Python pour la physique

Cette fiche présente des méthodes succintes pour réaliser sous python des tâches utiles au physicien. Elle a vocation à être utilisée dans le cadre de votre TIPE notamment.

Les fonctions discutées ci-dessous sont issues des bibliothèques scientifiques standards de python : `numpy`, `scipy`, `matplotlib`,... Des précisions sur chacune des méthodes peuvent être facilement trouvées sur internet.

Les lignes proposées dans cette fiche sont très peu commentées. La raison est simple : il est impossible d'apprendre tout python par coeur. Il faut donc vous habituer dès maintenant à **chercher les informations sur internet**. Dans le même esprit, pour espérer retenir quelques fonctions, il faut les **écrire**. Vous veillerez donc à recopier vous-même les lignes ci-dessous et à ne pas vous contenter d'un copier-coller.

Remarque : Vous pouvez apprendre la syntaxe python en réalisant des projets sur le très bon site HackerRank.com.

Table des matières

1 Définir une fonction

Une fonction peut prendre en argument des nombres, des listes, des tableaux numpy,... et renvoyer des nombres, des listes, des tableaux numpy,...

Exemple 1 : On souhaite définir la fonction $f(t) = t^2\,\mathrm{e}^{-t}$ de $\mathbb{R}$ dans $\mathbb{R}$. On écrit pour cela

```
import numpy as np

def f(t):
    return t**2 * np.exp(-t)
```

Exemple 2 : On souhaite définir la fonction $f(x, y) = x^2 + y^2$ de $\mathbb{R}^2$ dans $\mathbb{R}$. On peut l'écrire de deux manières différentes, en donnant en argument deux nombres, ou un tableau de taille 2 :

```
def f(x, y):
    return x**2 + y**2
```
ou
```
def f(z):
    return z[0]**2 + z[1]**2
```

Exemple 3 : On souhaite définir la fonction $f(x) = \begin{cases} \ln|x| \\ x^2\,\mathrm{e}^{-x} \end{cases}$ de $\mathbb{R}$ dans $\mathbb{R}^2$.

```
import numpy as np

def f(x):
    return [ np.log( np.abs(x) ), x**2 * np.exp(-x)]
```

et on récupère les deux composantes pour $x = 5$ dans les variables a et b par

```
x = 5
a, b = f(x)[0], f(x)[1]
```

2 Tracer une fonction

Python n'est pas un langage de calcul formel. La notion de continuité y est donc totalement absente, et une fonction s'y représente point par point. Il faut donc définir en premier lieu une liste de points pour les abscisses.

Définir les abscisses : on peut utiliser par exemple la fonction `linspace` de `numpy`.

```
import numpy as np

xmin = 0
xmax = 10
nb_pt = 200
x = np.linspace(xmin, xmax, nb_pt)
```

On peut aussi définir une échelle logarithmique par `logspace`.

Tracer la fonction : les lignes suivantes permettent d'obtenir un rendu assez joli. Il faut avoir défini une liste (ou un numpy array) d'abscisses `x` et une liste (ou un numpy array) d'ordonnées `y`. Vous pourrez trouver des explications sur les (nombreuses) commandes sur internet.

```
from matplotlib import pyplot as plt

plt.xlim([0.0, 8.0])                                                          (limites des abscisses pour le tracé)
plt.ylim([-0.05, 0.65])                                                       (idem en ordonnée)
plt.grid()                                                                    (rajoute une grille)
plt.xlabel(r"$t$", fontsize=22)                                               (légende des abscisses t)
plt.ylabel(r"$f(t)$", fontsize=22)                                            (idem en ordonnée f(t))
plt.plot(x, y, color='darkorange', linewidth=2.5, label=r"f")                 (tracé de y vs x)
plt.xticks(fontsize=16)                                                       (taille des marqueurs en abscisse)
plt.yticks(fontsize=16)                                                       (idem en ordonnée)
legend = plt.legend(loc = 1, handlelength=2., frameon=1, fontsize=22)         (légende)
frame = legend.get_frame()
frame.set_facecolor('white')
frame.set_edgecolor('white')
plt.tight_layout()                                                            (découpe la figure aux limites du tracé)
plt.savefig("courbe.pdf")                                                     (sauvegarde la figure en pdf)
plt.show()                                                                    (affiche à l'écran la figure)
```

