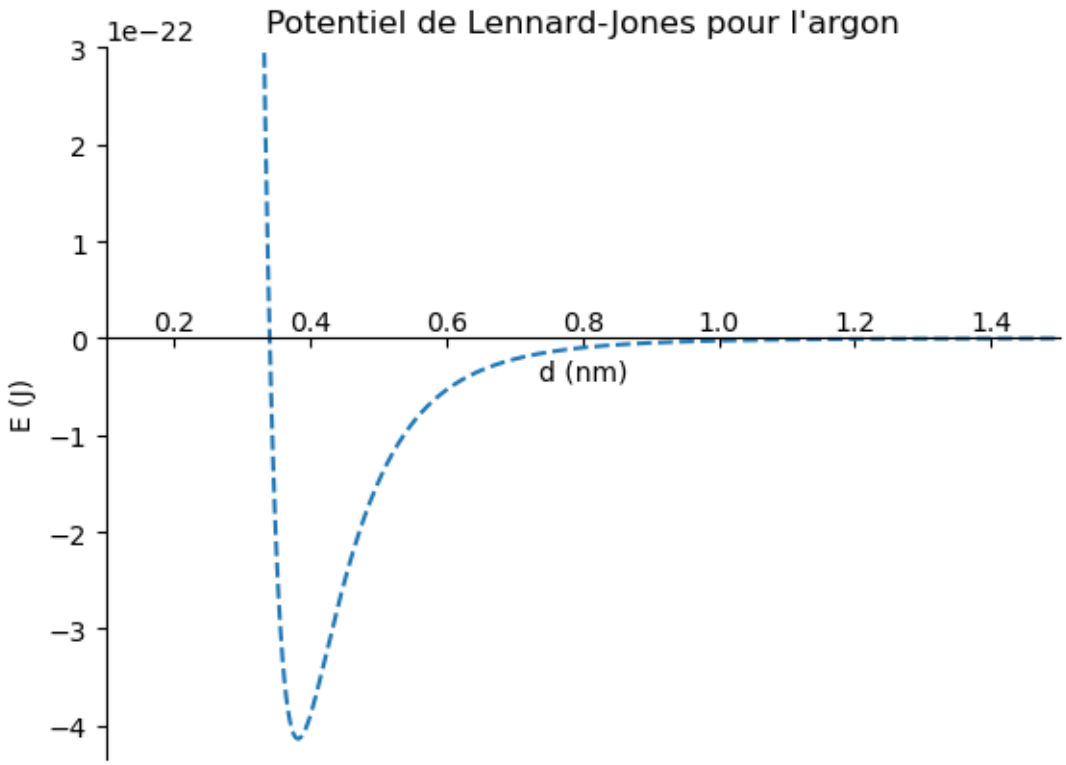
exo5-solution

October 25, 2023

1 tracé d'une fonction : Potentiel de Lennard-Jones

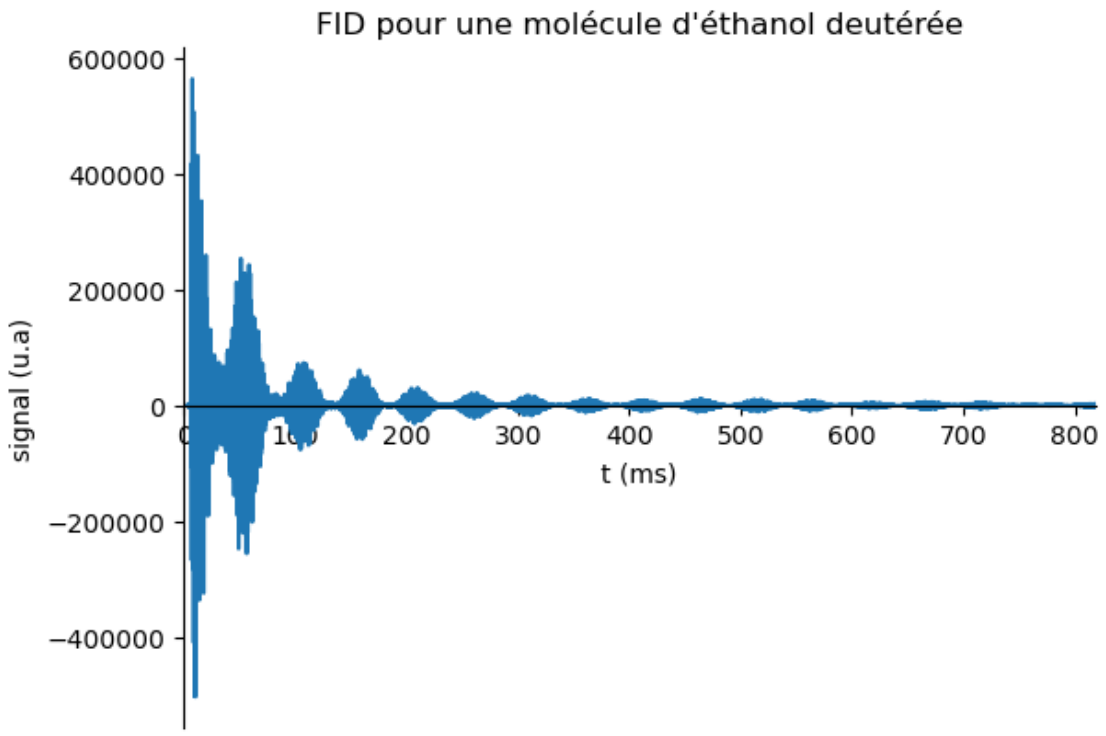
[59]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Tracé d'une fonction analytique
"""

