Oscillateurs en mécanique.
Étude expérimentale d'un système **solide-ressort.**
Compte rendu du TP.

* * *

Introduction

Le système qui sera étudié dans le présent TP est un mobile de masse $m = 66,9$ g, assimilé au point materiel **M**, attaché à un ressort sur un support horizontal. Ce dernier sera étudié dans le référentiel lié au plan horizontal sur lequel il se déplace, référentiel terrestre considéré galiliéen pendant la durée du mouvement. Dans le présent TP, nous entreprendrons dans un premier temps une étude théorique, déjà vue en cours, qui servira de base permettant de mener à bien le travail expérimental qui suivera. En effet, nous disposerons deux fichiers contenant les positions du mobile au cours du temps lors de deux expériences différentes, notre travail consistera à déterminer les constantes du système. Nous prendrons le soin d'expliquer chacune au début de leurs parties respectives.

Oscillations libres

Dans cette partie, comme son nom l'indique, le mobile oscillera librement, et on prendra en compte, dans notre étude théorique, les frottements fluides.

Étude théorique

Il s'agira d'établir l'équation différentielle décrivant l'évolution de l'allongement du ressort. Pour cela, nous appliquerons le principe fondamental de la dynamique, mais avant de pouvoir le faire, un bilan des forces s'impose. On finira par la plus caractéristique de ce type de systèmes : la froce de rappel du ressort.

Le point **M** est soumis à son poids $\vec{P}$, force verticale vers le bas, $\vec{R}$, force verticale vers le haut, parce qu'il n'y a pas de frottemtens avec le plan, les frottements fluides dont la résultante est notée $\overrightarrow{f_{\text{air}}}$ et la force de rappel du ressort $\overrightarrow{F_{\text{rappel}}}$. Par définition, la force de rappel caractérise la raideur du ressort et s'exprime de la manière suivante :

$$\boxed{F_{\text{rappel}} = k \times \text{allongement}}$$

Le point où est fixé le ressort sera l'origine du repère. Ainsi, l'allongement du ressort est égal à $x - \ell_0$. En tenant compte du signe de $x - \ell_0$ et de l'expression de la force de rappel, il vient que $\overrightarrow{F_{\text{rappel}}} = -k(x-\ell_0)\vec{e_x}$. De plus, comme la poistion du mobile n'évoluera que selon l'axe Ox, on peut en déduire que les forces du poids de la réaction du support s'annulent. On peut alors appliquer le principe fondamental de la dynamique afin d'en déduire l'équation différentielle régissant le mouvement.

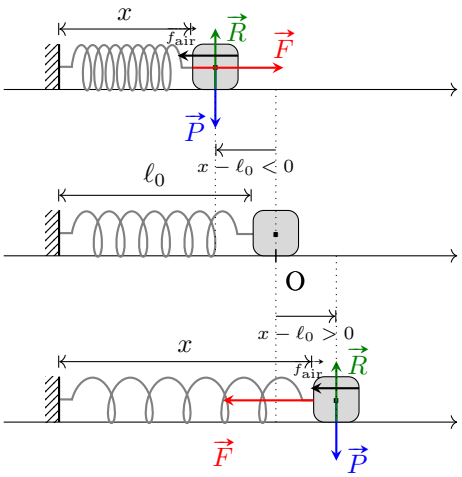

FIGURE 1 – Forces s'exerçant sur la masse accrochée au ressort horizontal.

L'application du **PFD** donne :

$$m\vec{a} = \vec{P} + \vec{R} + \overrightarrow{f_{\text{air}}} + \vec{F}$$

L'expression de la force de frottement fluide est $f_{\text{air}} = -\alpha\vec{v}$ où $\vec{v}$ est la vitesse du point **M**. Ainsi, en projetant, on obtient la relation :

$$m\ddot{x} = -k(x - \ell_0) - \alpha\dot{x}$$
$$\iff \boxed{\ddot{x} + \frac{\alpha}{m}\dot{x} + \frac{k}{m}x = \frac{k}{m}\ell_0}$$

On pose $\omega_0^2 = \frac{k}{m}$, $\lambda = \frac{\alpha}{2m}$ et $x_{eq} = \ell_0$. Ainsi, l'équation différentielle s'écrit

$$\ddot{x} + 2\lambda\dot{x} + \omega_0^2 x = \omega_0^2 x_{eq} \qquad \text{(E)}$$

