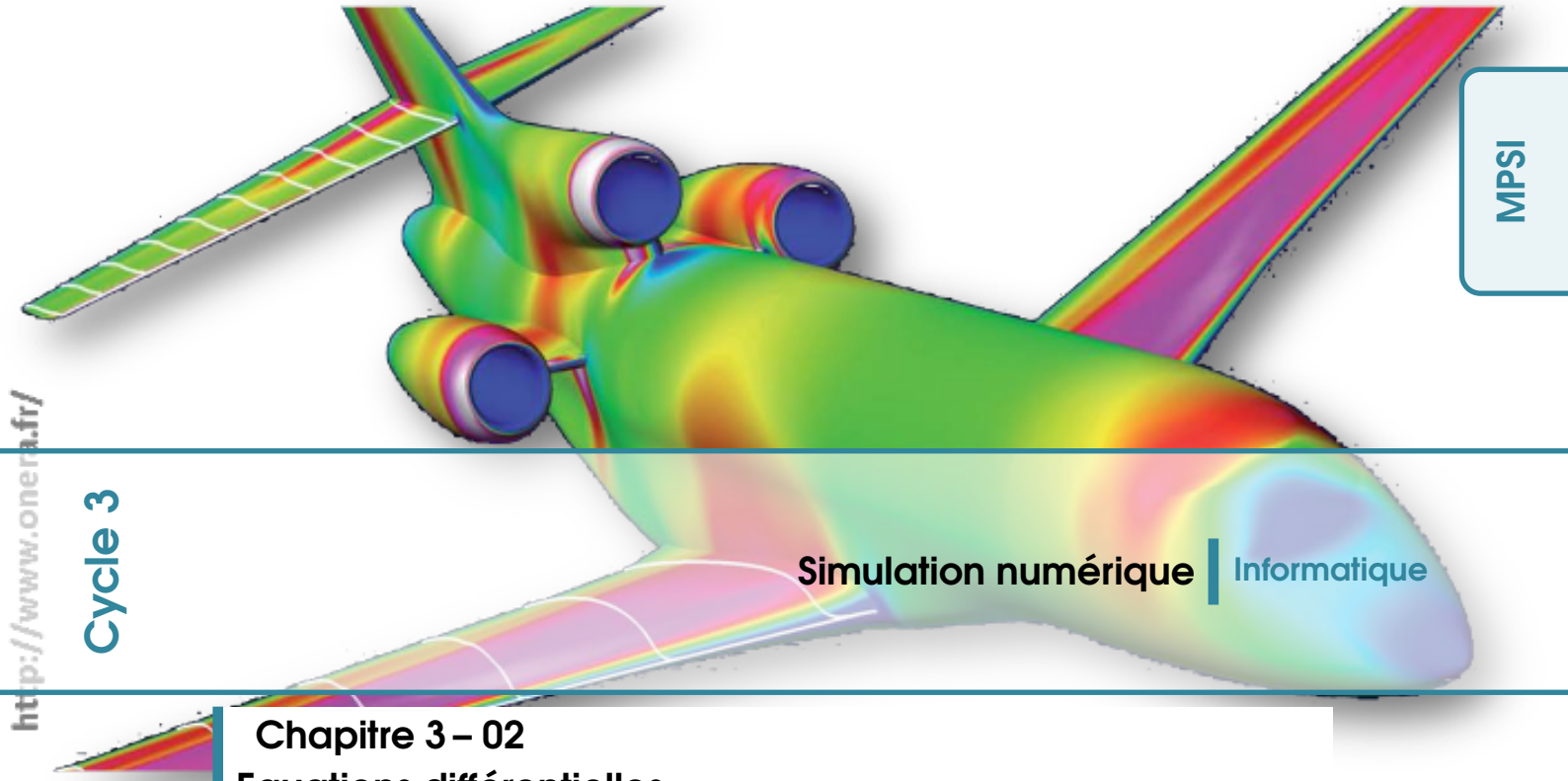

http://www.onera.fr/

Chapitre 3 – 02
Equations différentielles

12 Fevrier 2020

Cours

Savoirs et compétences :

- *SN.C2 : Étudier l'effet d'une variation des paramètres sur le temps de calcul, sur la précision des résultats, sur la forme des solutions pour des programmes d'ingénierie numérique choisis, tout en contextualisant l'observation du temps de calcul par rapport à la complexité algorithmique de ces programmes*
- *SN.S3 : Problème dynamique à une dimension, linéaire ou non. Méthode d'Euler.*