# Importation des librairies

import numpy as np
import scipy
import scipy.constants as constants
import matplotlib as mpl
import matplotlib.pyplot as plt

# Definition des fonctions
def Lennard_Jones(d, E0, r):
    """
     E_{LJ}(r) = 4E_0 * ((d/r)**12-(d/r)**6) $$
    d (float) : distance caractéristique du potentiel de Lennard-Jones en nm
    E0 (float) : préfacteur énergétique
    r (float ou ndarray) : distance(s) à laquelle calculer le potentiel de␣
 ↪Lennard-Jones
    retourne
    ndarray (ou float) : valeur du potentiel de Lennard-Jones en J
    """
    return E0 * ((d / r) ** 12 - (d / r) ** 6)


# Programme principal
if __name__ == "__main__":
    dmin = 0.1
    dmax = 1.5
    d = 0.3405
    E0 = 119.8 * constants.k
```

```
    rs = np.linspace(dmin, dmax, 1000)
    fvalues = Lennard_Jones(d, E0, rs)

    fig = plt.figure()
    gs = fig.add_gridspec(1, 1)
    ax1 = fig.add_subplot(gs[0, 0])

    ax1.plot(rs, fvalues, ls="--")
    ax1.set_ylim(1.05 * np.min(fvalues), 3e-22)
    ax1.set_xlim(dmin, dmax)
    ax1.set_title("Potentiel de Lennard-Jones pour l'argon")
    ax1.set_xlabel("d (nm)")
    ax1.set_ylabel("E (J)")
    ax1.spines["bottom"].set_position("zero")
    ax1.spines[["top", "right"]].set_visible(False)
    ax1.spines["left"].set_visible(True)

    # Pour placer les label de l'axe des x au dessus au lieu d'en-dessous
    offset = mpl.transforms.ScaledTranslation(0, 0.25, fig.dpi_scale_trans)
    for label in ax1.xaxis.get_majorticklabels():
        label.set_transform(label.get_transform() + offset)
    plt.savefig("lennard-jones.svg")
    plt.show()
```



2 Tracé à partir d'un fichier

[66]:
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Tracé d'une fonction numérique à partir d'un fichier csv
"""

# Importation des librairies

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

# Definition des fonctions


# Programme principal
if __name__ == "__main__":
    data = np.genfromtxt(
        "exo5-fid.csv", delimiter=";", names=["t", "signal"], skip_header=1
    )
    fig = plt.figure()
    gs = fig.add_gridspec(1, 1)
    ax1 = fig.add_subplot(gs[0, 0])

    ax1.plot(data["t"], data["signal"])
    # sans l'option names :
    """
    data = np.genfromtxt('exo5-fid.csv',delimiter=';',skip_header=1)
    ax1.plot(data[:,0],data[:,1])
    """

    ax1.set_title("FID pour une molécule d'éthanol deutérée")
    ax1.set_xlabel("t (ms)")
    ax1.set_ylabel("signal (u.a)")
    ax1.set_xlim(0, np.max(data["t"]))
    ax1.spines["bottom"].set_position("zero")
    ax1.spines[["top", "right"]].set_visible(False)
    ax1.spines["left"].set_visible(True)

    plt.savefig("fid.png")
    plt.show()
```