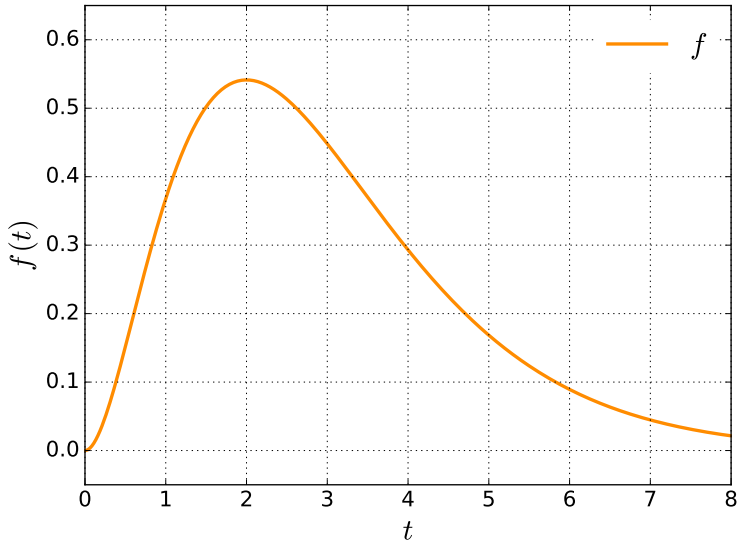

Figure : Courbe représentative de f, tracée avec les commandes ci-dessus.

Tracer une surface : Il y a deux manières de faire :
- soit dans un graphique 3D en traçant $z = f(x, y)$;
- soit en 2D en traçant des lignes de niveaux (appelés en anglais *contour plots*) ou en représentant la fonction en couleur (appelés en anglais *color plots*), par exemple du bleu au rouge ou en niveau de gris.

Vous trouverez facilement les commandes sur internet.

3 Dériver une fonction

On peut dériver (numériquement) une fonction par la méthode `derivative` du paquet `misc`(ellaneous) de `scipy`.

```
import numpy as np
from scipy import misc as scpm

x = np.linspace(0, 10, 200)

def f(t):
    return t**2 * np.exp(-t)

y = scpm.derivative(f,x,dx=1e-6)
```

Veillez à bien définir un `dx` petit, sinon la dérivée obtenue est trop éloignée de la réalité. Veillez également à ne pas le définir trop petit, sous peine d'avoir des erreurs dues aux arrondis.

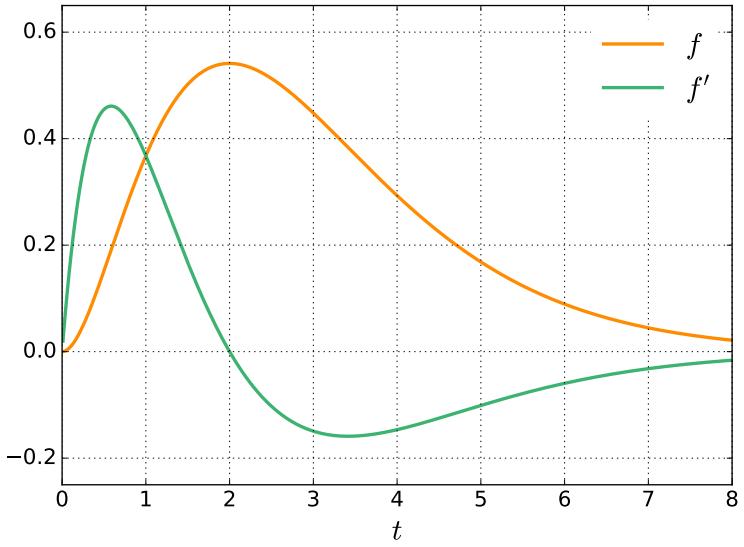

Figure : Courbes représentatives de f et de sa dérivée f', obtenue par les commandes ci-dessus.