Émilien Durif – Sylvaine Kleim
Xavier Pessoles
MPSI

CC BY NC SA

1 Cadre mathématique.

1.1 Problème de Cauchy.

Nous nous intéresserons au probème suivant, dit **de Cauchy**

$$y' = F(y,t), \text{ avec la condition initiale } \quad y(t_0) = y_0 \tag{1}$$

où :
- y est l'inconnue ;
- y est une fonction dérivable définie sur un intervalle I de $\mathbb{R}$;
- y et F sont à valeurs dans $E = \mathbb{R}^n$ ou $E = \mathbb{C}^n$ ($n \in \mathbb{N}^*$) ;
- F est définie sur $E \times I$;
- $t_0 \in I$ et $y_0 \in E$.

R En mathématiques et en physique, l'habitude est plutôt de considérer les équations différentielles sous la forme $y' = F(t,y)$ et non $y' = F(y,t)$.
Mais en Python, la fonction `odeint`, que nous verrons en fin de chapitre, utilise l'écriture $y' = F(y,t)$. Par souci de simplicité, nous nous conformerons à cette écriture dans ce cours.

1.2 Notion de solution.

On appelle **solution** du problème précédent tout couple (y, J) où :
- J est un sous-intervalle de I contenant t_0 ;
- $y : J \to E$ est une fonction dérivable vérifiant $y(t_0) = y_0$ et $\forall t \in J \quad y'(t) = F(y(t), t)$.

On dit qu'une telle solution est **maximale** s'il est impossible de prolonger y en une solution sur un intervalle strictement plus grand que J.

1.3 Premier exemple : équation linéaire.

Une solution de l'équation (1) avec la condition initiale $y(1) = 2\mathrm{e}$ où $F : \begin{array}{ccc} \mathbb{R}\times\mathbb{R} & \to & \mathbb{R} \\ (y,t) & \mapsto & y \end{array}$ est $\begin{array}{ccc} [0,1] & \to & \mathbb{R}, \\ t & \mapsto & 2\mathrm{e}^t \end{array}$.
Cependant, cette solution n'est pas maximale car elle est prolongeable (par exemple) par $\begin{array}{ccc} [0,17] & \to & \mathbb{R}, \\ t & \mapsto & 2\mathrm{e}^t \end{array}$. La seule solution maximale est dans ce cas $\begin{array}{ccc} \mathbb{R} & \to & \mathbb{R}, \\ t & \mapsto & 2\mathrm{e}^t. \end{array}$

1.4 Second exemple : équation non linéaire.

On cherche à résoudre sur $\mathbb{R}$ l'équation $y' = \frac{3}{7}y^3$ avec la condition initiale $y(0) = \frac{1}{36}$.
- Il n'y a pas de solution définie sur $\mathbb{R}$ tout entier (même en changeant la condition initiale, sauf pour une condition initiale $y(t_0) = 0$) ;
- il y a une unique solution maximale : $t \mapsto \sqrt{\frac{7}{252-6t}}$;
- cette solution maximale est définie sur $]-\infty, 42[$.

1.5 Théorème de Cauchy-Lipschitz.

Sous des hypothèses raisonnables sur F : pour chaque problème de Cauchy, il existe une unique solution maximale.

Attention : l'intervalle sur lequel une solution maximale est définie peut dépendre de la condition initiale.

1.6 En pratique

Bien souvent, on considère en pratique le cas $I = [a,b]$ (où $a, b \in \mathbb{R}$ avec $a < b$) et les solutions seront définies sur I tout entier.

On se placera dans ce cadre pour la suite de ce cours. On supposera que la condition initiale est donnée en a et on notera y_0 la valeur initiale de y en a. On note y l'unique solution maximale du problème de Cauchy.

2 La méthode d'Euler.

2.1 Principe général.

Nous avons vu en mathématiques que la résolution d'une équation différentielle linéaire du premier ordre se ramenait au calcul d'une primitive. Mais il existe des fonctions pour lesquelles nous ne savons pas calculer de primitive. Nous sommes donc incapables de résoudre ces équations de manière exacte. Quand il s'agit d'équations différentielles d'ordre supérieur ou non linéaires, la situation est encore moins favorable.

Que faire s'il nous faut vraiment une solution à une telle équation différentielle ? Une possibilité est d'essayer d'obtenir une **approximation numérique** d'une solution exacte. La méthode d'Euler est la méthode classique la plus simple pour faire cela. Le principe assez élémentaire de cette méthode est le suivant.