On se place dans le cas d'un régime pseudo-périodique, i.e $\Delta = 4\lambda^2 - 4\omega_0^2 < 0$, d'où les deux solutions complexes $r_{1,2} = -\lambda \pm j\sqrt{\omega_0^2 - \lambda^2}$. En posant $\omega = \sqrt{\omega_0^2 - \lambda^2}$ la pseudo-pulsation, on a trouve une solution $x(t)$ de la forme

$$\boxed{x(t) = A\exp(-\lambda t)\cos(\omega t + \phi) + x_{eq}}$$

Étude expérimentale

On dispose déjà d'un ficher contenant les positions du mobile en fonction du temps. À l'aide du module python `scipy.optimize`, on peut retrouver les différentes constantes du problème. L'utilisation de ce dernier impose de transformer les données en tableaux `numpy`, afin que les calculs puissent être menés. On crée alors un fichier que l'on nomme `utils.py`, qui contiendra un programme qui s'en occupera. En voici le code :

```python
import re
pattern = re.compile("-?[.0-9]*;-?[.0-9]*")
split_token = ";"

def exctract_data(file):
    x_data, y_data = [], []
    for line in file:
        is_data = pattern.match(line)
        if not is_data:
            continue
        x, y = line.split(split_token)
        x_data.append(float(x))
        y_data.append(float(y))
    return x_data, y_data
```

On utilise des expressions régulières pour s'assurer qu'on ne prend que les données qui nous intéressent, et on sépare les données en deux tableaux, un tableau contenant la variable de temps, et un autre contenant la varaible de position selon l'axe Ox. Ensuite, on modélise $x(t)$ en définissant un modèle qui sera la fonction `function` dans notre programme. On affiche les paramètres.

```python
from scipy.optimize import curve_fit
import numpy as np
from utils import exctract_data
import matplotlib.pyplot as plt


path_data_libre = "/Users/mac/documents/prepa/rabelais/physique/TPs/tp-pc-22/data-libre/data-y.txt"
path_data_force = "/Users/mac/documents/prepa/rabelais/physique/TPs/tp-pc-22/data-force/data-y.txt"


def function(x, a, b, c, d, e):
    return a*np.exp(-b*x)*np.cos(c*x+d)+e


if __name__ == "__main__":
    """
    Cas libre
    """
    file = open(path_data_libre, 'r')
    x_data, y_data = exctract_data(file)
    xdata, ydata = np.array(x_data), np.array(y_data)
    popt, pcov = curve_fit(function, xdata, ydata)
    print(popt)
```

On lit en console :

```
2.52510489   0.02635128   9.93313974   -1.09272005   -0.0617848
```
A λ ω ϕ x_{eq}

On a $\omega = \sqrt{\omega_0^2 - \lambda^2} \iff \omega_0^2 = \omega^2 + \lambda^2$. Or, $\omega_0^2 = \frac{k}{m} \iff k = m\omega^2$. L'application numérique donne : $k \simeq 6.6$ N/m.

Étude du décrément logarithmique

1. On montre que $\delta = \ln\left(\dfrac{x(t)}{x(t+T)}\right)$. On commence par remarquer que

$$\begin{aligned}\frac{x(t)}{x(t+T)} &= \frac{A\exp(-\lambda t)\cos(\omega t+\phi)+x_{eq}}{A\exp(-\lambda(t+T)\cos(\omega(t+T)+\phi)+x_{eq}} \\ &= \frac{A\exp(-\lambda t)\cos(\omega t+\phi)+x_{eq}}{A\exp(-\lambda t)\exp(-\lambda T)\cos(\omega t+\phi)+x_{eq}}\end{aligned}$$

   Or, x_{eq} est négligeable devant le reste, ainsi $\dfrac{x(t)}{x(t+T)} = \exp(\lambda T)$. Donc $\delta = \lambda T$; le même calcul nous permet d'en déduire $\dfrac{x(t)}{x(t+nT)} = \exp(\lambda nT)$, d'où

$$\ln\left(\frac{x(t)}{x(t+nT)}\right) = \lambda nT = n\delta \iff \boxed{\delta = \frac{1}{n}\ln\left(\frac{x(t)}{x(t+nT)}\right)}$$

2. En utilisant notre programme, on peut mesurer δ sur 39 périodes, par exemple. On modifie une partie de ce dernier pour avoir ceci :

```python
    if __name__ == "__main__":
        """
        Décrément logarithmique
        """
        file = open(path_data_libre, 'r')
        x_data, y_data = exctract_data(file)
        xdata, ydata = np.array(x_data), np.array(y_data)
        popt, pcov = curve_fit(function, xdata, ydata)
        n = 40
        t1, t2 = 0.63254776149755, n*0.63254776149755
        y1, y2 = function(t1, *popt), function(t2, *popt)
        delta = np.log(y1/y2)*(1/n)
        print(delta)
```

   On lit alors en console :

```
0.017595933694887392
```

3. On en déduit la valeur de λ, en effet on a $\delta = \lambda T \iff \lambda = \frac{\delta}{T}$. Donc après application numérique $\lambda \simeq 0.027$.

