Physique numérique en Python
Journée de formation

Lycée Camille Jullian
Mars 2018

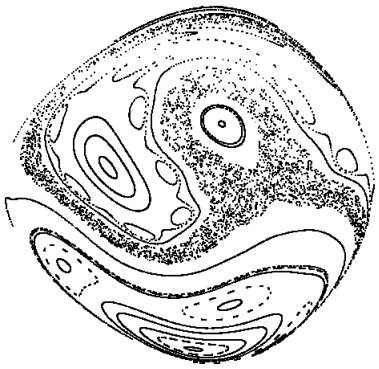

Table des matières

Introduction

L'objet de ce document est d'illustrer l'utilisation de Python dans le cadre de problèmes de physique. Les thèmes proposés s'appuient largement sur les contenus du programme de physique de TS. Partant de situations simples rencontrées en TS, on présente quelques pistes d'utilisation de résolution numérique avec Python. Certains exemples admettant une solution analytique, une confrontation avec les résultats numériques permet de mesurer la pertinence d'une approche informatique. D'autres exemples plus complexes, sans solution analytique simple pour des élèves de TS, voire sans solution analytique du tout, sont traités par approche numérique. Ces exemples peuvent être envisagés comme des prolongements du programme.

Une première série de trois activités propose de se familiariser avec le graphisme élémentaire de Python et avec les méthodes numériques de résolution numérique d'équations différentielles. Le problème de la chute sert de fil conducteur à ces quatre activités.

Une quatrième activité vise à mettre en application les compétences acquises avec des activités précédentes. Le fil conducteur est celui de l'oscillateur : oscillateur linéaire d'abord, oscillateur forcé ensuite, oscillateur non-linéaire enfin.

La cinquième activité illustre le problème du canon de Newton. À quelle condition une masse lancée depuis une certaine altitude, dans une certaine direction, peut-elle se satelliser ou échapper à l'attraction terrestre ? Cette activité est l'occasion d'illustrer la mécanique gravitationnelle dans la situation simple à deux corps en interaction. Elle requiert toute les compétences et la technicité des activités précédentes.

Un très grand merci à mes collègues Christophe Casseau et Marc Eldin pour leur aide à la construction et à la relecture de ce document.
Un grand merci également à David Boyer, IA-IPR de sciences-physiques, à l'initiative de ces journées, pour nous avoir réuni et donné l'occasion de monter cette première formation qui, nous l'espérons, sera suivie d'autres rencontres autour de la physique et de l'informatique. À suivre…

Laurent Sartre
laurent.sartre@ac-bordeaux.fr

Activité 1

Loi horaire

1.1 Position du problème

L'objectif de cette activité est de se familiariser avec quelques méthodes graphiques Python en vue de tracer une courbe représentative de la loi horaire d'un phénomène physique.
Pour illustrer notre propos, on s'intéresse à la chute libre d'un corps lâché sans vitesse initiale et pour lequel l'influence des frottements peut être négligée. Cette situation, à seul degré de liberté, peut être modélisée par une équation différentielle. Sa solution analytique peut être aisément trouvée en TS. Python est utilisé pour tracer la loi horaire de l'altitude du corps.

1.2 Mise en équation

Un point matériel M de masse m est soumis à la seule force de pesanteur. On note g la norme de l'accélération de la pesanteur. Le mouvement de M est étudié dans un référentiel local supposé galiléen. La verticale du lieu définit l'axe (Oz), vertical ascendant.
Le point M est initialement lâché d'une hauteur $h > 0$, sans vitesse initiale. Le schéma ci-dessous illustre la situation.
À un instant $t \geqslant 0$, on note $z(t)$ la cote du point M. Le principe fondamental de la dynamique permet d'établir la loi d'évolution de z sous la forme d'une équation différentielle.

