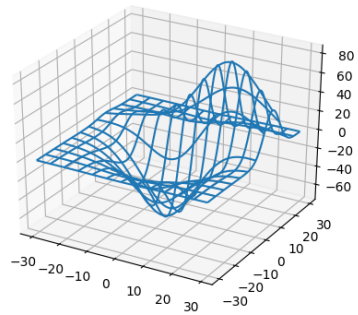
Représentation graphique de tableaux 2D

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





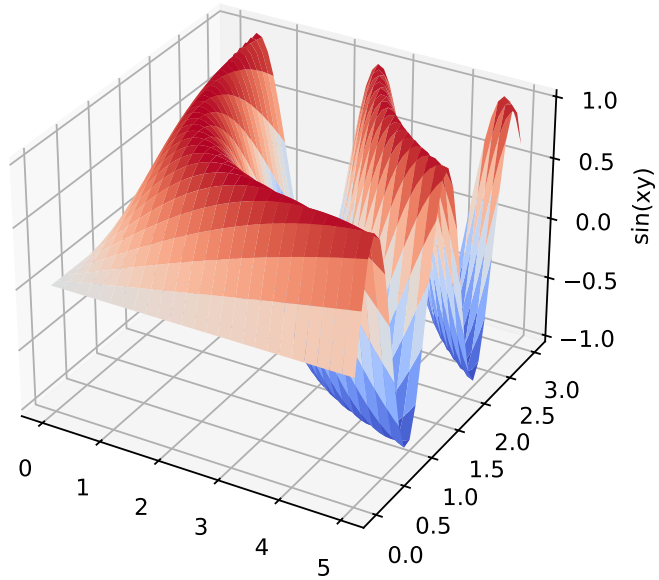




ou bien à l'aide d'échelles de couleurs.

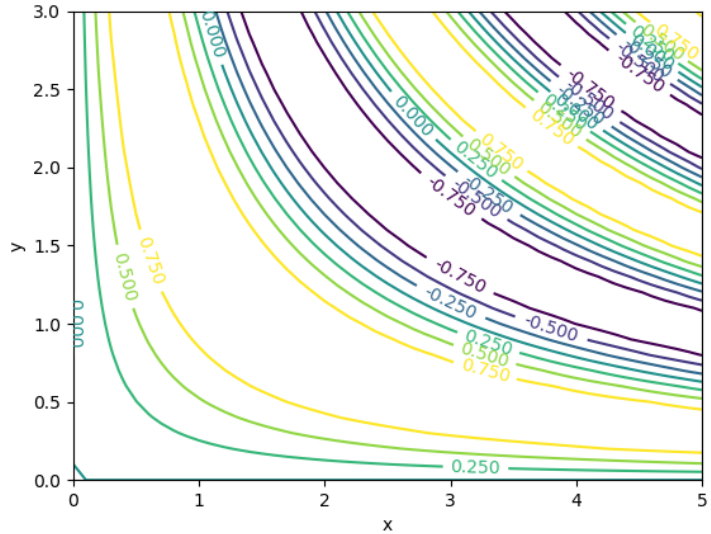
Surfaces 3D

```python
# charge les librairies
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D #  3D
import numpy as np

# initialise les donnees
x = np.linspace(0,5,51)
y = np.linspace(0,3,31)
xx, yy = np.meshgrid(x,y)
z = ma_fonction(xx,yy)  # =np.sin(x*y)

# trace le graphique
fig = plt.figure()
ax=plt.axes(projection='3d')  # cree un objet ax permettant
      # la representation 3D...grace a la methode plot_surface
surf = ax.plot_surface(xx, yy, z, cmap = plt.cm.coolwarm)
ax.set_zlim(-1.01, 1.01)  # fixe les limites de l'axe z
ax.set_zlabel('sin(xy)')  # titre de l'axe z
plt.show()
```