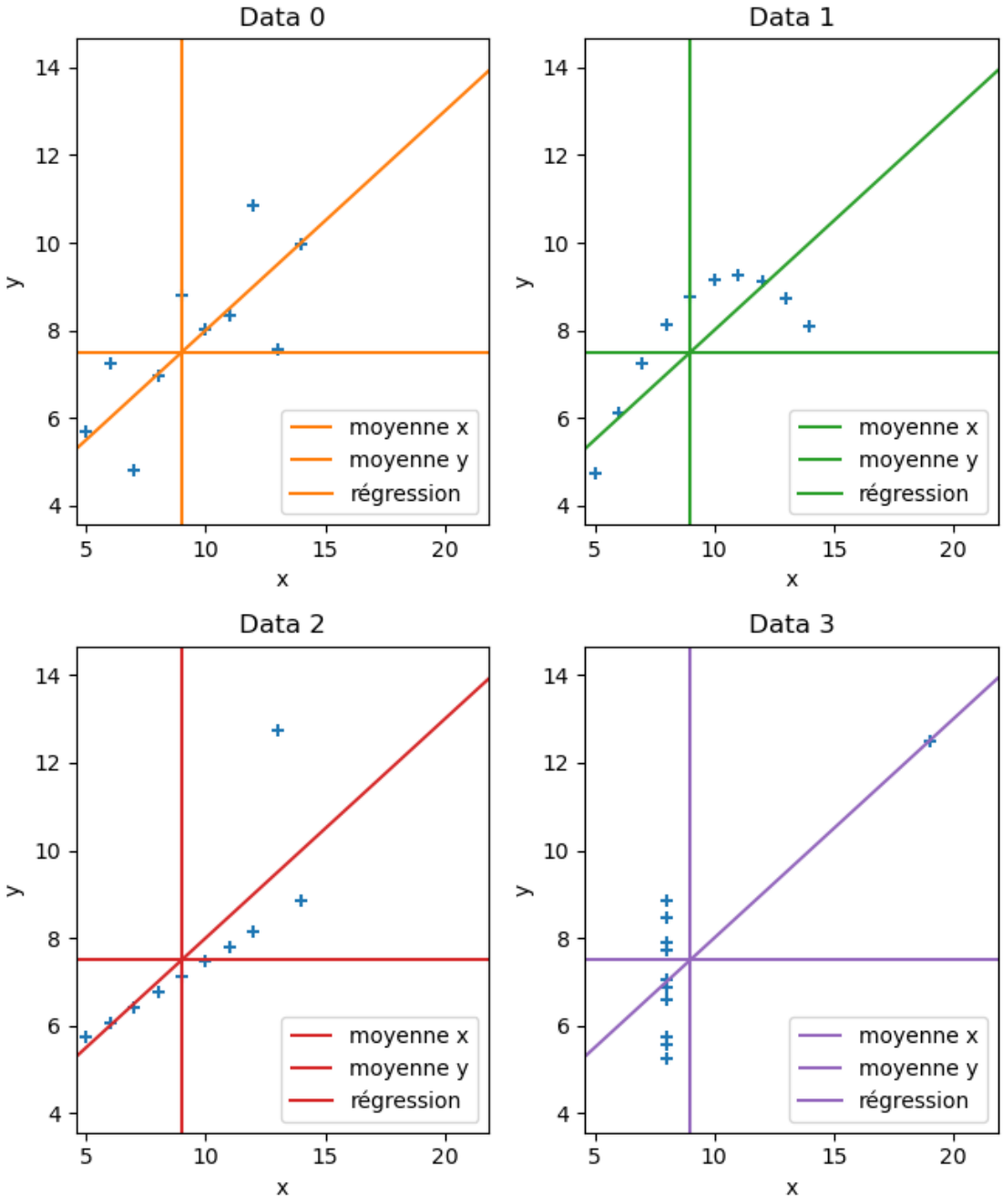


3 Graphiques multiples

[29]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Graphique multiples
"""