Dérivées d'ordre supérieur : elles s'obtiennent par l'option `order` (par défaut égal à 1). La dérivée seconde s'écrit

```
z = scpm.derivative(f,x,dx=1e-6,order=2)
```

4 Intégrer une fonction

On peut intégrer (numériquement) une fonction par la méthode `quad` du paquet `integrate` de `scipy`. Pour calculer l'aire sous la courbe représentative de f entre 0 et 3, on écrit

```
import numpy as np
from scipy import integrate as scpi

def f(t):
    return t**2 * np.exp(-t)

area = scpi.quad(f,0,3)

print(area)
```

5 Intégrer un système d'EDO (et obtenir un portrait de phase)

On peut intégrer une ou plusieurs équations différentielles ordinaires grâce à la méthode `odeint` du paquet `integrate` de `scipy`.

Intégrer une EDO : On cherche à résoudre $y'(t) = -y(t) + 4\cos(5\,t)$ avec $y(0) = 3$. On écrit pour cela

```
import numpy as np
from scipy import integrate as scpi
from matplotlib import pyplot as plt

t = np.linspace(0.0,10.0,200)

def eq(y, u):
    dydt = - y + 4 * np.cos(5 * u)
    return dydt

c_i = 3
y = scpi.odeint(eq,c_i,t)

plt.xlim([0.,10])
plt.ylim([-1.5, 4.5])
plt.grid()
plt.xlabel(r"$t$", fontsize=22)
plt.ylabel(r"$y$", fontsize=22)
plt.plot(t, y, color='mediumseagreen',
                          linewidth=2.5)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.tight_layout()
plt.show()
```

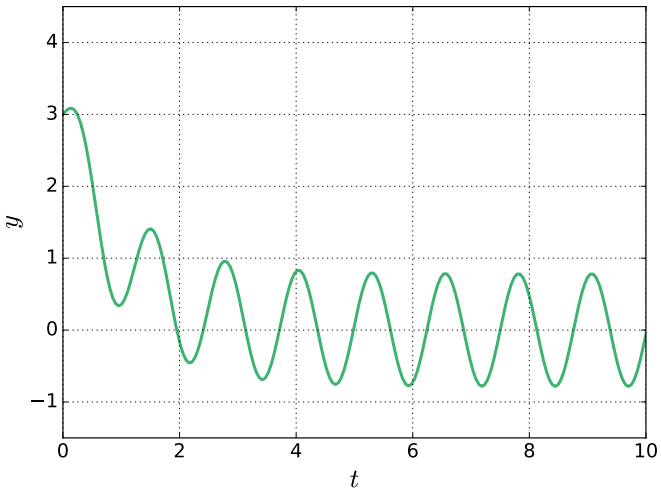

Figure : Représentation graphique de la solution de $y'(t) = -y(t) + 4\cos(5\,t)$ avec $y(0) = 3$.

Intégrer un système d'EDO : On cherche à résoudre $\begin{cases} \dot{x} = y \\ \dot{y} = -\dfrac{1}{3}\,y - x \end{cases}$ (qui correspond à un OH amorti).

On résout avec les conditions initiales $x(0) = 2$ et $y(0) = 0$ puis on trace le portrait de phase $(x, \dot{x})$.

```
import numpy as np
from scipy import integrate as scpi
from matplotlib import pyplot as plt

t = np.linspace(0.0,30.0,200)

def eq(z, u):
    x, y = z[0], z[1]
    dxdt = y
    dydt = - 1/3. * y - x
    return [dxdt, dydt]

c_i = [2,0]
z = scpi.odeint(eq,c_i,t)
x, y = z[:,0], z[:,1]

plt.xlim([-2.5,2.5)
plt.ylim([-2.5, 2.5])
plt.grid()
plt.xlabel(r"$x$", fontsize=22)
plt.ylabel(r"$\dot{x}$", fontsize=22)
plt.plot(x, y, color='mediumseagreen',
                           linewidth=2.5)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.tight_layout()
plt.show()
```

