BLAISE PASCAL
PT 2021-2022

TP 5 – Informatique

Modélisation de la trajectoire d'un volant de badminton

L'objectif de ce TP mi-informatique mi-physique est de réviser la résolution numérique d'équations différentielles tout en analysant la force de frottement subie par un volant de badminton à partir de l'étude de sa trajectoire. Nous résoudrons numériquement l'équation du mouvement du volant pour deux modèles de frottements, linéaire et quadratique, en utilisant la méthode d'Euler puis une fonction issue d'une bibliothèque. Les résultats obtenus seront ensuite comparés à une chronophotographie.

Données : masse du volant étudié $m = 5{,}3\,\mathrm{g}$ et accélération de la pesanteur $g = 9{,}81\,\mathrm{m\cdot s^{-2}}$; à définir comme deux variables globales.

I - Remise en route sur la méthode d'Euler

I.A - Schéma d'Euler

Raisonnons sur un exemple simple : le tension aux bornes d'un condensateur d'un bête circuit RC régi par l'équation différentielle

$$\frac{\mathrm{d}u}{\mathrm{d}t} + \frac{1}{\tau}u = \frac{1}{\tau}e \qquad \text{avec} \qquad \begin{cases} \tau = RC \\ e(t) \text{ un forçage quelconque mais connu} \end{cases}$$

Comme il n'est pas possible de résoudre une équation différentielle « à tout instant » avec un ordinateur, il est nécessaire de passer par une résolution à N instants discrets $t_0, t_1, ..., t_n, ..., t_{N-1}$ en cherchant à ce que la valeur obtenue numériquement u_n soit aussi proche que possible de ce que donnerait la solution analytique $u(t_n)$. Pour toute la suite, le pas de temps Δt sera pris constant : $t_n = n\,\Delta t$.

En pratique, il faut transformer l'équation différentielle en une relation de récurrence permettant de calculer u_{n+1} à partir uniquement de ce que l'on connaît, c'est-à-dire la tension aux instants passés $u_n, u_{n-1}, ...$, la tension de forçage à tout instant et les paramètres caractéristiques du système. La « recette » pour passer de l'équation différentielle à la relation de récurrence est appelée **schéma numérique**. Le schéma d'Euler explicite est le plus simple : la dérivée est approximée par un taux de variation entre les instants t_n et t_{n+1}, et toutes les autres grandeurs sont prises à l'instant t_n. Ainsi, l'équation différentielle devient

$$\frac{u_{n+1} - u_n}{\Delta t} + \frac{1}{\tau}\,u_n = \frac{1}{\tau}\,e(t_n) \qquad \text{soit} \qquad u_{n+1} = u_n + \frac{\Delta t}{\tau}\big(e(t_n) - u_n\big)\,.$$

Bien sûr, il existe des schémas numériques plus astucieux, plus précis et/ou plus rapides, d'où l'intérêt d'utiliser des fonctions déjà implémentées comme `odeint` du module `scipy`. Notons d'ailleurs qu'il existe des domaines de recherche comme la climatologie, la turbulence, l'astrophysique, la physique des plasmas, la météorologie, etc., dans lesquels le gros du travail théorique consiste à construire des schémas numériques performants pour résoudre des équations « assez simples » à poser.

I.B - Implémentations en Python

Rappelons que pour un même algorithme, ici une même équation différentielle et un même schéma numérique, il existe souvent de nombreuses implémentations différentes. Donnons ici trois exemples pour une entrée sinusoïdale, commençant tous les trois par le même préambule

```
1  import numpy as np

3  R = 1e3          # en ohms
4  C = 1e-6         # en farad
5  tau = R*C        # en secondes

7  E0 = 2           # amplitude forcage en volts
8  f = 1e3          # fréquence, en hertz

10 dt = 2e-5        # pas de temps, en s
11 N = 500          # nbre de pas de temps
```

- **Premier exemple**

```
t = [n*dt for n in range(N)]      # tps, en secondes
e = [E0 * np.cos(2*np.pi*f*tn) for tn in t]
                                  # tension d'entrée, en volts
u = [None for n in range(N)]      # tension du condensateur, en V
                                  # tout est initialisé à None,
                                  # c-a-d "rien du tout"
u[0] = 0        # condition initiale u(0) = 0 V

for n in range(N-1):
    u[n+1] = u[n] + dt/tau * ( e[n] - u[n] )
```