# Importation des librairies

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

# Definition des fonctions


# Programme principal
if __name__ == "__main__":
    cycle = plt.rcParams["axes.prop_cycle"].by_key()["color"]
```

```
    data2 = np.genfromtxt("exo5-anscombe.dat", delimiter="\t", skip_header=1)
    fig = plt.figure(figsize=(10, 8))
    gs = fig.add_gridspec(2, 3)
    for i in range(4):
        # on joue sur la quotient et le modulo pour automatiser le␣
↪positionnement des graphiques
        axi = fig.add_subplot(gs[i // 2, i % 2])
        # nuage de points sur les colonnes d'indice 2i et 2i+1 qui contiennent␣
↪x et y pour chaque jeu de données.
        axi.scatter(data2[:, 2 * i], data2[:, 2 * i + 1], marker="+")

        # bornes du graphique
        xmin, xmax = np.min(data2[:, 0::2]) * 1.15, np.max(data2[:, 0::2]) * 1.
↪15
        ymin, ymax = np.min(data2[:, 1::2]) * 1.15, np.max(data2[:, 1::2]) * 1.
↪15

        # tracé de la moyenne en x
        axi.vlines(
            np.mean(data2[:, 2 * i]), ymin, ymax, color=cycle[i + 1],␣
↪label="moyenne x"
        )
        # tracé de la moyenne en y
        axi.hlines(
            np.mean(data2[:, 2 * i + 1]),
            xmin,
            xmax,
            color=cycle[i + 1],
            label="moyenne y",
        )

        # détermination de la régression linéaire
        coeffs = np.polyfit(data2[:, 2 * i], data2[:, 2 * i + 1], 1)
        xs = np.linspace(xmin, xmax, 2)
        # calcul des valeurs de y pour les points extrémaux
        ys = np.polyval(coeffs, xs)
        # tracé de la régression
        axi.plot(xs, ys, label="régression", color=cycle[i + 1])

        # ajout de la légende
        axi.legend(loc="lower right")

        # titre et légende des axes
        axi.set_title("Data {}".format(i))
        axi.set_xlabel("x")
        axi.set_ylabel("y")
        # Même abscisse et ordonnée pour tout le monde
```

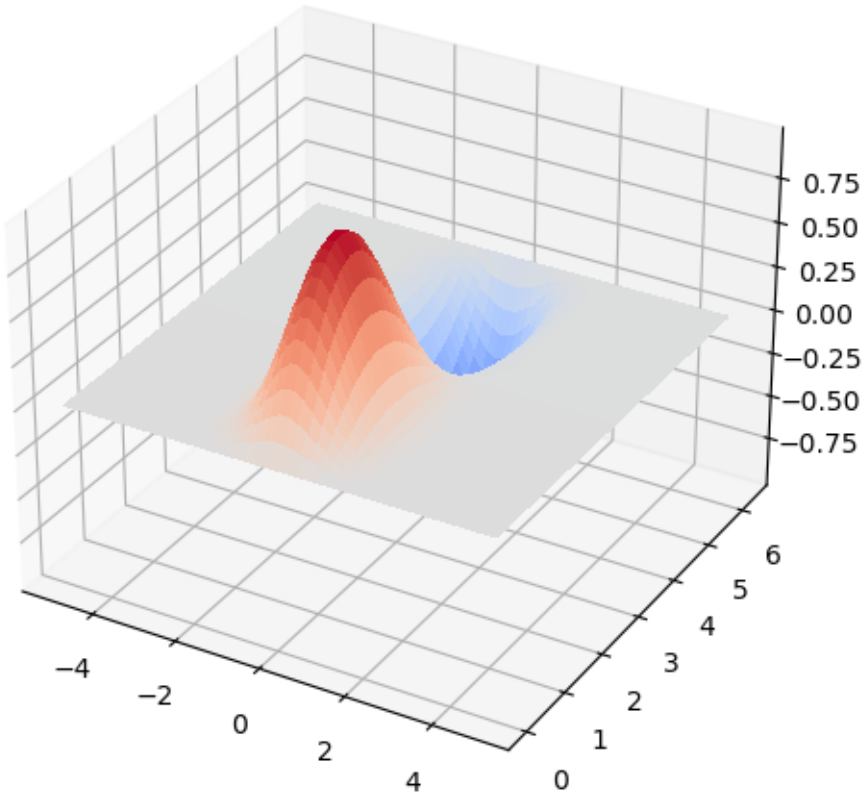
```
        axi.set_xlim(np.min(data2[:, 0::2]) * 1.15, np.max(data2[:, 0::2]) * 1.
↪15)
        axi.set_ylim(np.min(data2[:, 1::2]) * 1.15, np.max(data2[:, 1::2]) * 1.
↪15)