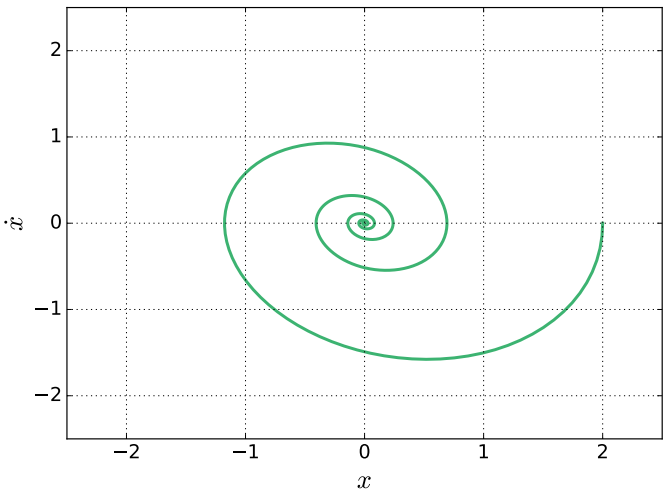

Figure : Représentation graphique du portrait de phase.

6 Ouvrir un fichier pour y récupérer des données

On peut récupérer des données depuis un fichier texte (ici data.dat). Les données doivent être rangées en colonne (x la première colonne et y la deuxième colonne). On utilise pour cela la méthode `loadtxt` du paquet `numpy`.

```
import numpy as np

x, y = np.loadtxt("data.dat", unpack=True)
```

7 Tracer des données (avec des barres d'erreur)

À partir d'un fichier de données (ici data.dat) contenant trois colonnes : les abscisses, les ordonnées et l'erreur sur les ordonnées (z), on peut tracer les mesures faites avec des barres d'erreurs par la méthode `errorbar` de `pyplot`.

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

x, y, z = np.loadtxt("data.dat", unpack=True)

plt.xlim([0., 8.])
plt.ylim([0.,10.])
plt.grid()
plt.xlabel(r"$h$ (m)", fontsize=22)
plt.ylabel(r"$v$ (m/s)", fontsize=22)
plt.errorbar(x,y,yerr=z,color='green',linestyle='None',elinewidth=2.5,marker='o',markersize=6.)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.tight_layout()
plt.show()
```

On donne le résultat du tracé ci-dessous.

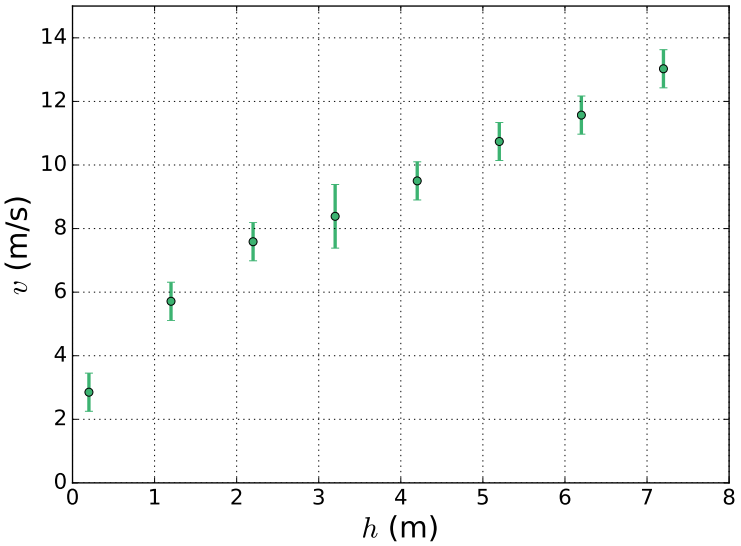

Figure : Représentation graphique de données avec des barres d'erreurs verticales.

Remarque : il est possible d'ajouter des barres d'erreurs horizontales avec l'option `xerr` de `plt.errorbar`.

8 Calculer le meilleur ajustement des données par une fonction

Si l'exemple précédent correspond à des mesures de la vitesse pour une vidange de Torricelli, alors on attend la loi $v = \sqrt{2\,g\,h}$. Pour réaliser un ajustement (un *fit* en anglais) de nos mesures expérimentales, c'est-à-dire trouver la meilleure courbe passant par nos points, on peut utiliser la méthode `curve_fit` de `scipy.optimize`.

