Ajustement de courbes

nov. 14, 2018

curve_fit

On va utiliser la fonction `curve_fit` du package `scipy.optimize`. Cette fonction permet aussi d'obtenir l'incertitude des paramètres sous forme d'une matrice de corrélation.

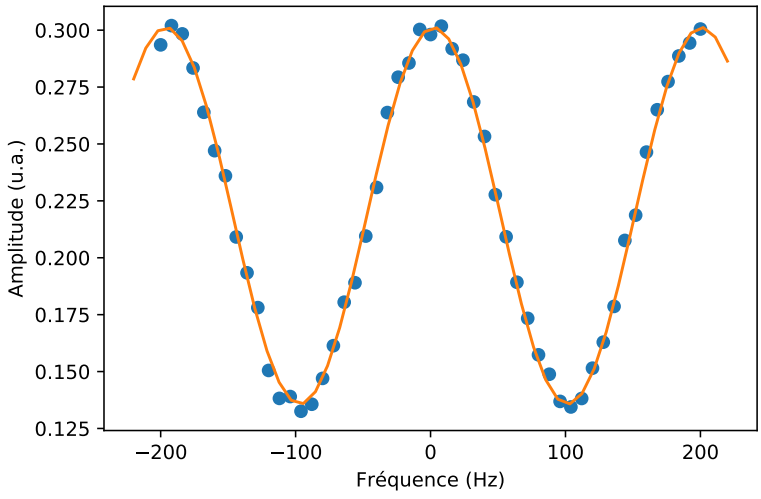
Fit de franges d'interférence

On souhaite ajuster les franges d'un interféromètre atomique. Les données sont dans le fichier `data/fit_sinus.dat`. La première colonne du fichiers (axe x) représente une fréquence en Hz. La seconde colonne représente la population mesurée pour une fréquence donnée. L'objectif est de trouver la position de la frange centrale.

On ajustera par une fonction cosinus avec une amplitude (b), un décalage vertical (offset, a), une position du centre f_0 et un interfrange ajustable Δf.

$$\mathcal{A}(f) = a + b\cos\left(2\pi(f - f_0)/\Delta f\right)$$

— Chargez et tracez les données sous forme de point.
— Écrivez la fonction de fit que l'on appellera `frange(x, ...)`. Tracez cette courbe entre -220 et +220 Hz avec des paramètres proches des paramètres expérimentaux.
— Calculez les paramètres optimaux. Quelle est la position et l'incertitude de la frange centrale.



Un peu de Python

La fonction `curve_fit` utilise la fonction `leastsq` (least square) du même package. Cependant elle est très peu pratique. En effet, les paramètres (`p0` ou `popt`) sont sous forme d'une liste ou d'un tableau alors qu'un dictionnaire serait beaucoup plus pertinant. Ecrire une fonction `my_curve_fit` qui fonctionne de la façon suivante

```
def fit_fonction(x, a=0, b=2, c=2): # paramètres initiaux par défaut
    return a*x**2 + b*x + c

popt = my_curve_fit(fit_fonction, xdata, ydata, p0={"a":4})
    # On utilise a=4 comme valeur initiale au lieu de a=0

print(popt['b'])
```

Dans la fonction `fit_fonction`, la première variable représente x et les suivantes représentent les paramètres du fit. Ce sont des keyword arguments dont la valeur par défault servira de paramètre initial du fit. Cette valeur peut être remplacé par le paramètre p0. On pourra automatiquement extraire le nom et la valeur de ces paramètres de la façon suivante

```
import inspect
args, varargs, varkw, defaults = inspect.getargspec(fit_fonction)
```

Diagramme de Bode

On souhaite réaliser un diagramme de Bode. Le signal issu d'un générateur passera par le filtre. Le signal de fréquence donnée (`frequence`) sera engeristré par un oscilloscope avant (`signal_reference`) et après le filtre (`signal_filtre`). On ajustera les deux sinusoides afin d'extraire leur amplitude et leur phase. On pourra donc en déduire le gain et le déphasage du filtre.

Créez une classe `BodePoint` qui représentera un point du diagramme de Bode

```
bode_point = BodePoint(temps, signal_reference, signal_filtre, frequence)
print(bode.gain)
```

On pourra l'utiliser avec les données suivantes

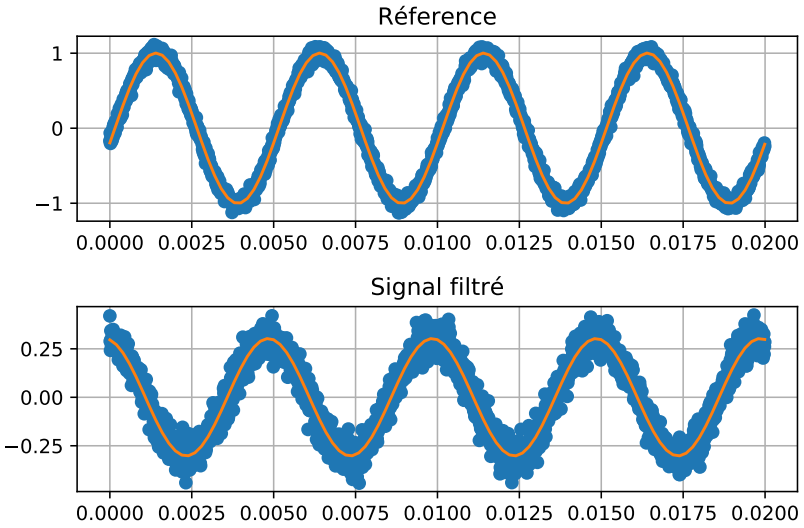
```
frequence = 200
dt = 1E-6
temps = dt*arange(5000)
signal_reference = sin(2*pi*frequence*temps - .2)
signal_reference += .05*np.random.normal(size=len(temps))
signal_filtre = .3*cos(2*pi*frequence*temps + .2)
signal_filtre += .05*np.random.normal(size=len(temps))
```

