1) Les bibliothèques utiles

Nous allons, grâce à Python, étudier et tracer des solutions numériques d'équations différentielles (non linéaires) utilisées en physique. Nous aurons besoin de trois bibliothèques :

— Numpy : bibliothèque de calcul numérique et matriciel, que nous importerons sous le nom `np` :

```
import numpy as np
```

— Scipy : bibliothèque de calcul scientifique "au dessus " de Numpy, qui gère les opérations de "haut niveau" (résolution d'équations différentielles, intégration, transformée de Fourier, de Laplace, etc...), Numpy s'occupant plutôt d'opérations "de base". Nous utiliserons uniquement la fonction `odeint` (pour "Ordinary Differential Equation Integrate"), importée grâce à :

```
from scipy.integrate import odeint
```

— Matplotlib : bibliothèque de tracé de courbes, surfaces, en 2D ou 3D, etc... Nous utiliserons essentiellement la sous-bibliothèque `pyplot` (pour le tracé en 2D), que nous importerons sous le nom `plt` grâce à :

```
import matplotlib.pyplot as plt
```

Nous en utiliserons plusieurs fonctions, pour tracer les courbes mais aussi nommer les figures, les axes, les sauvegarder, etc... Vous pouvez aussi reprendre l'aide mémoire `matplotlib` que je vous ai donné en début d'année.

Tous vos programmes devront donc commencer par ces trois instructions. Je vais maintenant détailler (à nouveau...)plus précisément les quelques fonctions de ces bibliothèques qui nous seront utiles.

a) Numpy.linspace

Une courbe, sur l'écran, n'est jamais qu'une succession de points, plus ou moins proches (plus proches si l'on veut une courbe plus précise) reliés par des petits segments de droite. Il faut donc commencer par créer une subdivision de l'intervalle considéré en un nombre raisonnable de points (par exemple 100). Numpy possède une fonction faite pour cela : `linspace`, dont l'utilisation a été vue (reprendre si nécessaire...)

```
t=np.linspace({debut},{fin},{nombre de points})
```

Attention : si l'on veut découper par exemple l'intervalle $[0,1]$ en dix intervalles $[0,0.1]$, $[0.1,0.2]$, ..., $[0.9,1]$, il y a onze points ! (c'est le célèbre problème des piquets et des barrières), et donc écrire :

```
t=np.linspace(0,1,11)
```

Le tableau `t` est un objet de type `array` : c'est en fait une matrice, à une ligne et 11 colonnes. Le type `array`, défini dans Numpy, est similaire au type `list` de python mais :

— L'opération "+" n'est pas la concaténation, mais l'addition des matrices (en particulier, elles doivent être de même taille).

— Tous les éléments doivent être de même type numérique (que des entiers, ou que des flottants,...).

La fonction `array` permet de transformer une `list` en `array`, et la fonction `list` permet de transformer un `array` en `list`. On aurait donc pu créer le tableau à la main grâce à la commande :

```
t=array([float(k)/10 for k in range(11)])
```

Numpy redéfinit aussi toutes les fonctions mathématiques standard, et il est conseillé de les utiliser, plutôt que celles du module math. On écrira donc `np.sin`, `np.log`, etc... Ces fonctions peuvent être appliquées à un `array` : dans ce cas, la fonction est appliquée à tous les éléments de la matrice. On pourra utiliser `np.pi` (vaut π...).

b) Scipy.integrate.odeint

L'outil essentiel de calcul numérique est la fonction `odeint` qui résout les équations différentielles **du premier ordre uniquement, et sous forme résolue** $y' = F(y,t)$. Elle fonctionne de la manière (simplifiée) suivante :

```
y=odeint({F},{y0},{t})
```

où :

— F est une fonction qui prend deux variables (même si l'équation est autonome) : y et t, qui doit être définie "à la main" (grâce à `def F(y,t):`). Elle peut avoir éventuellement d'autres paramètres, nous verrons cela plus tard.

— y_0 représente la condition initiale : c'est la valeur que doit prendre y au premier point du tableau `t`.

— t est un `array` (ou une liste, mais bon) indiquant les points où doit être évaluée la fonction solution.

Elle renvoie un `array`, ici `y`, de la même longueur que le tableau `t`.