```
import numpy as np
from scipy import optimize as scpo
from matplotlib import pyplot as plt

x, y, z = np.loadtxt("data.dat", unpack=True)

def f(x, a) :
    return np.sqrt(a * x)

popt, pcov = scpo.curve_fit(f,x,y)          (les paramètres optimaux popt (et la matrice de covariance pcov))

t = np.linspace(0,8,200)

plt.xlim([0., 8.])
plt.ylim([0.,15.])
plt.grid()
plt.xlabel(r"$h$ (m)", fontsize=22)
plt.ylabel(r"$v$ (m/s)", fontsize=22)
plt.errorbar(x,y,yerr=z,color='green',linestyle='None',elinewidth=2.5, marker='o',markersize=6.,
                                                                        label="exp.")
plt.plot(t,f(t,popt[0]),yerr=z,color='darkorange', linewidth=2.5, label=r"$\sqrt{a\, h}$")
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.text(4.5,3.5,r"$a =$ " + str(round(popt[0],1)) + r" m/s$^2$", fontsize=22,
                                                                    backgroundcolor='white')
legend = plt.legend(loc = 2, numpoints = 1, handlelength=2., frameon=1, fontsize=22)
frame = legend.get_frame()
frame.set_facecolor('white')
frame.set_edgecolor('white')
plt.tight_layout()
plt.show()
```

On donne le résultat du tracé ci-dessous.

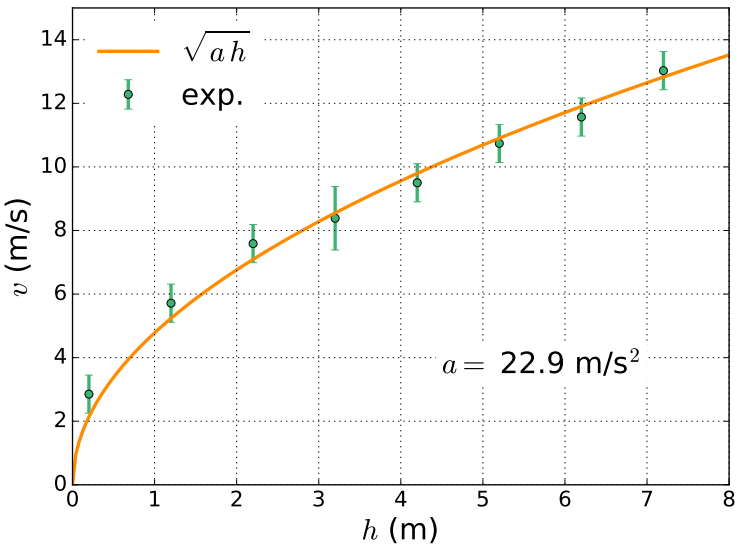

Figure : Représentation graphique de données avec un ajustement par `scipy.optimize.curve_fit`.

Remarque : Il est toujours préférable de travailler avec une droite. Ici on a notamment $v^2 = 2\,g\,h$. Le tracé de v^2 en fonction de h doit donner une droite de pente $2\,g$. Dans le cas d'une **régression linéaire**, on utilisera plutôt la

méthode `linregress` de `scipy.stats`, qui permet de récupérer facilement le coefficient R^2 (voir ci-dessous).

9 Faire une régression linéaire

Toujours sur l'exemple précédent, on peut aussi tracer v^2 en fonction de h qui doit donner une droite, et sur laquelle on peut par conséquent faire une régression linéaire. On utilise pour cela la méthode `linregress` de `scipy.stats`.