$$\forall t \in \mathbb{R}^+ \qquad \ddot{z}(t) = -g \tag{1.1}$$

La solution de (1.1) est :

$$\forall t \in \mathbb{R}^+ \qquad z(t) = h - \frac{1}{2} g t^2 \tag{1.2}$$

1.3 Méthodologie de tracé

Pour tracer la courbe représentative de la loi horaire de z avec Python, on doit :

- définir les paramètres physiques g et h en précisant leurs valeurs numériques ;
- définir la fonction $z : t \mapsto h - g t^2/2$;
- préciser l'intervalle de temps $[t_{\min}, t_{\max}]$ sur lequel on désire tracer la courbe ;
- tracer la courbe et l'afficher.

1. comme les fonctions trigonométriques, la racine carrée, etc.

1.4 Modules et packages

Python est un langage de programmation dont les fonctionnalités peuvent facilement étendues à l'aide de nouvelles instructions. Par défaut, il met à la disposition de l'utilisateur un jeu significativement important d'instructions mais pas toujours suffisant pour des besoins plus spécifiques. Tracer des courbes avec Python entre dans la catégorie des besoins particuliers.
Pour étendre les fonctionnalités du langage, de nouvelles commandes peuvent être créées et conservées dans des fichiers particuliers appelés *modules*. L'intérêt de ces modules est de pouvoir être réutilisés en fonction des besoins. Parfois, des ensembles de modules sont regroupés pour former des *packages*. Il s'agit simplement d'une hiérarchisation des modules.
En pratique, dès qu'un module connu est nécessaire, il doit être *importé* au début du script grâce à l'instruction `import`.
Dans ce qui suit, le *module* `numpy`, dédié au calcul numérique, est importé. Il permet en particulier la manipulation de tableaux multidimensionnels pour stocker des données. Ce module fournit également un grand nombre de fonctions dédiées aux calculs mathématiques [1] et numériques.

```
# importation de numpy
# alias np
import numpy as np
```

Ainsi, dans le script suivant, le module `pyplot` du package `matplotlib` est importé. Ce module enrichit Python d'un certain nombre de fonctionnalités graphiques dont certaines d'entre elles sont présentées dans la suite de l'activité.

```
# importation de matplotlib.pyplot
# alias plt
import matplotlib.pyplot as plt
```

L'ajout de l'instruction `as` suivie d'un nom particulier autorise la création d'un *alias* qui remplace avantageusement le nom du module. Son intérêt réside dans le fait que les instructions des modules sont accessibles par une *notation pointée* : chaque instruction d'un module donné ne peut être appelée qu'en tapant son nom précédé du nom du module suivi d'un point. Par exemple, pour tracer la courbe d'un tableau de valeurs `y` en fonction d'un tableau de valeurs `x`, on code comme suit.

```
plt.plot(x,y)
```

L'affichage de la courbe se fait en codant comme suit.

```
plt.show()
```

Il convient de savoir qu'il est possible d'éviter la définition et l'utilisation d'alias en important globalement toutes les instructions d'un module. Il suffit de réaliser l'import de la manière suivante.

```
from matplotlib.pyplot import *
from numpy import *
```

Cette importation est à utiliser avec prudence. Certains modules définissent de nouvelles commandes qui portent le même nom que d'autres commandes qui préexistent dans Python. Dès lors, la dernière définition écrase toute autre définition préalable. Ce qui peut présenter des inconvénients. Dans la suite, nous utilisons systématiquement la notation pointée associée à la définition d'alias.