- **Deuxième exemple**

```
t = np.linspace(0,(N-1)*dt,N)     # N points entre 0 et (N-1)*dt
e = E0 * np.cos(2*np.pi*f*t)      # numpy calcule terme à terme !
u = np.empty_like(e)              # initialisée à un tableau vide
                                  # de même forme que e
u[0] = 0

for n in range(N-1):
    u[n+1] = u[n] + dt/tau * ( e[n] - u[n] )
```

- **Troisième exemple**

```
tn = 0       # initialisation des valeurs courantes
un = 0

t = [tn]     # listes ne contenant que la valeur initiale
u = [un]

while tn < (N-1)*dt:
    tn += dt           # actualisation des valeurs courantes
    un += dt/tau * ( E0 * np.cos(2*np.pi*f*tn) - un )
    t.append(tn)       # nouveaux éléments ajoutés à chq pas de tps
    u.append(un)
```

Cette dernière implémentation est clairement moins aisément compréhensible que les deux précédentes ... mais elle renvoie le même résultat dans la liste `u` ! Lorsque le nombre de pas de temps est connu, utiliser une boucle **while** au lieu d'une boucle **for** est une mauvaise idée ... mais cela devient nécessaire si on ne connaît pas ce nombre à l'avance : imaginez que cherchiez à résoudre le PFD jusqu'à ce qu'un objet touche le sol, les itérations seraient à poursuivre « tant que $z(t) > 0$ ».

I.C - Systèmes différentiels

La résolution de système différentiels avec le schéma d'Euler ne pose aucune difficulté particulière.

Exemple : considérons la réaction chimique $2\,\mathrm{NO} + \mathrm{O_2} \longrightarrow 2\,\mathrm{NO_2}$, dont la loi de vitesse s'écrit

$$v = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}]\,.$$

On en déduit directement le système d'équations différentielles vérifiées par les concentrations des réactifs et du produit,

$$\begin{cases} -\dfrac{1}{2}\dfrac{\mathrm{d}[\mathrm{NO}]}{\mathrm{d}t} = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}] \\ -\dfrac{\mathrm{d}[\mathrm{O_2}]}{\mathrm{d}t} = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}] \\ \dfrac{1}{2}\dfrac{\mathrm{d}[\mathrm{NO_2}]}{\mathrm{d}t} = k\,[\mathrm{NO}]^2\,[\mathrm{O_2}] \end{cases}$$

que l'on transforme en relations de récurrence avec le schéma d'Euler,

$$\begin{cases} [\mathrm{NO}]_{n+1} = [\mathrm{NO}]_n - 2k\,\Delta t\,[\mathrm{NO}]_n^2\,[\mathrm{O_2}]_n \\ [\mathrm{O_2}]_{n+1} = [\mathrm{O_2}]_n - k\,\Delta t\,[\mathrm{NO}]_n^2\,[\mathrm{O_2}]_n \\ [\mathrm{NO_2}]_{n+1} = [\mathrm{NO_2}]_n + 2k\,\Delta t\,[\mathrm{NO}]_n^2\,[\mathrm{O_2}]_n \end{cases}$$

Il suffit ensuite simplement d'inclure ces trois relations de récurrence dans chaque itération de la boucle **for**.

I.D - Équations différentielles d'ordre 2

Le schéma d'Euler ne permet pas de résoudre directement des équations différentielles d'ordre 2. Pour contournier la difficulté, il est nécessaire de ruser en introduisant une deuxième fonction inconnue égale à la dérivée première, puis de se ramener à un système différentiel.

Exemple : considérons un oscillateur masse-ressort horizontal amorti par une force de frottement fluide, dont l'équation du mouvement s'écrit sous forme canonique

$$\ddot{x} + \frac{\omega_0}{Q}\,\dot{x} + {\omega_0}^2\,x = {\omega_0}^2\,x_{\text{éq}}\,.$$

On introduit alors $v = \dot{x}$, ce qui permet d'écrire l'équation sous la forme

$$\dot{v} + \frac{\omega_0}{Q}v + {\omega_0}^2\,x = {\omega_0}^2\,x_{\text{éq}}\,.$$

On peut alors appliquer le schéma d'Euler pour ces deux équations,

$$\begin{cases} \dfrac{x_{n+1} - x_n}{\Delta t} = v_n \\ \dfrac{v_{n+1} - v_n}{\Delta t} = -\dfrac{\omega_0}{Q}v_n + {\omega_0}^2\,(x_{\text{éq}} - x_n) \end{cases}$$

et en déduire les deux relations de récurrence à implémenter dans la boucle **for**,

$$\begin{cases} x_{n+1} = x_n + v_n\,\Delta t \\ v_{n+1} = v_n + \Delta t\left({\omega_0}^2\,(x_{\text{éq}} - x_n) - \dfrac{\omega_0}{Q}v_n\right) \end{cases}$$