Lignes de niveaux (contours) 2D

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



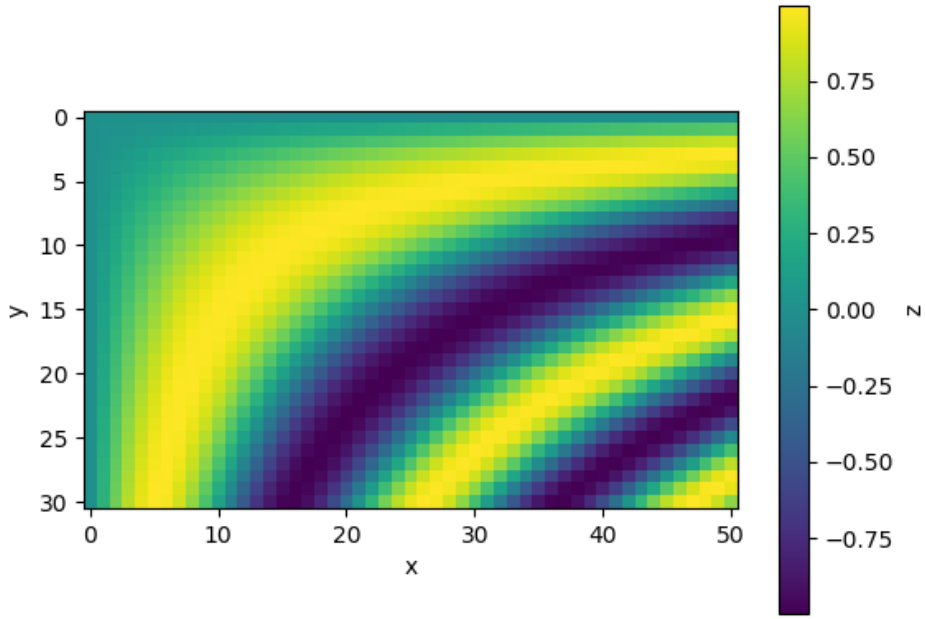
Le nombre de niveaux ou directement les niveaux des contours peuvent être précisés manuellement avec l'argument `levels` :

- `plt.contour(xx, yy, z, levels=6)` impose d'avoir 6 contours ;
- `plt.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 de barre de couleur
17 c=fig.colorbar(im)
18 c.set_label('z')
19 plt.show()
```



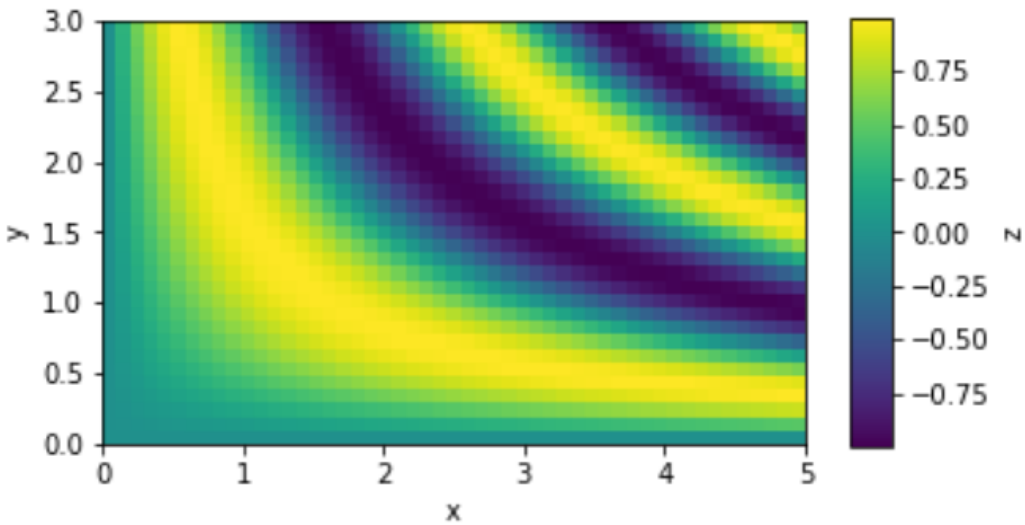
On remarque deux problèmes :

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

Carte colorée

`imshow` est adapté pour representer 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', extent
       =[0,5,0,3])
14 plt.xlabel('x')
15 plt.ylabel('y')
16 c = fig.colorbar(im,shrink=0.75, aspect=10) #
       shrink: hauteur; aspect: hauteur/largeu
17 c.set_label('z')
18 plt.show()
```