On cherche à déterminer une approximation sur $[a,b]$ de la fonction $f:[a,b]\to\mathbb{R}$ vérifiant $\forall t\in[a,b],\ f'(t)=F(f(t),t)$ et $f(a)=y_0$.

On commence par choisir un **pas** $h=\frac{b-a}{n}$, où $n\in\mathbb{N}^*$, et l'on subdivise l'intervalle $[a,b]$ en n segments, en posant pour chaque $k\in[\![0,n]\!]$: $t_k=a+kh=a+k\dfrac{b-a}{n}$.

Ainsi, $t_0=a$ et $t_n=b$. On dit que $(t_0,\dots,t_n)$ est une subdivision régulière du segment $[a,b]$, avec $n+1$ points ou n segments (ou morceaux).

On va ensuite construire $y_0,\dots,y_n$. L'approximation de la courbe de f sera alors la ligne brisée reliant les points de coordonnées (t_k,y_k).

Le premier point (t_0,y_0) est donné par la condition initiale.

Pour chaque $k\in[\![0,n[\![$, si les approximations $y_0,\dots,y_k$ aux temps $t_0,\dots,t_k$ sont construites, on approche la solution passant par le point de coordonnées (t_k,y_k) par sa tangente. Comme la pente de cette tangente est $F(y_k,t_k)$, l'équation de cette tangente est alors $y=y_k+F(y_k,t_k)(t-t_k)$.

L'approximation de la solution au temps t_{k+1} est alors (voir la figure 1)

$$\begin{aligned}y_{k+1}&=y_k+F(y_k,t_k)(t_{k+1}-t_k)\\&=y_k+hF(y_k,t_k).\end{aligned}$$

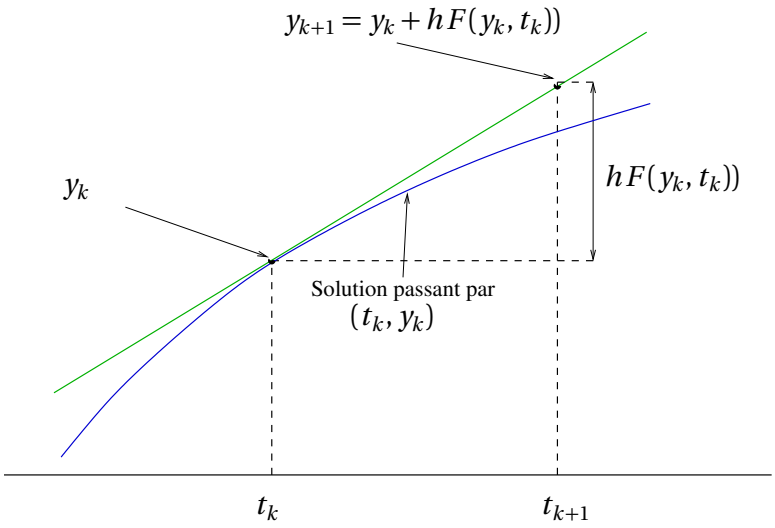

FIGURE 1 – Illustration du principe de la méthode d'Euler.

Par récurrence, on construit ainsi, de proche en proche, les réels y_0, y_1, y_2, $\cdots$, y_n qui sont des approximations des $f(t_k)$. Chaque approximation y_{k+1} est construite à partir de l'approximation précédente, y_k.

La ligne brisée joignant les points (t_k,y_k) est le graphe d'une fonction, qui est elle-même une approximation de la solution exacte f.

2.2 Premier exemple, une équation linéaire.

On cherche à approcher la solution (maximale) sur $[0,3]$ de $y'=y$ avec condition initiale $y(0)=1$.

R Vous savez bien entendu que c'est $t\mapsto \mathrm{e}^t$.

Numériquement, on regarde successivement les approximations obtenues par la méthode d'Euler avec les paramètres $n=3$, $n=30$ et $n=100$, soit des pas respectivement de 1, 1/10 et de 3/100 (voir la figure 2).

2.3 Quelques remarques

On peut conjecturer que

1. quand le pas diminue, l'approximation s'améliore ;
2. la méthode d'Euler ne corrige pas les erreurs d'approximation mais au contraire les propage en les augmentant.