I.E - À vous de jouer

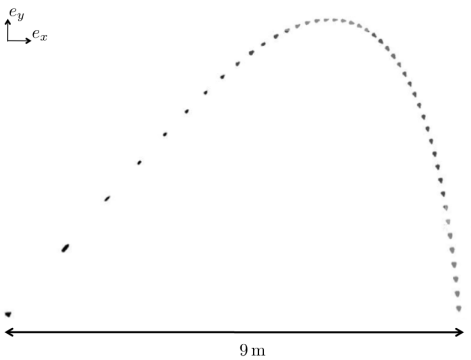
Écrire les relations de récurrence issues du schéma d'Euler pour résoudre numériquement les équations différentielles ci-dessous :

1 - Vidange d'un réservoir, relation de Torricelli : $\dfrac{\mathrm{d}h}{\mathrm{d}t} = \alpha\sqrt{2gh}$;

2 - Avec les notations des maths : $y' + y^2 = \mathrm{e}^{-\lambda t}$;

3 - Pendule simple : $\dfrac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + {\omega_0}^2 \sin\theta = 0$.

II - Trajectoire expérimentale



Notre étude se base sur un enregistrement vidéo de la trajectoire du volant, extrait de la thèse de Baptiste Darbois-Texier, réalisée au laboratoire PMMH de l'ESPCI, où des images du volant sont prises à la caméra rapide avec une période d'échantillonnage $T_\text{e} = 45\,\text{ms}$. Toutes ces images ont été superposées pour donner la chronophotographie ci-contre.

Les positions successives du volant ont été repérées à l'aide du logiciel ImageJ, puis exportées au format texte dans le fichier `pointage.txt` à récupérer sur l'ENT. Enregistrer ce fichier *dans le même dossier* que votre programme Python.

4 - Recopier les lignes suivantes dans votre fichier Python :

```
pointage = np.loadtxt("pointage.txt",skiprows=1)
Xexp = pointage[:,5]
Yexp = pointage[:,6]

x = Xexp - Xexp[0]
y = -(Yexp - Yexp[0])
```

Ouvrir le fichier `pointage.txt`, par exemple avec le bloc-notes, et afficher dans la console les deux listes `Xexp` et `Yexp`. En déduire ce que font les trois premières lignes proposées ci-dessus. Les lignes 5 et 6 permettent de corriger deux « défauts » de l'export réalisé par ImageJ : l'origine du repère est replacé à la position initiale du volant (au lieu d'un coin d'image), et l'axe des ordonnées réorienté vers le haut (alors qu'ImageJ l'oriente vers le bas).

5 - Afficher la trajectoire expérimentale, c'est-à-dire `y` en fonction de `x`. On pourra utiliser l'option d'affichage `marker='o'` pour représenter des points.

III - Modélisation

III.A - Équation du mouvement

Rappelons que l'objectif du TP est de comparer les trajectoires prévues par deux modèles de frottement à la trajectoire expérimentale pour déterminer lequel est le plus proche de la réalité. Ces deux modèles sont ceux d'une force de frottement linéaire, c'est-à-dire de norme proportionnelle à la vitesse du volant, ou quadratique, c'est-à-dire de norme proportionnelle au carré de sa vitesse :

$$\vec{F}_{\text{lin}} = -a\,\vec{v} \qquad \text{ou} \qquad \vec{F}_{\text{quadr}} = -bv\,\vec{v}$$

où on note comme d'habitude en physique $v = ||\vec{v}||$: contrairement aux SI, $\vec{v}$ **n'est pas** un vecteur unitaire. Les deux coefficients de frottement a et b sont phénoménologiques : ils ne sont pas connus a priori, et dépendent entre autres de la forme du volant et de la façon dont il se déforme au cours de son mouvement.

Par application du théorème de la résultante cinétique au volant dans le référentiel terrestre considéré galiléen, on obtient l'équation du mouvement sous la forme

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} + \frac{a}{m}\vec{v} = \vec{g} \qquad \text{ou} \qquad \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} + \frac{b}{m}v\,\vec{v} = \vec{g}\,.$$

La première est la classique équation linéaire du premier ordre, en revanche la deuxième est non-linéaire, ce qui la rend moins simple à résoudre analytiquement.

III.B - Estimation des coefficients de frottement

6 - Sur chacune des deux équations, montrer qu'après une phase transitoire la vitesse du volant tend vers une vitesse limite $\vec{V}_{\text{lim}}$ verticale. En déduire que la mesure de cette vitesse limite permet de déterminer a et b.

7 - Justifier que la ligne de code ci-dessous permet d'estimer la vitesse limite expérimentale V_{lim}.

```
1 Vlim = np.sqrt( (x[-2] - x[-1])**2 + (y[-2] - y[-1])**2 ) / Te
```

À partir de l'observation de la chronophotographie, indiquer ce qui limite la précision de l'estimation.

8 - En déduire numériquement les valeurs de a et b. Les conserver sous forme de deux variables globales `a` et `b`, au même titre que g, m ou encore T_{e}.

III.C - Estimation de la vitesse initiale

Pour pouvoir résoudre les équations du mouvement, il est nécessaire de connaître les composantes de la vitesse initiale $\vec{v}_0$ du volant, données par

```
1 V0x = (x[1] - x[0])/Te
2 V0y = (y[1] - y[0])/Te
```

IV - Résolution par la méthode d'Euler

IV.A - Avec frottement linéaire

9 - Établir les deux relations de récurrence permettant de résoudre l'équation du mouvement obtenue pour le modèle de frottement linéaire.

10 - Implémenter une fonction `euler_lin` résolvant l'équation du mouvement. Cette fonction prendra en argument le pas de temps `dt` et le nombre de points `N`, et renverra trois listes de même longueur `t` (temps), `Vx` et `Vy` (composantes de la vitesse à chaque instant). Toutes les autres variables physiques sont des constantes dans l'expérience, et donc définies comme des variables globales.

IV.B - Avec frottement quadratique

11 - Reprendre la démarche précédente pour écrire une fonction `position_quadr` jouant un rôle analogue avec le modèle quadratique.

IV.C - Comparaison des modèles

12 - En reprenant le schéma d'Euler, écrire une fonction `position` prenant pour arguments le pas de temps `dt` et deux listes `Vx` et `Vy` contenant les composantes du vecteur vitesse et renvoyant deux listes `X` et `Y` contenant les composantes du vecteur position du volant. On rappelle qu'à l'instant initial le volant se trouve par convention au point de coordonnées $(0, 0)$.

13 - En combinant les différentes fonctions, calculer puis représenter les trajectoires prévues par le modèle linéaire et par le modèle quadratique. On prendra `dt = Te/50` et `N = 50 * len(x)` : justifier l'expression de N. Conclure sur le meilleur modèle.

14 - L'auteur de la thèse indique une vitesse initiale $V_0 = 40\,\mathrm{m}\cdot\mathrm{s}^{-1}$ formant un angle $\theta_0 = 52°$ avec l'horizontale, et une vitesse limite $V_{\mathrm{lim}} = 6{,}7\,\mathrm{m}\cdot\mathrm{s}^{-1}$. Remplacer vos valeurs estimées par celles de l'auteur (**Attention !** commentez vos expressions, mais ne les effacez pas, pour pouvoir revenir en arrière.). Commenter l'influence de ce changement.

V - Résolution avec une bibliothèque

Nous allons maintenant comparer les résultats de la section précédente à ceux obtenus en utilisant une fonction déjà implémentée dans une bibliothèque de calcul scientifique : la fonction `odeint` du module `scipy.integrate`, que l'on importera par `from scipy.integrate import odeint`. Cette fonction utilise un schéma numérique nettement plus performant que celui d'Euler.

15 - La fonction `odeint` prend trois arguments obligatoires, nommés `func`, `y0` et `t` dans la documentation, ainsi qu'un certain nombre d'arguments optionnels que l'on n'utilisera pas. Ouvrir la documentation (`help(odeint)` dans la console) et comprendre ce qu'ils représentent et les informations physiques qu'ils contiennent.

16 - Réécrire les équations physiques dans une forme `odeint`-compatible.

17 - Coder les fonctions utiles à la résolution de l'équation par `odeint`. On pourra résoudre l'équation sur la vitesse, puis ensuite remonter à la position.

18 - Résoudre l'équation du mouvement avec `odeint` et comparer les résultats à ceux donnés par la méthode d'Euler. On s'intéressera en particulier à l'influence du pas de temps et des conditions initiales.

Étienne Thibierge, 17 novembre 2021, www.etienne-thibierge.fr