Étude de la vitesse v

On modifie la dernière partie de notre programme :

```python
if __name__ == "__main__":
    """
    Vitesse
    """
    file = open(path_data_libre, 'r')
    x_data, y_data = exctract_data(file)
    xdata, ydata = np.array(x_data), np.array(y_data)
    popt, pcov = curve_fit(function, xdata, ydata)
```

```
    plt.plot(xdata, function(xdata, *popt), 'r-', label='x(t)')
    plt.plot(xdata, fp(xdata, *popt), 'b-', label='v(t)')
    plt.show()
```

Et on l'exécute :

On mesure le retard entre $x(t)$ et $v(t)$, on trouve qu'il est de l'ordre de -164 ms. On le note t_r. On rappelle que la formule liant déphasage et retard : $\boxed{\Delta\varphi = 2\pi t_r}$. L'application numérique donne : $\Delta\varphi = \frac{-\pi}{3}$. (La mesure a été faite manuellement, directement sur la fenêtre de visualisation de la figure ; elle aurait pu être automatisée, mais cela aurait constitué une perte de temps étant donnée la simplicité de la tache.)

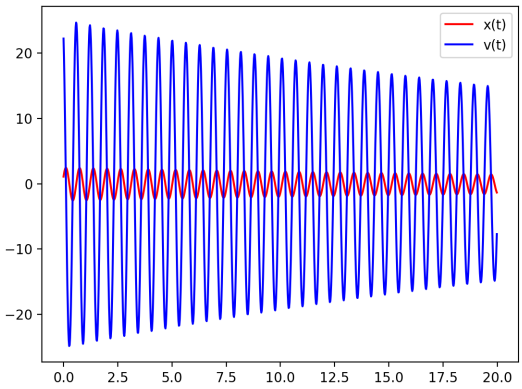

FIGURE 2 – Les courbes de position et de vitesse en fonction du temps.

Étude énergétique

On sait que l'expression de l'énergie cinétique est $E_c = \frac{1}{2}mv^2$. On possède le tableau des valeurs de la vitesse en fonction du temps, ainsi que celui de la position en fonction du temps. Ainsi, comme la seule énergie potentielle considérée est celle élastique, on a $E_p = \frac{1}{2}kx$. On modifie la fin de notre programme :

```
def function(x, a, b, c, d, e):
    return a*np.exp(-b*x)*np.cos(c*x+d)+e

def fp(x, a, b, c, d, e):
    return -a*np.exp(-b*x)*(b*np.cos(c*x+d)+c*np.sin(c*x+d))

def ep(x, a, b, c, d, e):
    return 1/2*6.6*(function(x, a, b, c, d, e)**2)

def ec(x, a, b, c, d, e):
    return 1/2*0.0666*(fp(x, a, b, c, d, e)**2)

if __name__ == "__main__":
    """
    Énergies cinétique, potentielle et mécanique
    """
    file = open(path_data_libre, 'r')
    x_data, y_data = exctract_data(file)
    xdata, ydata = np.array(x_data), np.array(y_data)
    popt, pcov = curve_fit(function, xdata, ydata)
    Ec = ec(xdata, *popt)
    Ep = ep(xdata, *popt)
    Em = Ec + Ep
    plt.figure(figsize=(20, 6), dpi=80)
    plt.plot(xdata, Ec, 'r-', label='Énergie cinétique')
    plt.plot(xdata, Ep, 'b-', label='Énergie potentielle')
    plt.plot(xdata, Em, 'g-', label='Énergie mécanique')
    plt.legend()
    plt.xlabel('Énergie (J)')
```

```
30      plt.ylabel('temps (s)')
31      plt.show()
```

On a, après exécution du programme :