```
import numpy as np
from scipy import stats as scps
from matplotlib import pyplot as plt

x, y, z = np.loadtxt("data.dat", unpack=True)

y2 = y**2

slope, intercept, r_value, p_value, std_err = scps.linregress(x, yN)

t = np.linspace(0,8,200)

plt.xlim([0., 8.])
plt.ylim([0.,200.])
plt.grid()
plt.xlabel(r"$h$ (m)", fontsize=22)
plt.ylabel(r"$v^2$ (m$^2$/s$^2$)", fontsize=22)
plt.plot(t, slope*t + intercept, color='darkorange', linewidth=2.5, label=r"pente $a =$ " +
                                                                    str(round(slope,1)) + r" m/s$^2$")
plt.plot(x, yN, color='mediumseagreen', linestyle='None', marker='o', markersize=6.,
                                                                                label=r"exp.")
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.text(5.5,20.,r"$R^2 =$ " + str(round(r_value**2,3)), fontsize=22, backgroundcolor='white')
legend = plt.legend(loc = 2, numpoints = 1, handlelength=2., frameon=1, fontsize=22)
frame = legend.get_frame()
frame.set_facecolor('white')
frame.set_edgecolor('white')
plt.tight_layout()
plt.show()
```

On donne le résultat du tracé ci-dessous.

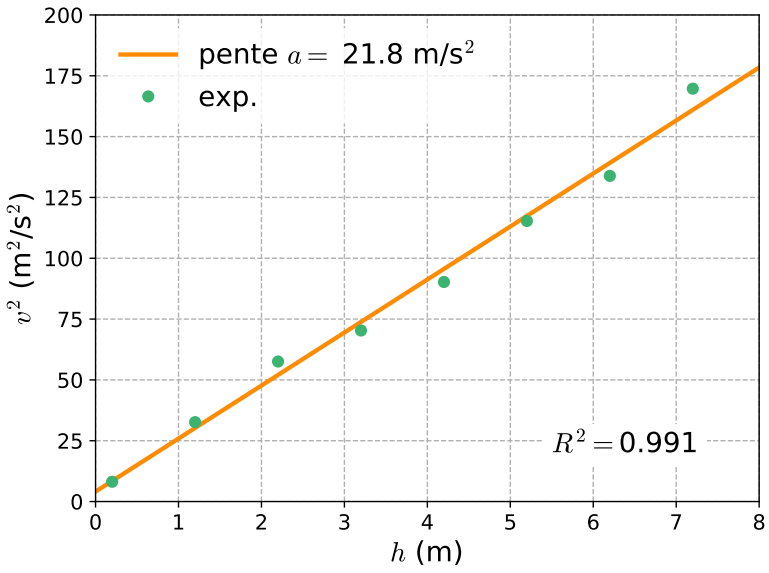

Figure : Représentation graphique de données avec un ajustement par `scipy.stats.linregress`.

10 Calculer et tracer une transformée de Fourier

La transformée de Fourier est un outil très utile pour étudier les signaux périodiques. En pratique, on cherche souvent à tracer le spectre du signal (le module carré de sa TF). On peut utiliser pour cela la méthode `fft` de `numpy.fft`. Un exemple est présenté ci-dessous.

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

t, y = np.loadtxt("data.dat", unpack=True)                          (récupérer les données dans un fichier)

tf = np.fft.fft(y)                                                  (calculer la TF)
frequency = np.fft.fftfreq(n=t.shape[-1], d=np.mean(np.diff(t)))    (calculer les fréquences dans la TF)
psd = np.square(np.abs(tf))                                         (calculer le spectre)
psdn = psd/psd.max()                                                (normaliser le spectre)

plt.xlim([0., 8.])
plt.ylim([0.,15.])
plt.grid()
plt.xlabel(r"$f$ (Hz)", fontsize=22)
plt.ylabel(r"Sp($f$) (u. a.)", fontsize=22)
plt.bar(frequency,psdn,width=2*np.mean(np.diff(t)))                 (tracer comme un histogramme)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.tight_layout()
plt.show()
```