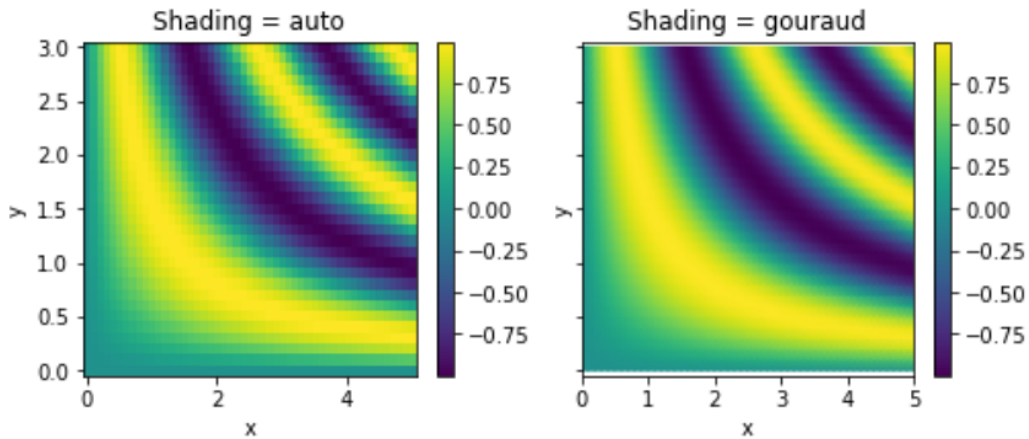
Deux mots clés sont importants avec `imshow` :

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

Carte colorée

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

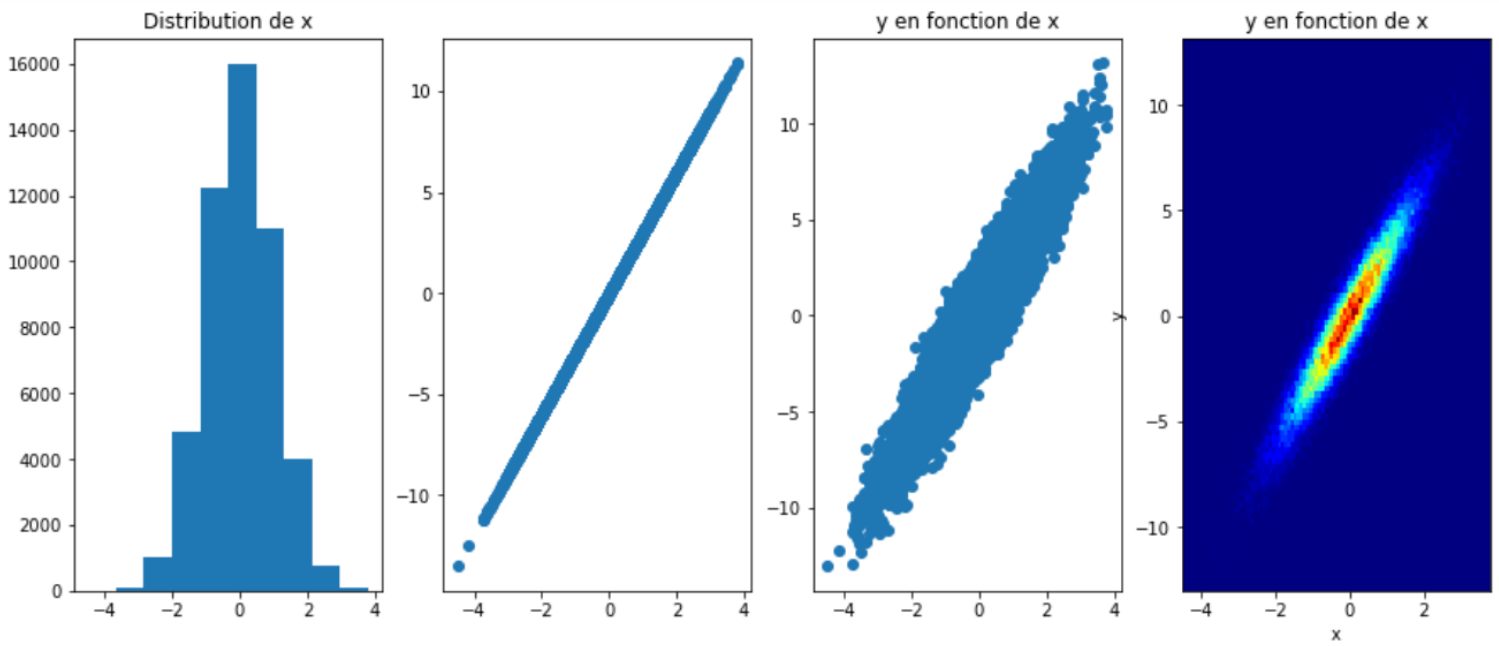
```
1  # charge les librairies
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # initialise les donnees
6  x = np.linspace(0,5,51)
7  y = np.linspace(0,3,31)
8  xx, yy = np.meshgrid(x,y)
9  z = ma_fonction(xx,yy)
10
11 # trace le graphique
12 fig, axs = plt.subplots(1, 2, figsize=(8,3),
       sharey=True)  # meme axe y
13 im = axs[0].pcolormesh(x,y,z,shading='auto')
       # chaque pixel de im a une valeur donnee
14 plt.colorbar(im, ax=axs[0])
15 im = axs[1].pcolormesh(x,y,z,shading='gouraud'
       )  # interpolation entre pixels de im
16 # Henri Gouraud, informaticien francais ne en
       1944
17 plt.colorbar(im, ax=axs[1])
18 axs[0].set_xlabel('x')
19 axs[0].set_ylabel('y')
20 axs[0].set_title("Shading = auto")
21 axs[1].set_xlabel('x')
22 axs[1].set_ylabel('y')
23 axs[1].set_title("Shading = gouraud")
24 plt.show()
```