Par exemple, pour résoudre le problème de Cauchy $\begin{cases} y' = 3\sin y - t \\ y(1) = 5 \end{cases}$ sur $[1,6]$, on peut procéder de cette façon :

```
t=np.linspace(1,6,101)

def F(y,t):
        return 3*np.sin(y)-t

y=odeint(F,5,t)
```

`y` contiendra alors un tableau de 101 éléments, qui sont les images des éléments de t. On aura forcément $y[0] = 5$ (car $t[0] = 1$) : c'est la condition initiale.

c) Cas particulier des équations du second ordre

`odeint` ne résout que les équations du premier ordre... Mais comment vais-je faire pour résoudre celles du second ordre ? Il faut en fait, comme on l'a vu en maths, commencer par transformer l'équation différentielle.

Supposons qu'elle soit sous forme résolue (de toutes façons remarquez que pour que le programme fonctionne, il faut que la solution soit unique, et donc, d'après le théorème de Cauchy, que l'équation soit sous forme résolue) $y'' = F(y, y', t)$. On pose $Y = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$. On a donc

$$Y'(t) = \begin{pmatrix} y'(t) \\ y''(t) \end{pmatrix} = \begin{pmatrix} y'(t) \\ F(y(t), y'(t), t) \end{pmatrix} = \tilde{F}(Y(t), t)$$

où :

$$\tilde{F} : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2((a, b), t) \to (b, F(a, b, t))$$

Par exemple, l'équation du pendule simple $y'' = -\sin y$, couplée avec les conditions de Cauchy $y(0) = 1$ et $y'(0) = -2$, sera vue comme une équation du premier ordre, $Y' = \tilde{F}(Y, t)$ avec :

$$\tilde{F}((a, b), t) = (b, -\sin a) \qquad Y(0) = (y(0), y'(0)) = (1, -2)$$

On résoudra donc cette équation (par exemple sur $[0, 3]$) en python avec les commandes suivantes :

```
t=np.linspace(0,3,101)

def F(y,t):
        f=y[0]; fp=y[1]
        fs=-np.sin(f)
        return [fp,fs]

Y0=[1,-2]
Y=odeint(F,Y0,t)
```

J'ai noté `f=y[0]` la fonction, `fp=y[1]` sa dérivée (`fp` pour "f prime"), et `fs` sa dérivée seconde, calculée grâce à l'équation (`fs` pour "f seconde").

Le résultat, `Y` sera une matrice de 101 lignes et deux colonnes : la première colonne, c'est-à-dire `Y[:,0]` est la fonction y, et la deuxième `Y[:,1]` la fonction y'.

d) Matplotlib.pyplot

Une fois calculée la solution, il reste à l'afficher à l'écran. Pyplot est là pour vous ! Il y a tout un tas de commandes pour modeler le graphique à votre sauce :

— Principalement, la commande `plot`, qui fonctionne ainsi :

```
plot({abscisses},{ordonn\'ees})
```

où *abscisses* est un tableau contenant les abscisses et *ordonnées* un tableau (de même taille !) contenant... les ordonnées ! A noter que plusieurs commandes `plot` successives traceront les courbes sur le même graphique, dans des couleurs différentes (dans l'ordre : bleu - vert - rouge - bleu ciel - violet - jaune - noir, puis ça recommence).
On peut ainsi tracer le *portrait de phases* d'une solution, c'est-à-dire la dérivée en fonction de la valeur de la fonction (c'est-à-dire $y'(t)$ non pas en fonction de t, mais en fonction de $y(t)$).

— Des *labels* (les fonctions parlent d'elle mêmes...) :

```
plt.title(Nom de la figure)
plt.xlabel(Nom de l'axe des x)
plt.ylabel(Nom de l'axe des y)
plt.grid()\                {\it trace une grille}
```

— Afficher la figure, ou l'effacer :

```
plt.show()
plt.clf()
```

Si l'on veut graduer l'axe des x ou des y de manière plus précise, pour mesurer des quantités, on pourra utiliser les commandes `plt.xticks` et `plt.yticks`. Par exemple :

```
plt.xticks(np.linspace(0,10,21))
```

produira une grille graduée tout les .5 de 0 à 10.

e) Faisceaux de courbes

Souvent, l'équation différentielle comporte un paramètre (par exemple, ω dans $y'' + \omega^2 y = 0$). La fonction `F` prendra donc, en plus de `y` et `t` d'autres paramètres, en nombre quelconque.