En fait :

- le premier point est vrai mathématiquement ;

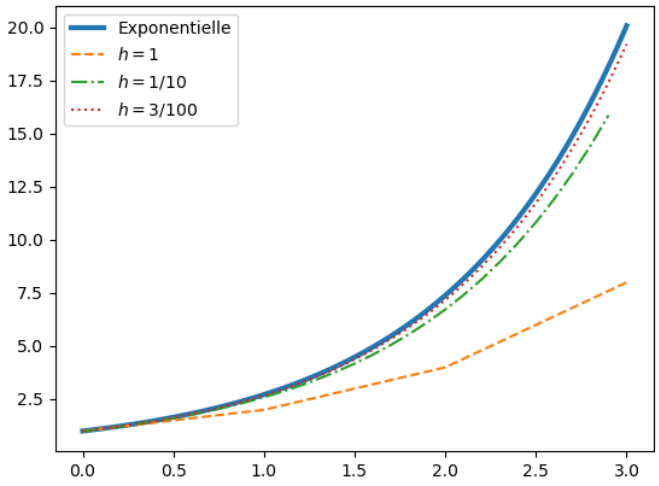

FIGURE 2 – Approximations de la solution du problème de Cauchy $y'=y$, $y(0)=1$ sur $[0,3]$.

- le second dépend des équations considérées.

Informatiquement :

- si le pas est grand, l'approximation est mauvaise ;
- si le pas est petit, elle est longue à calculer ;
- s'il est vraiment trop petit, les erreurs d'arrondis causent en plus d'autres problèmes.

3 Mise en œuvre.

3.1 Implantation de la méthode d'Euler.

On traduit naïvement la méthode d'Euler. Cette fonction est extrêmement importante, vous devez savoir la reproduire.

```
def euler(F, a, b, y0, h):
    """Solution de y'=F(y,t) sur [a,b], y(a) = y0, pas h"""
    y = y0
    t = a
    y_list = [y0] # la liste des valeurs renvoyées
    t_list = [a] # la liste des temps
    while t+h <= b:
        # Variant : floor((b-t)/h)
        # Invariant : au tour k, y_list = [y_0,...,y_k], t_list = [t_0,...,t_k]
        y = y + h * F(y, t)
        y_list.append(y)
        t = t + h
        t_list.append(t)
    return t_list, y_list
```

R Il est primordial d'utiliser ici l'écriture `y = y + h * F(y, t)` au lieu de `y += h * F(y, t)`, qui ne donne pas le résultat escompté dans le cas vectoriel.

R La variante écrite au dessus correspond au cas où le pas h nous est donné. Si le nombre de subdivisions n nous est donné à la place, on pourra préférer écrire la fonction avec une boucle `for`, en utilisant la fonction `linspace` du module `numpy`.

3.2 Exemple et représentation graphique.

Problème de cinétique chimique : on s'intéresse à une réaction chimique faisant disparaître un réactif A d'une solution.

On suppose que la vitesse de disparition de A est proportionnelle à sa concentration (réaction d'ordre 1). La concentration de A suit donc l'équation

$$\frac{\mathrm{d}[A]}{\mathrm{dt}} = -\alpha[A],$$

où α est une constante s'exprimant en s^{-1}.

L'équation est bien de la forme

$$[A]' = F([A],t) \qquad \text{où } F:(y,t)\mapsto -\alpha y.$$

On suppose $\alpha = 1$ (unité : s^{-1}) et, au temps $t = 0$, $[A] = 1$ (unité : $mol.l^{-1}$). On étudie l'évolution sur l'intervalle de temps $[0,6]$ (unité : s).

Le code suivant permet de tracer (voir figure 3) simplement la solution approchée par la méthode d'Euler pour le pas de $h = \frac{1}{10}$ (on considère que la fonction `euler` a déjà été écrite).

```
import matplotlib.pyplot as plt

A0 = 1 # mol
alpha = 1 # s**-1
t0,t1,h = 0,6,.1

def F(y,t) :
    return -alpha*y

t_list,A_list = euler(F,t0,t1,A0,h)
plt.plot(t_list,A_list)
plt.xlabel('Temps ($s$)')
plt.ylabel('Concentration ($mol$)')
plt.grid()
plt.savefig('reaction_ordre1.png')
```

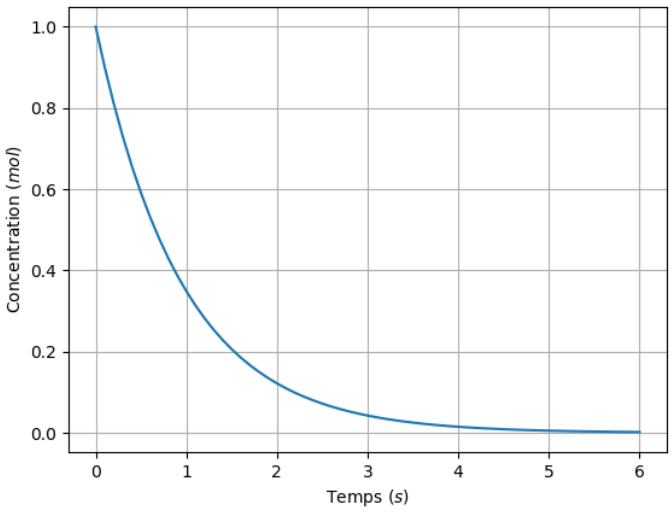

FIGURE 3 – Tracé de la trajectoire.