1.5 Codage

Pour tracer la courbe représentative de la loi horaire de z, les modules nécessaires sont d'abord chargés. Le script suivant permet le tracé de la loi horaire avec $h = 5\,\text{m}$ sur un intervalle de temps $[0, 1\,\text{s}]$ [2]. La courbe obtenue est celle de la figure 1.1.

```
# importation de numpy
# alias np
import numpy as np
# importation de matplotlib.pyplot
# alias plt
import matplotlib.pyplot as plt
```

On définit ensuite une *fonction* qui rend compte de la loi horaire. Dans le script suivant, cette fonction est notée `z` et trois *arguments* doivent lui être transmis pour qu'elle renvoie un résultat. Cela permettra de faire tracer des trajectoires en faisant varier les paramètres physiques h et g [3].

```
# fonction associée à la loi horaire
def z(t,g,h):
    return h - g * t**2 / 2
```

Vient à présent la définition des paramètres physiques, dont les valeurs sont *affectées* à des *variables* informatiques. Le nom de ces variables est arbitraire mais il ne peut pas débuter par un chiffre. En pratique, on attribue des noms pertinents au regard du rôle que les variables son censées jouer.

```
# paramètres physiques
g = 9.81 # accélération de la pesanteur
h = 1.0 # hauteur initiale en m
```

Afin de tracer la loi horaire, on doit se fixer un intervalle de temps. C'est généralement le sens physique qui prévaut pour déterminer cet intervalle. Dans le script qui suit, on choisit des instants compris entre 0 s et 5 s.

```
# tracé des trajectoires
t_min = 0.0 # instant initial en s
t_max = 1.0 # instant final en s
n_t = 100 # nombre de points de calculs
```

Pour tracer la loi horaire, on choisit un nombre de points `n_t` de calculs dans l'intervalle de temps précédent. Il ne reste plus qu'à faire calculer par Python les différents instants équirépartis dans cet intervalle et les valeurs de z associées. L'instruction `linspace` construit un *tableau* de `n_t` instants allant de `t_min` à `t_max`. Ce tableau est affecté à la variable `tab_t`.

```
# tableau des instants
tab_t = np.linspace(t_min,t_max,n_t)
```

L'une des forces du module `numpy` est de permettre la construction d'un tableau en appliquant une fonction sur un autre tableau. Le script suivant affecte à la variable `tab_z` le tableau obtenu en appliquant la fonction `z` au tableau `tab_t` et aux variables `g` et `h`. Cette facilité est l'un des points forts de Python pour le calcul scientifique.

```
# tableau des z
tab_z = z(tab_t,g,h)
```

Il reste à tracer la courbe en portant, dans un système d'axes orthogonaux, les valeurs présentes dans la tableau `tab_z` en fonction des valeurs présentes dans le tableau `tab_t`. Noter que le nombre de valeurs dans chaque tableau doit être le même pour que cette opération ait du sens.

```
# tracé de tab_z en fonction de tab_t
# option 'r-' : tracé en rouge (r)
# en reliant les points (-)
plt.plot(tab_t,tab_z,'r-')
# affichage de la courbe
plt.show()
```

Le résultat de l'exécution de la séquence des instructions précédentes est le graphique suivant.

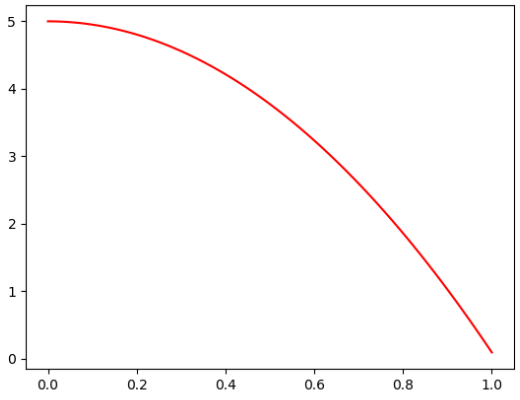

FIGURE 1.1 – Tracé d'une chute libre.

1.6 Compléments graphiques

Sur le graphique précédent, aucune information ne permet de connaître la nature des informations portées sur les axes. Il n'y a également pas de titre, ni d'information sur les valeurs des paramètres h et t_{max}. Ces information peuvent être ajoutées en faisant précéder l'instruction `plt.plot` d'un certain nombre d'instructions dédiées à l'affiche de commentaires.

