TP – Chute d'une bille dans du glycérol

NOM :

L'objectif de ce TP est d'apprendre à résoudre numériquement une équation différentielle (ED) d'ordre 1. Savoir résoudre numériquement une ED est une compétence indispensable en physique puisque cela permet d'obtenir la solution d'un problème, même si l'ED n'admet pas de solution analytique.

I) Exemple général

Considérons l'équation différentielle d'ordre 1 suivante, vue dans le chapitre E1 :

$$\frac{df}{dt} + \frac{f(t)}{\tau} = \frac{f_{\text{SP}}}{\tau}$$

1) Résolution par la méthode d'Euler

La méthode d'Euler consiste à isoler $f(t+dt)$ puis à discrétiser la relation obtenue.

On rappelle les grandes étapes de la résolution d'une ED par la méthode d'Euler :

- écrire $df = f(t+dt) - f(t)$ puis isoler $f(t+dt)$;
- discrétiser la relation obtenue : remplacer $f(t) \mapsto f[n]$ et $f(t+dt) \mapsto f[n+1]$;
- faire une boucle qui calcule récursivement les valeurs de $f[n]$ à l'aide de la relation précédente.

🏠 Dans l'équation différentielle ci-dessus, exprimer $f(t+dt)$ en fonction de $f(t)$, f_{SP}, τ et dt.

🏠 Discrétiser la relation précédente : obtenir une relation de récurrence pour $t[n]$ et $f[n]$.

🏠 Écrire une boucle `while` qui calcule récursivement les valeurs de $f[n]$ à l'aide de la relation précédente.

2) Résolution avec la fonction « `odeint` »

La bibliothèque `scipy.integrate` possède une fonction `odeint` qui permet de résoudre une ED. Démarche à suivre :

- définir une fonction qui renvoie la dérivée $f'(t)$, dont l'expression est donnée par l'équation différentielle ;
- définir la condition initiale ;
- définir un array des temps auxquels la fonction $f(t)$ sera évaluée ;
- appeler la fonction `odeint`.

```
def deriv(f, t):
    return # à compléter : f → fonction, t → temps

CI = # à compléter
t = np.linspace(0, t_max, 10000) # t_max à définir
f = odeint(deriv, CI, t)
```

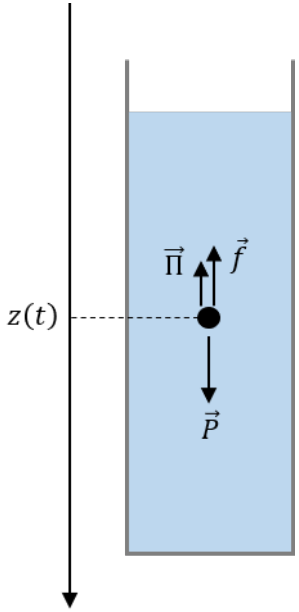
II) Chute d'une bille dans du glycérol

1) Étude théorique

On étudie la chute verticale d'une bille en acier de volume V et de masse volumique ρ_b (donc de masse $m = \rho_b V$) dans une éprouvette cylindrique remplie de glycérol de masse volumique μ.

La bille est soumise à 3 forces :

- son poids $\overrightarrow{P} = \rho V g \overrightarrow{u_z}$
- la poussée d'Archimède $\overrightarrow{\Pi} = -\mu V g \overrightarrow{u_z}$
- une force de frottement fluide $\overrightarrow{f} = -k v^n \overrightarrow{u_z}$



L'objectif du TP est de déterminer le nombre n qui intervient dans la force de frottement fluide.

🏠 Projeter la seconde loi de Newton sur l'axe (Oz) et montrer que l'équation différentielle du mouvement se met sous la forme :

$$\frac{dv}{dt} = b \times \left[1 - \left(\frac{v}{a}\right)^n\right]$$

avec a et b des constantes positives à déterminer.

🏠 Que représente le paramètre a ?

2) Analyse des données expérimentales

Par souci de temps, l'analyse de la vidéo a été faite en amont. Les données expérimentales (vitesse de chute en fonction du temps) ont été renseignées dans un script Python.

💻 Télécharger le fichier Python. Exécuter le code pour visualiser les données expérimentales.

💻 Résoudre l'équation différentielle par la méthode d'Euler. Quelle valeur de n ajuste au mieux les données expérimentales ? *Consigner votre code ci-dessous et commenter.*

💻 Résoudre l'équation différentielle avec la fonction `odeint`. *Consigner votre code ci-dessous et commenter.*

3) Pour aller plus loin...

Écrire une fonction `my_odeint` qui reproduit le comportement de la vraie fonction `odeint`. La fonction doit prendre les 3 mêmes arguments et renvoyer la solution de l'équation différentielle, obtenue par la méthode d'Euler. Déterminer la solution de l'ED à l'aide de la fonction `my_odeint`. Afficher la solution obtenue par-dessus les précédentes résolutions. *Consigner votre code ci-dessous.*

Aides pour Python

`import numpy as np` permet d'importer l'ensemble des fonctions du module numpy.

`u.append(a)` ajoute l'élément `a` à la fin de la liste `u`.

`u[-1]` renvoie le dernier élément de la liste `u`.

`np.linspace(a, b, N)` crée un array de `N` valeurs linéairement espacées entre `a` et `b`.

`odeint(deriv, CI, t)` renvoie la solution de l'équation différentielle donnée par la fonction `deriv`, avec pour condition initiale `CI`, et évaluée aux temps `t`.

`plt.plot(x, y, 'b-')` trace `y` en fonction de `x` avec un trait `'-'` bleu `'b'`. Il est possible de changer la couleur : rouge `'r'`, vert `'g'`, noir `'k'` et de remplacer le trait `'-'` par un trait pointillé `'--'` ou par un nuage de points `'o'`.