4 Équations d'ordres supérieurs

4.1 Dérivation de fonctions à valeurs vectorielles

En mathématiques et en physique, vous avez pris l'habitude de dériver des fonctions à valeurs complexes (et à variables réelles). Par exemple, si $z:\mathbb{R}\to\mathbb{C}$ a pour partie réelle x et pour partie imaginaire y, alors si z est dérivable

on a $z' = x' + iy'$.

On procède de même pour dériver des fonctions à valeurs vectorielles (du moins, dans un espace de dimension finie !) : on dérive composante par composante. Ainsi, si une fonction $X : \mathbb{R} \to \mathbb{R}^2$ a pour composantes x et y, **i.e.** $X : t \mapsto \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$, alors si X est dérivable, on a $X' : t \mapsto \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix}$.

4.2 Vecteurs numpy

La bibliothèque numpy fournit le type `array` (vecteur, en anglais), qui permet d'effectuer directement du calcul vectoriel. Les vecteurs sont à distinguer des listes (même si ces deux types partagent certaines propriétés, comme le tranchage). Notamment, les vecteurs sont des données homogènes de taille fixée.

```
>>> from numpy import array
>>> v = array([1.,2.])
>>> v
array([1., 2.])
>>> w = array([-5,2.5])
>>> v + 3*w
array([-14. ,   9.5])
```

On peut accéder aux coordonnées d'un vecteur avec les notations habituelles.

```
>>> v[0],v[1]
(1.0, 2.0)
```

4.3 Vectorialisation d'équation différentielle.

On donne ici l'exemple d'une équation différentielle d'ordre 2 (le cas le plus courant que vous rencontrerez), cela se généralise sans peine aux ordres supérieurs.

On considère un problème de Cauchy d'ordre 2 : $y''(t) = f(y(t), y'(t), t), \quad y(t_0) = y_0, y'(t_0) = z_0$, défini sur un intervalle de temps I et où $t_0 \in I$.

L'idée est alors de **vectorialiser** cette équation, pour obtenir une équation **d'ordre 1** sur une variable vectorielle de **dimension 2**. Pour cela, on considère la variable $X(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$.

On a alors $X'(t) = \begin{pmatrix} y'(t) \\ y''(t) \end{pmatrix} = \begin{pmatrix} y'(t) \\ f(y(t), y'(t), t) \end{pmatrix}$. Avec F : $\left\{ \begin{array}{ccc} \mathbb{R}^2 \times I & \longrightarrow & \mathbb{R}^2 \\ \left(\begin{pmatrix} a \\ b \end{pmatrix}, t \right) & \longmapsto & \begin{pmatrix} b \\ f(a,b,t) \end{pmatrix} \end{array} \right.$,

on obtient donc bien le problème de Cauchy équivalent : $X'(t) = F(X(t), t) \quad$ et $\quad X(t_0) = \begin{pmatrix} y_0 \\ z_0 \end{pmatrix}$.

R On notera bien que la condition initiale de ce problème de Cauchy est vectorielle.

4.4 Exemple : système ressort-amortisseur, tracé de la trajectoire et du portrait de phase.

Considérons l'équation d'un système ressort-amortisseur : $m\,y'' = -c\,y' - k\,y$, sur l'intervalle de temps $[0,20]$ (unité : s), avec les conditions initiales $y(0) = 1$ (unité : m) et $y'(0) = 0$ (unité : $m.s^{-1}$).

On vectorialise l'équation en considérant la variable $X(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$, l'équation s'écrit alors $X'(t) = F(X(t), t)$ avec

$F : \left\{ \begin{array}{ccc} \mathbb{R}^2 \times [0,20] & \longrightarrow & \mathbb{R}^2 \\ \left(\begin{pmatrix} a \\ b \end{pmatrix}, t \right) & \longmapsto & \begin{pmatrix} b \\ -\dfrac{k\,a + c\,b}{m} \end{pmatrix} \end{array} \right.$ et la condition initiale $X(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

On considère les constantes suivantes : $m = 250$ (unité : kg), $k = 1000$ (unité : $kg.s^{-2}$) et $c = 100$ (unité : $kg.s^{-1}$).