2. Pourquoi ces choix de valeurs?
3. Il est possible de travailler avec des variables informatiques globales, mais là encore, pour des raisons de prudence, il convient re réduire leur utilisation au maximum.

```
plt.xlabel('Temps (en seconde)')
plt.ylabel('Position de $M$ (en mètre)')
plt.title("Chute libre sans frottement et
↪  sans vitesse initiale")

plt.text(0.7,4.8,r'$h = 5$ mètres')
plt.text(0.7,4.5,r'$t_{max} = 1$ seconde')
```

Le résultat graphique est alors le suivant.

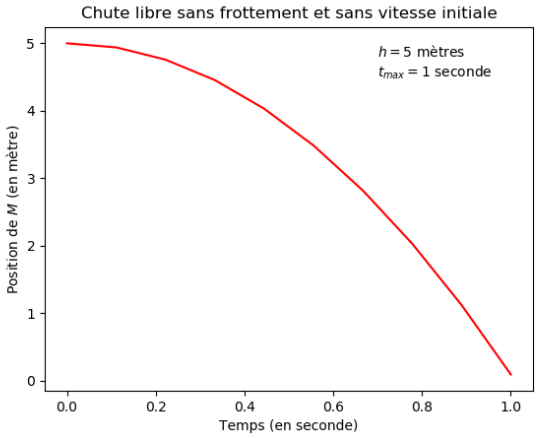

FIGURE 1.2

De nombreuses autres options sont disponibles avec le `pyplot` du package `matplotlib`. Le lecteur intéressé peut se reporter à la documentation et aux nombreux exemples disponibles sur le site de `matplotlib` (https://matplotlib.org/).

1.7 À vous de jouer

▶ **Question 1.** Modifier le script précédent pour afficher les deux courbes associées aux valeurs $h = 5\,\text{m}$ et $h = 10\,\text{m}$.

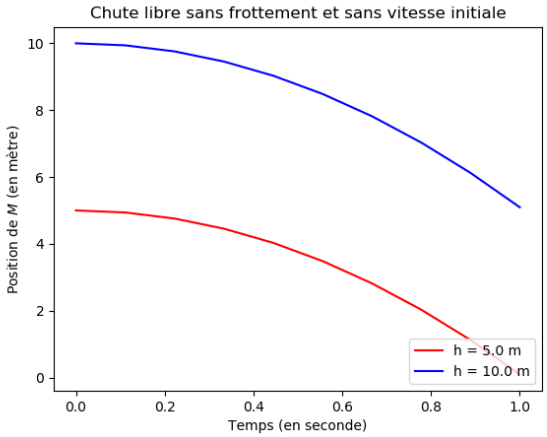

FIGURE 1.3

Pour afficher chaque valeur de h et le code couleur de la courbe associée, ajouter l'option `label` dans l'instruction `plot`.

```
plt.plot(tab_t,tab_z,'r-',label="h = " +
↪  str(h) + " m")
```

Cette option reçoit trois chaînes *chaîne de caratères* mises bout à bout[4] : la chaîne `"h = "`, la chaîne `str(h)` où l'instruction `str` convertit la valeur contenue dans la variable `h` en une chaîne de caractères, la chaîne `" m"`. Noter la présence des espaces.
L'instruction `plt.legend(loc=4)` saisie juste avant la commande d'affichage `plt.show()` place les commentaires précédents dans le coin inférieur droit du graphique.

▶ **Question 2.** Modifier le nombre de points du tracé pour afficher seulement quelques points de la courbe, comme sur la figure 1.4.