— On pourra commencer par créer une classe `FitSinus` qui permettra d'obtenir l'amplitude ou la phase d'un signal.

— La fréquence restera un paramètre du fit même si on la connait. Par contre, il faudra l'utiliser comme paramètre initial.
— On pourra utiliser des `property`, par exemple

```
    @property
    def gain(self):
        return self._sin_fil.amplitude / self._sin_ref.amplitude
```

— On fera aussi une méthode `plot` qui affichera les deux signaux fittés :



Fit par une loi de Poisson

On effectue une expérience de spectroscopie en régime de comptage de photons. La forme de raie est une Lorentzienne :

$$\frac{1}{1 + ((f - f_0)/\Gamma)^2}$$

On va prendre $f_0 = 0$ et $\Gamma = 1$.

Le signal est très faible et on mesure environ 5 coups par seconde au sommet de la courbe. La statistique est une loi de Poisson dont le paramètre est donnée par la forme de raie. La probabilité d'avoir k photons à la fréquence f est donné par :

$$P(k, f) = \frac{\lambda(f)^k}{k!} e^{-\lambda(f)}$$

avec

$$\lambda(f) = \frac{\alpha}{1 + ((f - f_0)/\Gamma)^2}$$

Les mesures ont été prises pour des fréquences f allant de -3Γ à 3Γ avec un pas de $\Gamma/10$.

— On simule un jeu de donnée à l'aide de la fonction suivante

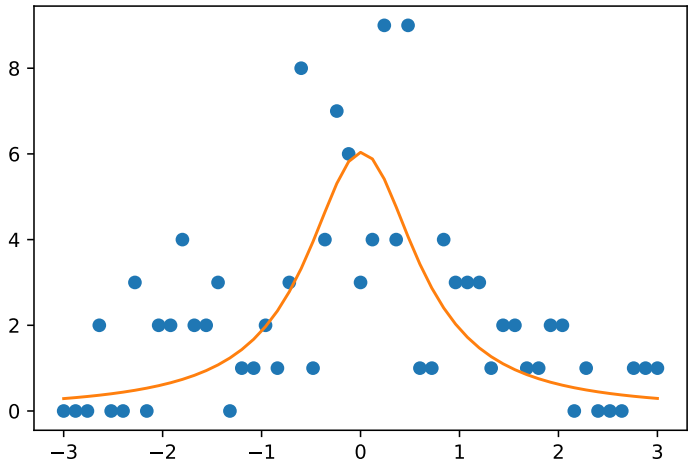
```
    Gamma = 1
    f_0 =  0
    amplitude = 5.

    f_mesure = linspace(-3, 3, 51)

    def lorentz(f, amplitude=amplitude, f_0=f_0, Gamma=Gamma):
        return amplitude/(1+((f-f_0)/Gamma)**2)

    def simulate_data():
        return np.random.poisson(lam=lorentz(f_mesure))
```

— Effectuez un fit par la méthode des moindre carré
— Effectuez un fit en maximisant la fonction de vraissemblance (on pourra utiliser `minimise` de `scipy.optimize` et la fonction `scipy.stats.poisson.pmf` pour calculer la fonction de vraissemblance).
— Comparez la variance des deux estimateurs.



Fit d'une image

Le fichier `data/double_star` contient une image de 64 par 64 pixels d'une image d'une étoile double. L'objectif de cette partie est d'ajuster cette image par la somme de deux Gaussiennes afin d'en déterminer la distance.

L'ajustement d'une image procède de la même manière que l'ajustement d'une courbe. Le tableau xdata sera alors un tableau de taille N par 2 correspondant aux coordonnées des N points de l'image ($N = 64^2$ dans notre exemple) et ydata un tableau de tailles N.

Voici l'exemple d'un fit par une gaussienne simple

```
ny, nx = image.shape
X,Y = meshgrid(range(nx), range(ny))
xdata = array([X.flatten(), Y.flatten()]).transpose()
def gauss(xdata, amplitude, center_x, center_y, diameter):
```

```
    x = xdata[:,0]
    y = xdata[:,1]
    return amplitude*exp(-((x-center_x)**2 + (y-center_y))/diameter**2)

popt, pcov = curve_fit(gauss, xdata, image.flatten(), p0)
```

— Adaptez cet exemple afin de fitter l'image par deux gausiennes