Le script suivant permet d'appliquer la méthode d'Euler à l'équation vectorialisée (on suppose la fonction `euler` déjà écrite).

```
y0 = 1 # m
yp0 = 0 # m * s**-1
t0, t1, h = 0, 20, .01 # s
m = 250 # kg
c = 100 # kg * s**-1
k = 1000 # kg * s**-2
```

```
def F(X,t) :
    y,yp = X
    ypp = -(k*y + c*yp)/m
    return array([yp,ypp])

X0 = array([y0,yp0])

t_list, X_list = euler(F,t0,t1,X0,h)
```

ATTENTION ! La variable `X_list` est une liste de vecteurs ! On peut récupérer les positions et les vitesses comme suit.

```
y_list = [X[0] for X in X_list]
yp_list = [X[1] for X in X_list]
```

On trace ensuite simplement la trajectoire (position en fonction du temps, voir figure 4) et le portrait de phase (courbe (position,vitesse), voir figure 5) comme suit.

```
plt.clf()
plt.plot(t_list,y_list)
plt.xlabel('Temps ($s$)')
plt.ylabel('Position ($m$)')
plt.grid()
plt.savefig('oscillateur_amorti_trajectoire.png')

plt.clf()
plt.plot(y_list,yp_list)
plt.xlabel('Position ($m$)')
plt.ylabel('Vitesse ($m.s^{-1}$)')
plt.grid()
plt.savefig('oscillateur_amorti_phase.png')
```

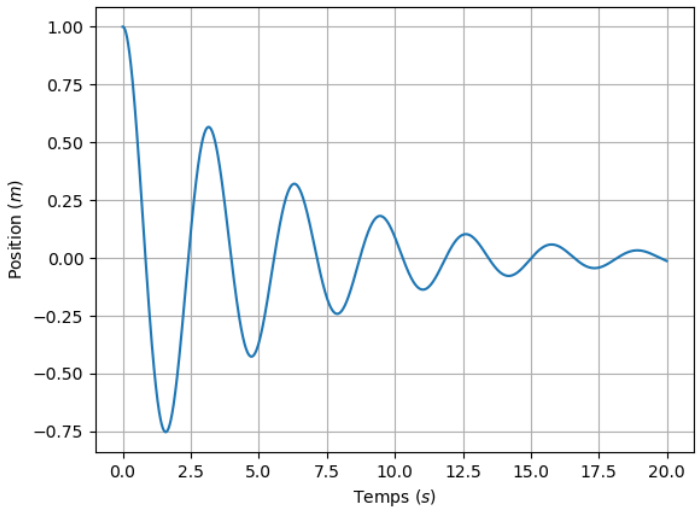

FIGURE 4 – Oscillateur amorti, trajectoire par la méthode d'Euler.

5 Utilisation de scipy

Nous ne sommes pas les seuls à vouloir résoudre numériquement des équations différentielles, donc il doit déjà exister des implantations pour ça.

Il existe de nombreux intérêt à réutiliser une implantation existante :

- gain de temps (pas besoin de la reprogrammer) ;

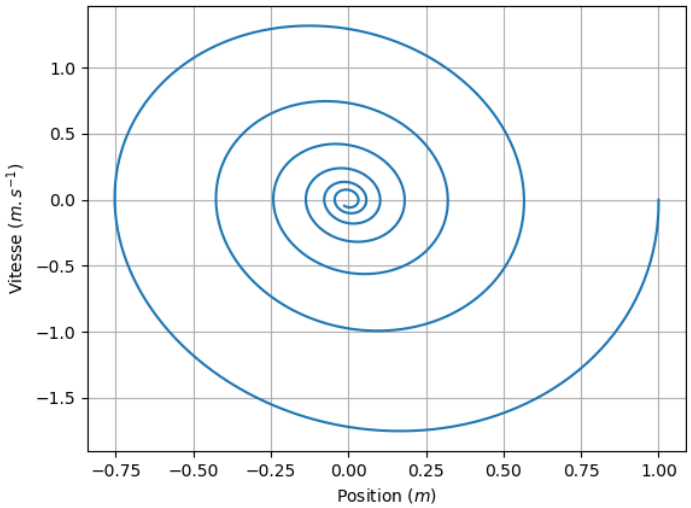

FIGURE 5 – Oscillateur amorti, portrait de phase par la méthode d'Euler.

- bugs connus (trouvés par d'autres) ;
- problèmes de performance connus (remarqués par d'autres).

