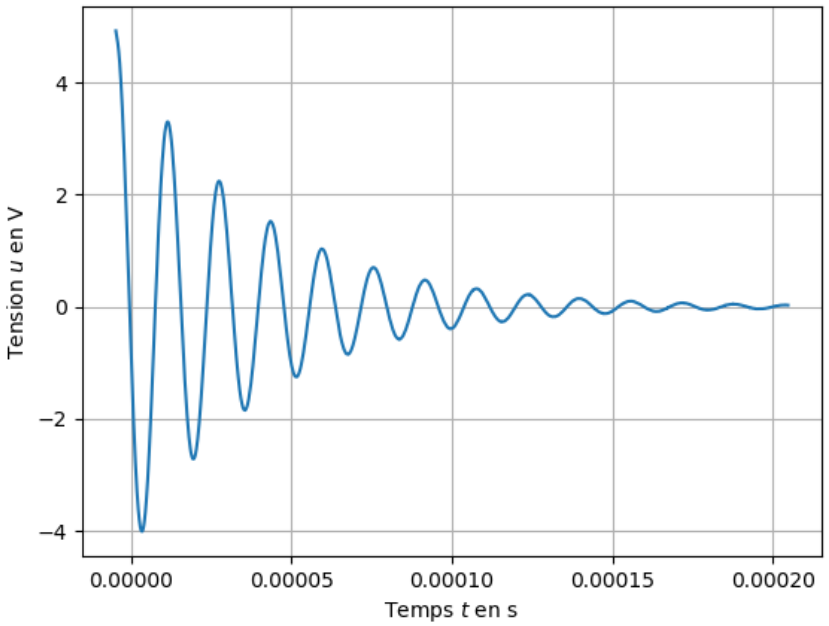
PP06 : Temps caractéristique τ d'atténuation

Énoncé : Vous disposez d'un signal oscillant (pseudo-)périodiquement autour de 0 de manière régulière et vous avez réussi à déterminer la position des maxima consécutifs (voir un exercice précédent). Débrouillez-vous pour, après avoir récupéré les positions des maxima en utilisant la méthode de l'exercice précédent, faire un ajustement de $\ln(u_{\max})$ en fonction du temps dont le coefficient directeur correspond à $-1/\tau$. Le but est bien sûr de trouver τ.

Hint : si jamais votre détection n'est pas parfaite, vous pouvez très bien sélectionner un sous-ensemble de maxima « qui vont bien » pour faire l'ajustement par la suite.



On verra plus tard dans l'année que Numpy permet de faire des calculs « au vol » sur un certain nombre de grandeur, notamment appliquer le logarithme néperien sur une « pseudo-liste » (on l'appellera bientôt un « Numpy array ») que l'on peut produire au vol avec la fonction `np.array`. On peut donc faire une ligne du type[1] `lnU = np.log(np.array(u))` où `u` est la liste des maxima que vous avez déterminés.

Pour ajuster n'importe quelle formule sur des données dont les abscisses sont stockées dans `x` et les ordonnées expérimentales dans `y`, on peut utiliser la fonction `curve_fit(f,x,y)` du module `scipy.optimize`. Elle prend principalement trois arguments :

- la fonction `f` de modélisation dont le premier argument doit être les abcisses `x` utilisées et les arguments suivants correspondent aux paramètres que l'on veut ajuster ;
- la liste `x` des abscisses ;
- la liste `y` des ordonnées expérimentales correspondantes.

Elle renvoie deux éléments dont le premier seulement nous intéresse ici : il s'agit de la liste `p` des paramètres (dans l'ordre de définition de la fonction `f`) dont les valeurs permettent de coller au plus près des données. Voici un exemple où l'on essaie d'ajuster une fonction trigonométrique de pulsation, d'amplitude et de déphasage inconnus.

1. Le logarithme néperien « ln » en français est appelé `log` dans les langages de programmation alors que le logarithme décimal « log » français est noté `log10` en général.

```python
import scipy as sp
import scipy.optimize
import matplotlib.pyplot as plt

# Définition de la fonction d'ajustement (appelée 'onde' ici plutôt que 'f')
def onde(x,A,omega,phi):
    return A*np.cos(omega*x + phi)

# On appelle la procédure d'ajustement complète (la liste p contient les
# paramètres après ajustement, pcov ne nous intéresse pas pour l'instant) en
# donnant la fonction d'ajustement, les abcisses x et les ordonnées y (dont
# l'obtention n'est pas montrée ici)
p,pcov = sp.optimize.curve_fit(onde,x,y,p0=[2,2,2])
# Remarque: l'option "p0=..." permet de donner une première estimation des
# paramètres pour "guider" la procédure de résolution, histoire qu'elle parte
# assez proche des bonnes valeurs pour pouvoir les trouver sans peine.

# On récupère les paramètres dans des noms explicites
A,omega,phi = p

# On regarde l'ajustement correspondant
ajustement = onde(x,A,omega,phi)

# Ne reste plus qu'à faire le tracé graphique
# D'abord les données
plt.plot(x,y,'o',label='Donnees')
# Ensuite le label pour l'ajustement
lab = 'Ajustement: $A={}$,\n $\\omega={}$, $\\varphi={}$'
lab = lab.format(round(A,2),round(omega,2),round(phi,2))
# et l'ajustement proprement dit
plt.plot(x,ajustement,label=lab)
# Puis les fioritures
plt.xlabel('Abcisses $x$')
plt.ylabel('Ordonnees $y$')
plt.title("Exemple d'ajustement")
plt.grid()
plt.legend(loc='best')
nom='PS/tau_second_ordre_ajustement.png'
plt.savefig(nom)
plt.clf()
```