Histogrammes 2D

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)

fig,ax = plt.subplots(1,4,figsize=(15,6))
ax[0].hist(x)
ax[0].set_title("Distribution de x")
ax[1].plot(x,3*x,'o')
ax[2].plot(x,y,'o')
ax[2].set_title("y en fonction de x")
ax[3].hist2d(x, y, bins=(100, 100), cmap=plt.cm.jet)
ax[3].set_xlabel('x')
ax[3].set_ylabel('y')
ax[3].set_title("y en fonction de x")
plt.show()
```



Intégration d'une fonction de deux variables

Fonction *dblquad* du module *scipy.integrate*
Elle permet de calculer des intégrales du type :

$$\int_{x=a}^{x=b} \left(\int_{y=g(x)}^{y=h(x)} f(y,x)\mathrm{d}y \right) \mathrm{d}x = \int_{x=a}^{x=b} \int_{y=g(x)}^{y=h(x)} f(y,x)\mathrm{d}x\mathrm{d}y,$$

où les bornes du domaine suivant y peuvent dépendre de x.
Attention : le premier argument de f doit être y, puis x.

Intégration d'une fonction de deux variables

Exemple avec :

$$\int_0^{\pi} \mathrm{d}x \int_0^{2x} \mathrm{d}y \sin(xy)$$

In [9]:

```
from scipy.integrate import dblquad
import numpy as np

#bornes suivant x
ax, bx = 0, np.pi

#bornes suivant y
h = lambda x : 0
g = lambda x : 2*x

ff = lambda y, x: np.sin(x*y)
print(dblquad(ff, ax, bx, h, g))
```

Out [9]:

```
(1.7611276682757178, 5.967868259385284e-09)
```

Ajustement par une fonction de deux variables

On souhaite ajuster une fonction $z = f(x, y)$ sur des données expérimentales z_i avec leurs incertitudes σ_i dépendant de deux variables x et y (par exemple ajuster une forme sur une image).

Tout ce que nous avons vu à la séance 3 se généralise au cas d'un modèle $z = f(x, y)$ quelconque :

- notre modèle est de la forme : $z = f(x, y, \beta_0, \beta_1...)$, où $\beta_0, \beta_1...$ sont des paramètres à ajuster ;
- on cherche les valeurs des paramètres $\beta_0, \beta_1...$ qui minimisent la somme des écarts entre points expérimentaux et valeurs données par le modèle $z = f(x, y)$;
- on pondère la contribution des points par un coefficient $w_i = 1/\sigma_i^2$.

On cherche donc les valeurs des paramètres $\beta_0, \beta_1...$ qui minimisent :

$$\chi^2 = \sum_i \frac{(z_i - f(x_i, y_i, \beta_0, \beta_1...))^2}{\sigma_i^2}$$

Ajustement par une fonction de deux variables

Rappel : utilisation de la fonction `curve_fit` dans le cas 1D