De plus on peut espérer :

- que les bugs ont été corrigés ;
- que les performances ont été optimisées.

Dans le cas du logiciel libre :

- c'est souvent le cas ;
- sinon, on peut réparer le logiciel soi-même (ou le faire réparer).

La bibliothèque `scipy` propose un grand nombre de méthodes de calcul numérique. En particulier, `odeint` :

- est une fonction de la bibliothèque `scipy.integrate` ;
- résout numériquement des EDO ;
- utilise une méthode plus raffinée que celle d'Euler ;
- fonctionne sur une équation vectorielle avec le type `array` ;
- choisit elle-même le pas à utiliser.

On l'utilise comme suit.

```
from scipy.integrate import odeint
y_list = odeint(F, y0, t_list)
```

Reprenons l'exemple de l'oscillateur amorti (voir le résultat figure 6).

```
from scipy.integrate import odeint
from numpy import linspace

y0 = 1 # m
yp0 = 0 # m * s**-1
t0, t1 = 0, 20 # s
n = 1000
m = 250 # kg
c = 100 # kg * s**-1
k = 1000 # kg * s**-2

X0 = array([y0,yp0])

def F(X,t) :
    y,yp = X
    ypp = -(k*y + c*yp)/m
    return array([yp,ypp])
```

```
t_list = linspace(t0,t1,n)
X_list = odeint(F,X0,t_list)
y_list = [X[0] for X in X_list]

plt.clf()
plt.plot(t_list,y_list)
plt.xlabel('Temps ($s$)')
plt.ylabel('Position ($m$)')
plt.grid()
plt.savefig('oscillateur_amorti_odeint.png')
```

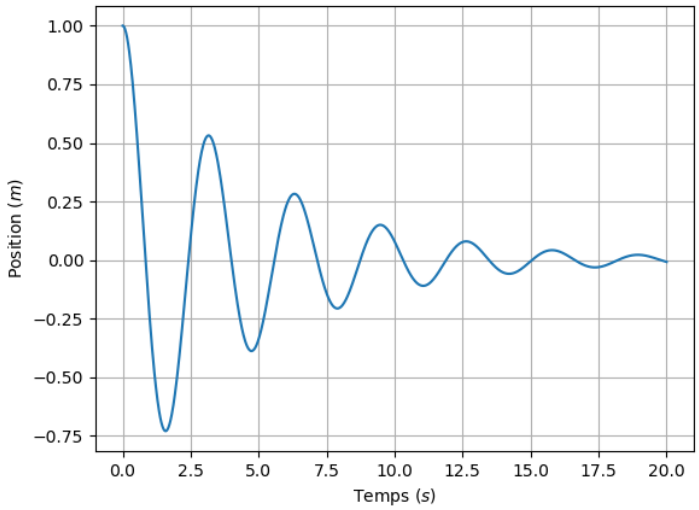

FIGURE 6 – Oscillateur amorti, trajectoire par `odeint`.

ATTENTION ! On ne donne pas les extrémités de l'intervalle mais les points où l'on veut des valeurs, `y0` (condition initiale) est la valeur en `t_list[0]`.

6 Influence du pas de discrétisation

On reprend l'exemple de l'amortisseur, avec $c = 100$ et avec différentes valeurs du pas de discrétisation h (voir figure 7).

La première courbe est clairement délirante sur le plan physique... Plusieurs facteurs contribuent à l'erreur :

- des **erreurs de méthode** (la méthode d'Euler n'est pas parfaite)
- des **erreurs de calcul** (les flottants ne sont pas les réels)

Rappelons que pour un pas h on note n le nombre de morceaux utilisés. Pour les erreurs de méthode, on dit qu'il y a **convergence** si :

1. la somme $e(h) = \sum_{k=0}^{n} |y(t_k) - y_k|$ des petites erreurs commises à chaque étape tend vers 0 quand h tend vers 0 ($e(h)$ est appelée **erreur de consistance relative à la solution** y).
2. et certaines conditions de **stabilité** de la méthode employée sont respectées (essentiellement, si l'on commet une petite erreur à chaque étape du calcul des y_k, menant à de nouvelles approximations $\tilde{y}_k$, alors si l'erreur de consistance relative aux y_k est petite, l'erreur entre les y_k et les $\tilde{y}_k$ doit être petite aussi).

On dit qu'une méthode est d'**ordre** p s'il existe une constante $K > 0$ telle que $e(h) \leq K h^p$, autrement dit l'erreur de consistance est un $O(h^p)$.

La méthode d'Euler est une méthode d'ordre 1.