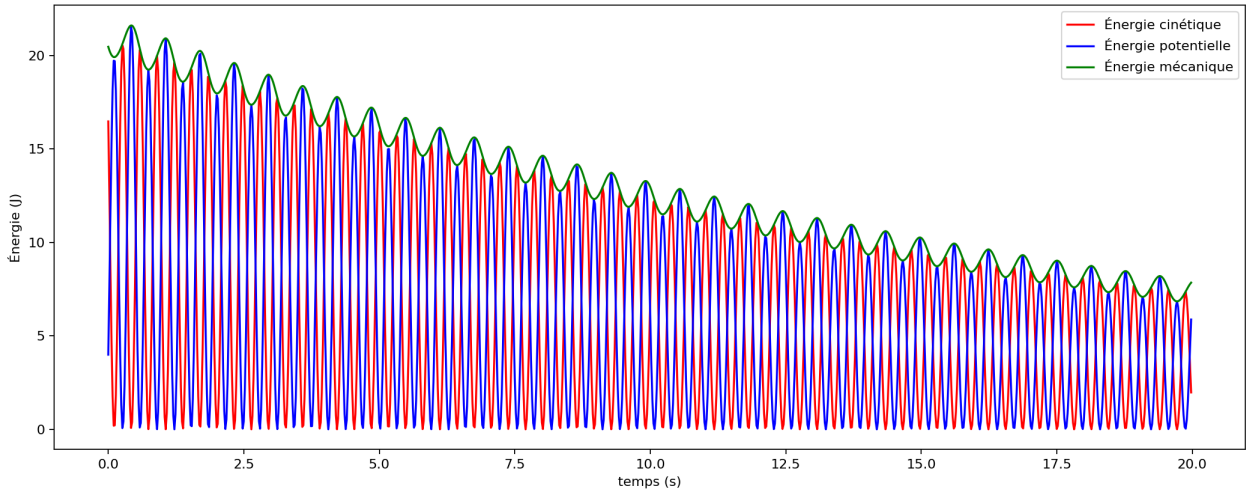

FIGURE 3 – Tracé des courbes énergétiques.

Portrait de phase

On dispose des tableaux de la position et de la vitesse en fonction du temps. Il suffit de représenter le second en fonction du premier :

```
1   if __name__ == "__main__":
2       """
3       Énergies cinétique, potentielle et mécanique
4       """
5       file = open(path_data_libre, 'r')
6       x_data, y_data = exctract_data(file)
7       xdata, ydata = np.array(x_data), np.array(y_data)
8       popt, pcov = curve_fit(function, xdata, ydata)
9       plt.figure(figsize=(10, 6), dpi=80)
10      plt.plot(ydata, fp(xdata, *popt), 'r-')
11      plt.ylabel('Vitesse (m/s)')
12      plt.xlabel('Position (m)')
13      plt.show()
```

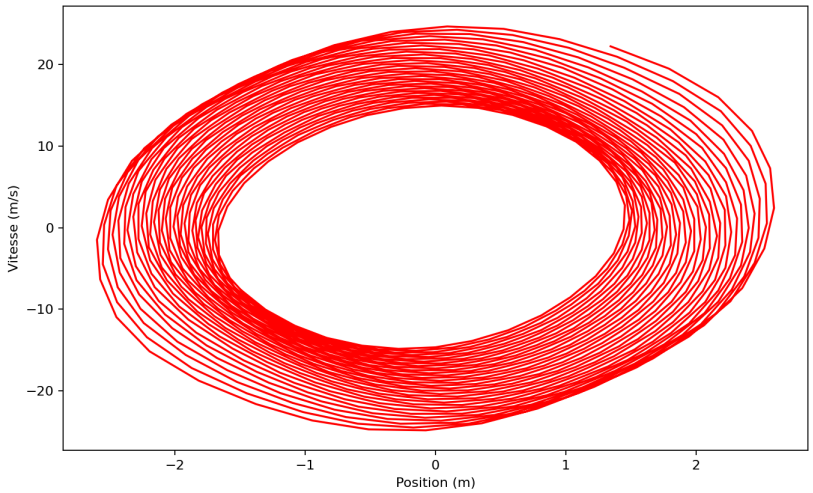

FIGURE 4 – Portrait de phase.

Oscillations forcés

L'expérience précédente est répétée, mais à quelques détails près. En effet, le ressort horizontal a une extrémité qui est excitée de façon sinusoïdale. On recommence la même étude que tout à l'heure, en commençant par une étude théorique.