```
params,
cov=curve_fit(fit_func,x,y,p0=params0,sigma=sigma_y)
```

Paramètres d'entrée :

- `fit_func` est le nom de la fonction servant de modèle. Elle est définie avant dans le programme à l'aide d'un `def`
- `x,y` sont des tableaux `np.array` contenant les valeurs expérimentales x_i et y_i
- `sigma_y` est un tableau `np.array` contenant les valeurs des incertitudes σ_y sur les valeurs y_i
- `params0` est un tableau contenant les valeurs initiales des paramètres du modèle pour amorcer l'algorithme

Paramètres de sortie :

- `params` est un tableau 1D contenant les résultats de la minimisation des moindres carrés : ici `params[i]` $= \beta_i$
- `cov` est un tableau 2D. C'est la matrice de covariance. **Ses termes diagonaux contiennent les incertitudes sur les paramètres de la régression :** `cov[i,i]` $= \sigma^2_{\beta_i}$

Ajustement par une fonction de deux variables

Supposons que nous ayons un ensemble de valeurs expérimentales z_i mesurées en fonction de deux paramètres x_i et y_i :

- Les valeurs de x_i sont dans un tableau 1D à N_x valeurs
- Les valeurs de y_i sont dans un tableau 1D à N_y valeurs
- Les valeurs de z_i sont dans un tableau 2D à N_y lignes et N_x colonnes
- on souhaite modéliser ce nuage de points (3D !) par une fonction $z = f(x, y, \beta_0, \beta_1, \cdots)$.
- la fonction `curve_fit` peut être utilisée, moyennant une mise en forme des données d'entrée : en argument de la fonction, on ne peut fournir qu'un tableau unique pour les valeurs des variables (ainsi que les paramètres de l'ajustement).

Voici quelques fonctions utiles pour "mettre en forme les tableaux de données" avant leur utilisation avec `curve_fit`.

- `X.reshape(Ny,Nx)` met un tableau 1D de longueur `Ny*Nx` sous la forme d'un tableau 2D.
- `X.ravel()` réorganise les éléments d'un tableau 2D sous la forme d'un tableau 1D.
- `Mxy=np.vstack((x1d,y1d))` forme un tableau 2D à deux lignes et `N` colonnes à partir de deux tableaux 1D de longueur `N`.

Ajustement par une fonction de deux variables

```python
# Definition de la fonction modele
# Necessite de combiner x et y en un seul argument d'entree Mxy
def fit_func(Mxy,param1,param2):
    ''' Mxy: tableau 2D (2 lignes, N colonnes) : 2N valeurs '''
    x=Mxy[0]
    y=Mxy[1]
    z=param1*x**2-param2*y**2
    return z

# x: tableau des x_i; y: tableau des y_i
XX, YY = np.meshgrid(x,y) # coord des points ou on mesure z
xdata=XX.ravel() # on transforme les 2 tableaux 2D XX et YY en
ydata=YY.ravel() # 2 tableaux 1D de meme longueur Nx*Ny
Mxy=np.vstack((xdata,ydata)) # avec lesquels on forme un tableau (2,Nx*Ny)
```

```python
# zdata est un tableau 1D de meme longueur, obtenu a partir de ZZ,
#  qui est a comparer a f(XX,YY, parametres)
zdata = ZZ.ravel() # on transforme les tableaux 2D des valeurs mesurees ZZ
sigma_zdata = sigma_ZZ.ravel() # et des incertitudes en deux tableaux 1D
                               # a Nx*Ny valeurs chacun

initialisation=[param1_initial,param2_initial]

# dans curve_fit, le tableau contenant les variables x et y est Mxy
# et le tableau contenant les donnees est zdata. Tout le reste est pareil !
params, cov=curve_fit(fit_func,Mxy,zdata,p0=initialisation,sigma=sigma_zdata)
```