    plt.tight_layout()
    plt.savefig("anscombe.png")
    plt.show()
```

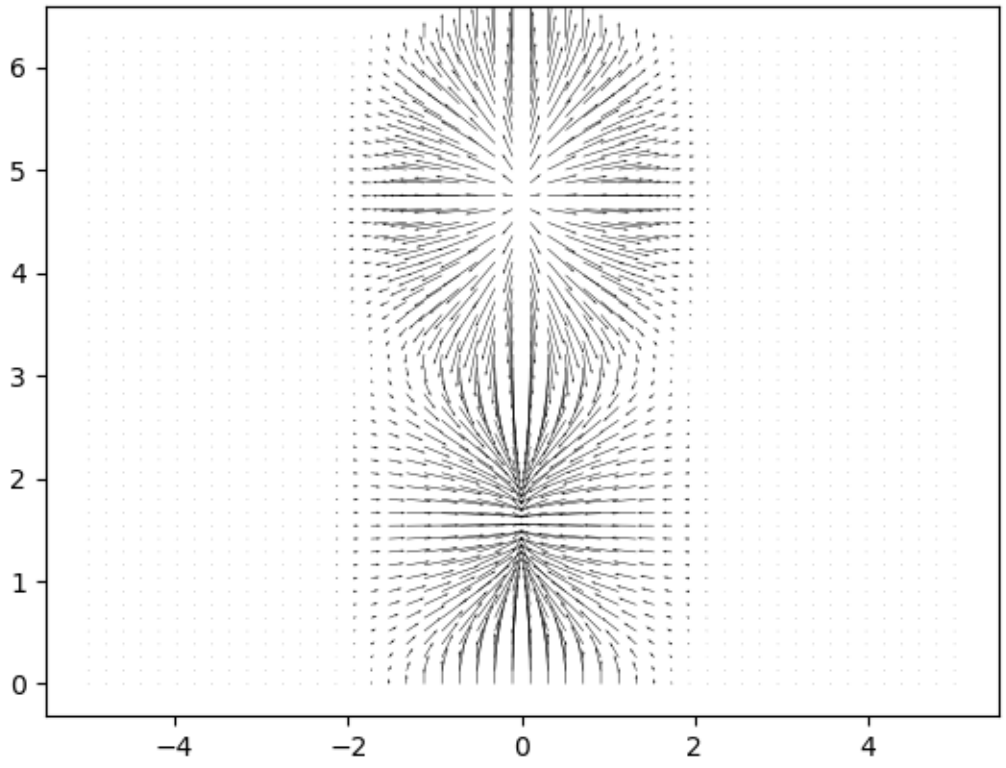


4 Graphiques en trois dimensions

[35]:
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Descriptif du fichier
"""

# Importation des librairies
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# Definition des fonctions
def createGrid(xmin, xmax, nx, ymin, ymax, ny):
    """
    Creer une grille uniforme à deux dimensions

      xmin valeur minimale de x
      xmax valeur maximale de x
      nx échantillonage entre xmin et xmax

      idem pour ymin, ymax, ny
    """
    x, dx = np.linspace(xmin, xmax, nx, retstep=True)
    y, dy = np.linspace(ymin, ymax, ny, retstep=True)
    return np.meshgrid(x, y, indexing="ij"), (dx, dy)


def knownFunc(x, y):
    """
    fonction f(x,y)= exp(-x^2)*sin(y)
    x,y arrays retourné par meshgrid
    """
    return np.exp(-(x**2)) * np.sin(y)

# Programme principal
if __name__ == "__main__":
    # print(createGrid(-5, 5, 50, 0, 2*np.pi, 50) )
    nRows = 3
    nCols = 3
```

```
    grid, delta = createGrid(-5, 5, 50, 0, 2 * np.pi, 50)
    xx, yy = grid
    f = knownFunc(xx, yy)
    grad = np.gradient(f, delta[0], delta[1], edge_order=2)

    fig = plt.figure()
    ax = fig.add_subplot(projection="3d")
    surf = ax.plot_surface(xx, yy, f, cmap=cm.coolwarm, linewidth=0,␣
↪antialiased=False)
    plt.tight_layout()
    plt.show()

    fig = plt.figure()
    surf = plt.quiver(xx, yy, grad[0], grad[1], width=0.0009)
    plt.savefig("func-grad.png")
    plt.show()
```




6
5
4
3
2
1
0
−4
−2
0
2
4

[ ]: