TD n° 13

Oscillateur harmonique

Dans ce TD, on va travailler sur un simple oscillateur harmonique, et se servir de ce modèle simple pour tester les limites de la méthode d'Euler.

On considère donc un oscillateur non amorti d'équation différentielle

$$\ddot{x} + \omega_0^2 x = 0 \tag{1}$$

avec $\omega_0 = \sqrt{\frac{k}{m}}$. De manière arbitraire, on va choisir $m = 1$ et $k = 1$ donc $\omega_0 = 1$ pour la suite.

1. Définissez m, k, ω_0 et la période T_0 dans un fichier Python.
2. À l'aide d'une méthode d'Euler, simulez ce système pour les CI : $x(0) = 1$ et $\dot{x}(0) = 0$, sur une durée de 5 périodes, avec comme pas temporel $\frac{T_0}{100}$. Commentez la courbe obtenue.
3. Tracez le portrait de phase $v(x)$; commentez.
4. À l'aide des tableaux déjà calculés par la méthode d'Euler, générez les tableaux correspondant à l'énergie cinétique E_c, à l'énergie potentielle E_p, et à l'énergie mécanique E_m. Tracez $E_c(t)$, $E_p(t)$ et $E_m(t)$: est-ce en accord avec ce qu'on attend ?
5. Retracez les courbes précédentes avec un pas de $\frac{T_0}{1000}$; commentez.
6. Le défaut majeur de la méthode d'Euler en mécanique est donc qu'elle fait augmenter artificiellement l'énergie. Pour résoudre cela, on a 3 solutions :
   - diminuer toujours plus le pas, mais cela va poser des problèmes d'arrondis dès qu'on dépasse environ $10^{-8}T_0$ (je n'entre pas dans les détails de cette limite)
   - utiliser une méthode sur le même principe que la méthode d'Euler mais plus précise, la méthode de Runge-Kutta
   - utiliser une méthode spécialement pensée pour conserver l'énergie mécanique, comme la méthode d'Euler semi-implicite (ou méthode de Euler-Cromer)

   Pour cette année, nous allons nous contenter d'utiliser la méthode de Runge-Kutta déjà implémentée dans la bibliothèque Scipy. On va apprendre à utiliser la fonction `scipy.integrate.odeint` qui est au programme[1]. La fonction `odeint` utilise le même principe que la méthode d'Euler : une récurrence. Dans cette récurrence, la règle générale pour toute variable f est : $f_{n+1} = f_n + pas \times f'(t_n, f_n)$:

ordre 1

$$\begin{cases} x_{n+1} = \boxed{x_n} + pas \times \boxed{\dot{x}(t_n, x_n)} \end{cases}$$

variable — dérivée

ordre 2

$$\begin{cases} x_{n+1} = \boxed{x_n} + pas \times \boxed{v_n} \\ v_{n+1} = \boxed{v_n} + pas \times \boxed{\ddot{x}(t_n, x_n, v_n)} \end{cases}$$

variables — dérivées

`odeint` travaille avec autant d'inconnues qu'on veut : on lui donnera une **liste** d'inconnues (même s'il n'y en a qu'une seule). Ensuite, on lui dit comment calculer la liste des dérivées correspondantes, et ensuite il itère tout seul la récurrence ; seulement, au lieu d'utiliser les formules de récurrence d'Euler, il utilise celles de Runge-Kutta, un peu plus compliquées mais beaucoup plus précises.

Voici donc les étapes à faire :

1. Mais qui est désormais jugée obsolète par les concepteurs de Scipy qui ont décidé de modifier complètement leur bibliothèque ; dans l'avenir la fonction sera `scipy.integrate.solve_ivp`, de syntaxe légèrement différente, mais d'idée générale identique.

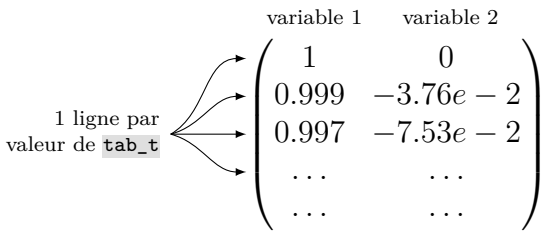
— **il faut toujours commencer par choisir la liste des inconnues, et leur ordre**, en l'écrivant par exemple comme commentaire dans le fichier Python. Par exemple, ici, on va choisir $[x, v]$, mais on aurait pu choisir l'ordre opposé : c'est uniquement une convention ;

— on crée alors une fonction `derivee(inconnues,t)` (qu'on peut nommer autrement, peu importe) qui prend obligatoirement comme arguments une **liste** d'inconnues et la variable (ici le temps), et qui renvoie la **liste** des dérivées des inconnues ; ici par exemple elle doit renvoyer $[\dot{x}, \dot{v}]$ c'est-à-dire $[v, -\omega_0^2 x]$;

— on crée une liste contenant les conditions initiales $[x(0), v(0)]$;

— on crée un tableau des instants pour lesquels on va simuler le système, le plus souvent avec `numpy.linspace(debut,fin,points)` ; pour commencer on met souvent 1000 points ;

— on appelle alors `odeint(derivee,ci,temps)` qui renvoie un tableau des solutions avec une inconnue par colonne ; on le stocke dans une variable ;



— on extrait les différentes inconnues à l'aide de la syntaxe `tableau[:,i]` qui permet d'extraire la colonne i du tableau (l'opérateur `:` signifiant : «tous les indices de ligne»).

Tracez alors la solution $x(t)$ obtenue par `odeint`, puis le portrait de phase ; tracez aussi l'énergie mécanique : elle augmente toujours, mais c'est beaucoup plus faible qu'auparavant.

Remarque : avec `odeint`, on ne précise pas le pas temporel ; il est déterminé automatiquement.

7. Augmentez le temps de simulation à 30 périodes ; observez que la courbe reste bien périodique.
8. On ajoute maintenant des frottements en introduisant dans l'équation un terme $\frac{\omega_0}{Q}\dot{x}$. Simulez avec `odeint` le système sur $30T_0$ avec $Q = 10$, et tracez les mêmes courbes que précédemment.
9. Reprenez avec $Q = 2$, puis $Q = \frac{1}{2}$, puis $Q = 0{,}1$.