Le résultat du tracé est :

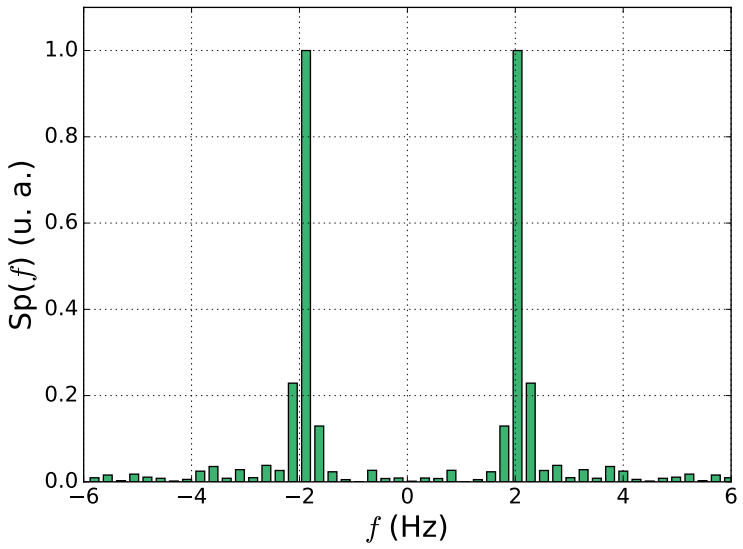

Figure : Représentation graphique d'un spectre obtenu par la méthode `fft` de `numpy.fft`.

Remarque : la transformée de Fourier inverse (de g) se calcule par `np.fft.ifft(g)`.

11 Résoudre une équation algébrique

On peut résoudre une équation algébrique $f(x) = 0$ (ou un système d'équations algébriques) grâce à la méthode `fsolve` du paquet `optimize` de `scipy`.

Résoudre une équation : On cherche à résoudre $\mathrm{e}^{-x} = 1 - x/5$ dont la solution est proche de 5.

On écrit

```
import numpy as np
from scipy import optimize as scpo

def eq(x):
    res = 1 - x / 5 - np.exp(-x)
    return res

esti = [5.]
y = scpo.fsolve(eq,esti)

print(y)
```

12 Écrire les résultats dans un fichier

Après avoir fait faire des calculs à python, on peut vouloir exporter les données, par exemple les écrire dans un fichier texte (ici data.txt). Pour écrire un tableau de deux colonnes, d'abscisse x et d'ordonnée y, on peut utiliser les lignes suivantes.

```
x = np.linspace(0,10,20)
y = x**2

fichier = open('data.txt','w')

for i in range(len(x)) :
    fichier.write( "%.2e\t %.2e\n" %(x[i], y[i]) )

fichier.close()
```

Explication : \t crée une tabulation, tandis que \n passe à la ligne.

13 Nombres aléatoires

La modélisation de certains processus physiques nécessite l'utilisation d'aléatoire (par exemple le modèle de la marche aléatoire pour rendre compte de la diffusion de particules). La bibliothèque `numpy` possède des générateurs de nombres aléatoires avec des distributions variées. Les distributions uniforme et gaussienne (ou « normale ») sont les plus utiles.

Distribution gaussienne : il faut préciser sa moyenne, son écart-type, et le nombre de points qu'on souhaite.

```
import numpy as np

moy = 5.
ecarttype = 2.
nb_pt = 100

x = np.random.normal(moy, ecarttype, nb_pt)
```

Distribution uniforme : il faut préciser sa borne basse, sa borne haute et le nombre de points qu'on souhaite.

```
import numpy as np

basse = 0.
haute = 10.
nb_pt = 100

x = np.random.uniform(basse, haute, nb_pt)
```

Entier aléatoire : Pour obtenir seulement un entier aléatoire entre a et b on utilisera

```
import numpy as np

a = 0
b = 10

x = np.random.randint(a, b)
```