Supposons donc que l'on veuille tracer sur le même graphique les solutions de $y'' + \omega^2 y = 0$, pour ω entre 0 et 1, avec un pas de 0.1. On commence par écrire la fonction $F(y, t, omega)$ correspondant à l'équation différentielle

```
def F(y,t,omega):
        f=y[0]~;~fp=y[1]
        fs=-omega*omega*f
        return [fp,fs]
```

Il faut donc expliquer à `odeint` que la fonction `F` possède un paramètre de plus : on rajoute donc un *tuple*, c'est-à-dire la liste des paramètres entre parenthèses (ici qui n'aura qu'une variable, d'où la virgule "étrange") en plus. Voici donc la fonction qui résout l'équation et trace la courbe :

```
def trace(omega):
        t=np.linspace(0,20,100)
        Y=odeint(F, [1,0],t,(omega,))
        plt.plot(t,Y[:,0])
```

Pour finir le tracé, il suffit donc de faire une boucle et d'appeler la fonction `trace` :

```
def trace_all():
for i in range(11):
                omega=i*.1
                trace(omega)
        plt.show()
```

On peut alors avoir autant de variables que l'on veut, mais il faut bien faire attention à ce qu'elles soient dans le même ordre dans `F` et dans `odeint`.

2) Applications diverses à des ED non linéaires

a) Le pendule simple

On considère le pendule simple, non amorti (on néglige les forces de frottement). On note θ l'angle fait par le pendule avec la verticale, en fonction du temps t. Après avoir fait un peu de physique, on obtient l'équation du pendule simple :

$$\ddot{\theta} + \omega^2 \sin\theta = 0$$

où $\omega = \sqrt{\frac{g}{l}}$. Pour les calculs numériques, on prendra $\omega = 1$.

1. Tracer la courbe de θ en fonction du temps (pour t de 0 à 6π). On supposera qu'à $t = 0$, le pendule est lâché à vitesse nulle à l'horizontale ($\theta_0 = \frac{\pi}{2}$, $\dot{\theta_0} = 0$).
2. Tracer le portrait de phases.
3. Tracer sur le même graphique, en fonction du temps, l'énergie cinétique ($E_c = \frac{1}{2}\dot{\theta}^2$), l'énergie potentielle ($E_p = 1 - \cos\theta$), et la somme des deux (on vérifiera qu'elle est constante !).
4. Reprendre la question 2. avec différentes valeurs de $\theta_0 \in]0;\pi[$ (toujours avec une vitesse initiale nulle). Que remarque-t-on si θ_0 est petit ? S'il est grand ?
5. Tracer, sur le même graphique, les neuf diagrammes de phase correspondant à $\theta_0 = \frac{k\pi}{10}$ avec $k \in [1, 9]$. On utilisera la méthode décrite dans le chapitre "faisceau de courbes" : on pourra créer une fonction `trace(theta)` qui trace la courbe en fonction de `theta` (mais il est inutile ici de faire des paramètres supplémentaires, puisqu'on ne modifie pas l'équation elle-même).
6. Rajouter sur le portrait de phase précédent le cas correspondant à $\theta_0 = \pi$ (par contre, pour avoir la courbe complète, il faut prendre $t \in [0, 120\pi]$, et calculer 5000 points). Après avoir remarqué que la fonction constante $y(t) = \pi$ est solution du problème de Cauchy pour $\theta_0 = \pi$ et $\dot{\theta}_0 = 0$, comment expliquer que le pendule bouge pour $\theta_0 = \pi$? Pourquoi faut-il étudier ce cas sur un temps très long ?