Étude théorique

Le bilan des forces diffère du premier cas en un seul point : la force d'excitation $\overrightarrow{F_e}$. Étant donné qu'elle agit de manière sinusoïdale, on a

$$\overrightarrow{F_e} = F_m \cos(\Omega t)\overrightarrow{e_x}$$

L'application du **PFD** donne :

$$m\vec{a} = \vec{P} + \vec{R} + \overrightarrow{f_{\text{air}}} + \vec{F} + \overrightarrow{F_e}$$

Ainsi, en projetant, on obtient la relation :

$$m\ddot{x} = -k(x - \ell_0) - \alpha\dot{x} + F_m \cos(\Omega t)$$

$$\iff \boxed{\ddot{x} + \frac{\alpha}{m}\dot{x} + \frac{k}{m}x = \frac{k\ell_0 + F_m \cos(\Omega t)}{m}}$$

La solution de l'équation différentielle est une somme d'une solution homogène et d'une solution particulière. Dans notre cas, on remarquera que la solution homogène devient négligeable devant la solution particulière lorsque le régime est permanent. Or, en passant aux complexes, on trouve une solution particulière de la forme :

$$x_p(t) = x_{pmax} \cos(\Omega t + \phi_p)$$

Étude expérimentale

On lance alors le programme suivant :

```python
from scipy.optimize import curve_fit
import numpy as np
from utils import exctract_data
import matplotlib.pyplot as plt


path_data_libre = "/Users/mac/documents/prepa/rabelais/physique/TPs/tp-pc-22/data-libre/data-y.txt"
path_data_force = "/Users/mac/documents/prepa/rabelais/physique/TPs/tp-pc-22/data-force/data-y.txt"


def function(x, a, b, c):
    return a*np.cos(b*x+c)


def fp(x, a, b, c):
    return -a*b*np.sin(b*x+c)


def ep(x, a, b, c):
    return 1/2*6.6*(function(x, a, b, c)**2)


def ec(x, a, b, c):
    return 1/2*0.0666*(fp(x, a, b, c)**2)


if __name__ == "__main__":
    """
    Cas régime forcé
    """
    file = open(path_data_force, 'r')
    x_data, y_data = exctract_data(file)
    xdata, ydata = np.array(x_data), np.array(y_data)
    popt, pcov = curve_fit(function, xdata, ydata)
    Ec = ec(xdata, *popt)
    Ep = ep(xdata, *popt)
    Em = Ep+Ec
    fig = plt.figure(1, figsize=(10, 6), dpi=80)
    ax1 = fig.add_subplot(211)
    ax1.plot(Ec, 'r-', label='Énergie cinétique')
    ax1.plot(Ep, 'b-', label='Énergie potentielle')
    ax1.plot(Em, 'g', label='Énergie mécanique')
    plt.legend()
    plt.xlabel('temps (s)')
    plt.ylabel('Énergie (J)')
    ax2 = fig.add_subplot(212)
    ax2.plot(function(xdata, *popt), fp(xdata, *popt), '-b', label='Portrait de phase')
    plt.xlabel('Position (m)')
    plt.ylabel('Vitesse (m/s)')
    plt.legend()
    plt.show()
```

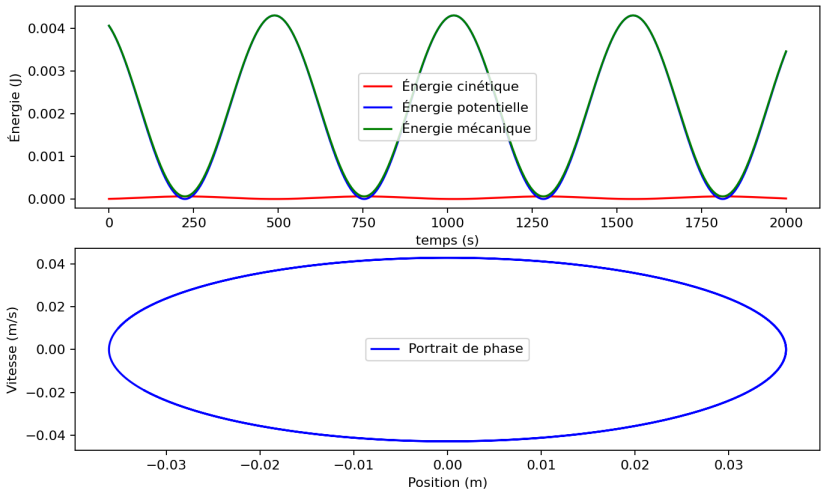

FIGURE 5 – Dernière figure !