7 Annexe : autres méthodes.

7.1 Méthode d'Euler implicite

La méthode d'Euler présentée jusqu'ici est dite **explicite** : y_{k+1} ne dépend que y_k et t_k. $y_{k+1} = y_k + hF(y_k, t_k)$

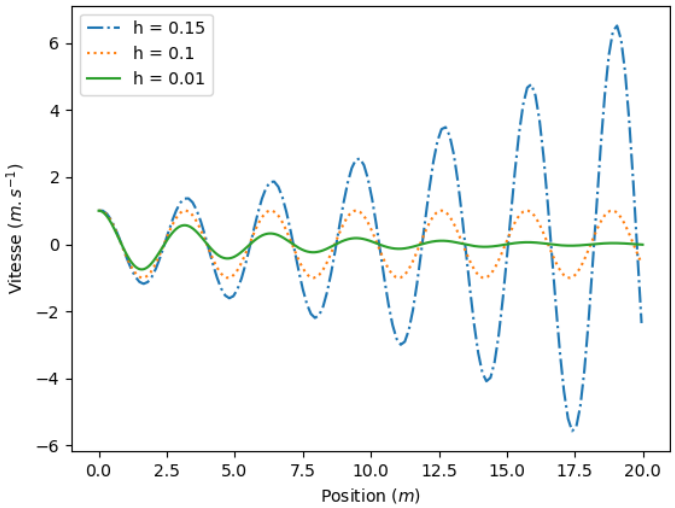

FIGURE 7 – Oscillateur amorti, trajectoires par la méthode d'Euler avec différents pas.

Mais en reprenant les approximations qui ont conduit à ce schéma, nous pouvons aussi bien écrire : $y_{k+1} = y_k + hF(y_{k+1}, t_k)$ ce qui mène à la méthode d'Euler **implicite**.

Elle a un gros inconvénient : pour trouver y_{k+1}, il faut résoudre une équation (numériquement).

Mais elle a un avantage : elle est souvent plus stable.

7.2 Méthode de Heun

Si $y_n = y(t_n)$, il existe $c \in]t_n, t_{n+1}[$ tel que $y(t_{n+1}) = y_n + h\,y'(c)$.

Tout le problème est d'estimer $y'(c)$.

On connaît l'idée de la méthode d'Euler explicite : $y'(c) \approx y'(t_n)$. Donc on pose $\boxed{k_1 = F(y_n, t_n)}$.

Puis $\boxed{y_{n+1} = y_n + hk_1}$ (erreur locale en $O(h^2)$).

L'idée de la méthode de Heun est la suivante : $y'(c) \approx \frac{y'(t_n)+y'(t_{n+1})}{2}$. On pose $\boxed{k_1 = F(y_n, t_n)}$. Alors $k_1 = y'(t_n)$.

On estime $y'(t_{n+1})$ en utilisant $y'(t_{n+1}) = F(y(t_{n+1}), t_{n+1})$.

Pour cela, on estime $y(t_{n+1})$ par la méthode d'Euler explicite.

On pose donc : $\boxed{k_2 = F(y_n + hk_1, t_n + h)}$ (erreur en $O(h^2)$). Puis $\boxed{y_{n+1} = y_n + h\left(\frac{k_1+k_2}{2}\right)}$.

On peut alors montrer que l'erreur locale est en $O(h^3)$.

7.3 Méthode de Runge-Kutta

Pour évaluer $y'(c)$, on calcule

$$\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

où k_1, k_2, k_3 et k_4 sont des évaluations des pentes :

k_1 pente en y_n

k_2 pente évaluée en $t_n + \frac{h}{2}$ en utilisant k_1 pour estimer $y(t_n + \frac{h}{2})$

k_3 pente évaluée en $t_n + \frac{h}{2}$ en utilisant k_2 pour estimer $y(t_n + \frac{h}{2})$

k_4 pente évaluée en $t_n + h$ en utilisant k_3 pour estimer $y(t_n + h)$.

$$k_1 = F(y_n, t_n)$$
$$k_2 = F(y_n + \frac{h}{2}k_1, t_n + \frac{h}{2})$$
$$k_3 = F(y_n + \frac{h}{2}k_2, t_n + \frac{h}{2})$$
$$k_4 = F(y_n + hk_3, t_n + h)$$
$$y_{n+1} = y_n + h\left(\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\right)$$

On peut alors montrer que l'erreur locale est en $O(h^5)$.
Il s'agit d'une méthode d'ordre de convergence 4.
Elle est facile à programmer et donc très populaire.