b) Oscillateur anharmonique

On se place au voisinage d'une position d'équilibre stable. L'énergie potentielle se développe, très proche de q_{eq} selon la formule du cours : $E_p(q) = E_p(q_{eq}) + \frac{1}{2}k_{eff}(q - q_{eq})^2$. Si on s'éloigne un peu de q_{eq}, le terme suivant du développement de Taylor n'est plus négligeable, et cela conduira à $E_p(q) = E_p(q_{eq}) + \frac{1}{2}k_{eff}(q - q_{eq})^2 + \frac{1}{6}Q(q - q_{eq})^3$; la conservation de l'énergie mécanique conduit à l'ED en $X = q - q_{eq}$:

$$\ddot{X} + \frac{k}{m}X + \frac{Q}{2m}X^2 = 0$$

Ecrire la fonction `F(y,t)` décrivant l'équation différentielle précédente, puis la fonction `trace(omega)` qui trace la solution de l'équation pour `omega`.

c) Pour aller plus loin : Oscillateur amorti en régime sinusoïdal forcé

On considère l'équation différentielle suivante, modélisant un oscillateur amorti en régime forcé,

$$\ddot{\theta} + \frac{\omega_0}{Q}\dot{\theta} + \omega_0^2\theta = a.\sin\omega t$$

où Q, ω, ω_0 et a sont des constantes réelles positives :
— ω_0 est la pulsation propre du système.
— Q est le facteur de qualité.

— ω est la pulsation du forçage.
— a est l'amplitude du forçage.

1. Ecrire la fonction `F(y,t,omega_0,Q,omega,a)` décrivant l'équation différentielle précédente, puis écrire deux fonctions :
   — `trace(omega_0,Q,omega,a,y0,yp0)` qui trace la solution
   — `trace_phase(omega_0,Q,omega,a,y0,yp0)` qui trace son portrait de phase
   en fonction de `omega_0`, `omega`, `Q` et `a` avec comme conditions initiales $y(0) =$ `y0` et $y'(0) =$ `yp0`. On tracera les solutions sur l'intervalle $[0, 20]$. Dans la suite, on n'utilisera que ces deux fonctions !
2. Etude d'un cas particulier : Tracer la solution, puis son portrait de phase, avec $\omega_0 = 1$, $\omega = 2$, $Q = 1$, $a = 1$ et les conditions initiales $y(0) = 1$ et $y'(0) = 0$. Que remarque-t-on ?
3. Etude de l'influence de ω :
   (a) Tracer, sur un même graphique, les solutions de l'équation avec les paramètres et conditions initiales précédentes, mais en faisant varier ω de 0 à 2, avec un pas de .5. Qu'observe-t-on si $\omega = 0$? Lorsque ω augmente ?
   (b) On veut tracer, dans l'espace des phases, uniquement le régime permanent (pour t assez grand). Pour ce faire, il suffit de faire résoudre l'équation différentielle, par exemple sur $[0; 100]$, mais de la tracer uniquement sur $[50, 100]$. Ecrire une fonction `trace_phase_t(omega_0,Q,omega,a,y0,yp0)` qui fait cela (il suffit de recopier `trace_phase` et de modifier la dernière ligne : en effet, si `n=len(t)` et qu'on veut tracer uniquement la deuxième moitié du portrait de phase, il suffit d'écrire `plt.plot(y[n:,0],y[n:,1])`, grâce au *slicing*...).
   Tracer les portraits de phase en régime permanent pour ω allant de $0,5$ à 2, par pas de $0,25$. Que peut-on en déduire ?
4. Etude des autres paramètres : On prend comme paramètres de base $\omega_0 = 1$, $\omega = 1.5$, $Q = 1$, $a = 1$, $y(0) = 1$ et $y'(0) = 1$. On va faire varier un seul des paramètres, les autres étant constants. Il est conseillé d'user et d'abuser du "copier-coller"...
   (a) Tracer les solutions pour a variant de 0 à 5 par pas de 1. Que peut-on en déduire ?
   (b) Tracer les solutions pour Q variant de 1 à 5 par pas de 1. Que peut-on en déduire ?
   (c) Tracer les solutions pour $y(0)$ variant de 0 à 5 par pas de 1. Que peut-on en déduire ?
   (d) Tracer les solutions pour $y'(0)$ variant de 0 à 5 par pas de 1. Que peut-on en déduire ?
   (e) On peut tester des valeurs négatives pour Q : Tracer les solutions pour Q variant de -1 à -5 par pas de -1. Que se passe-t-il ?