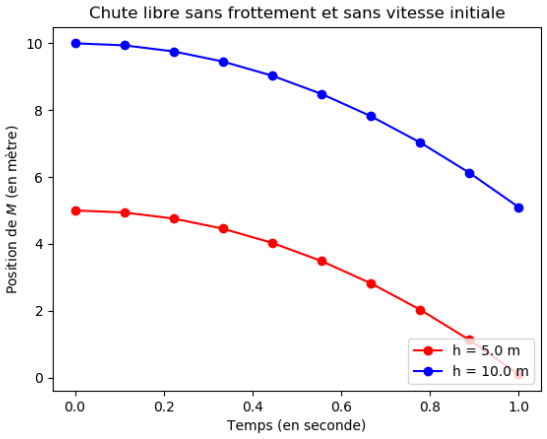

FIGURE 1.4

▶ **Question 3.** Comment obtenir la figure 1.5 ?

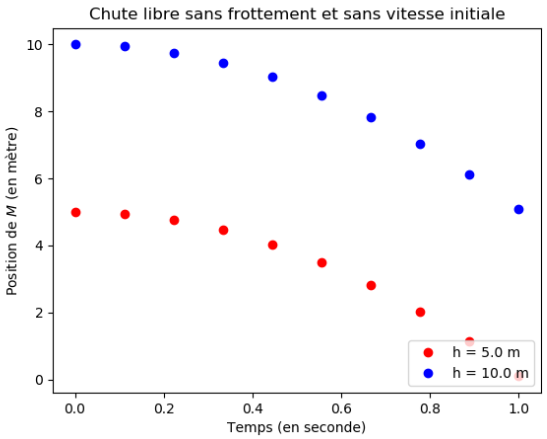

FIGURE 1.5

▶ **Question 4.** Sur un même graphique, tracer l'évolution temporelle de l'énergie cinétique par unité de masse et de l'énergie potentielle par unité de masse. Tracer également la somme de ces deux quantités.

4. En informatique, la mise bout à bout de symboles est appelée la *concaténation*.

Activité 2

Frottements fluides

2.1 Position du problème

L'objectif de cette activité est de se familiariser avec des méthodes numériques de résolution d'équations différentielles. La méthode d'Euler est introduite de trois façons pour en montrer les avantages et les inconvénients. Puis une méthode interne à Python, qui s'appuie sur des méthodes numériques classiques, est présentée comme un prolongement à même de répondre aux besoins numériques du physicien.

L'étude de la chute libre est encore le fil conducteur de cette activité. Des frottements fluides sont ajoutés à la situation précédente. *A priori*, cette situation, à un degré de liberté encore, peut être modélisée par une équation différentielle du second ordre dont l'inconnue est l'altitude du corps. Bien qu'une solution analytique existe, sa détermination ne peut cependant pas être trouvée par des élèves de TS. En introduisant la vitesse du corps, on se ramène à une équation différentielle du premier ordre. Des méthodes numériques fournissent des solutions approchées à cette équation. L'enseignant peut alors fournir l'expression de la solution analytique afin d'observer l'efficacité des méthodes numériques. Une discussion peut s'engager sur la confiance à accorder aux résultats numériques en vue de sensibiliser les élèves aux problèmes numériques d'une part, de montrer la nécessité d'une attitude critique à l'égard des résultats fournis par l'outil numérique d'autre part. Fort de ces compétences, le travail peut être poursuivi pour étudier l'évolution de l'altitude du corps et pour faire un bilan énergétique numérique.

2.2 Étude de la vitesse

La force de frottements fluides est modélisée par le vecteur force $\vec{F}_f = -\lambda \dot{z} \hat{e}_z$, λ étant un coefficient strictement positif lié aux frottements, $\hat{e}_z$ désignant un vecteur unitaire vertical ascendant. L'équation donnant l'évolution de z est à présent :

$$\forall t \in \mathbb{R}^+ \qquad \ddot{z}(t) = -g - \frac{\lambda}{m}\dot{z}(t) \tag{2.1}$$

En raison du choix d'orientation de l'axe vertical, $\dot{z}$ est négative : la vitesse du corps augmente en descendant. Il peut être commode de poser $v = -\dot{z}$ de manière à raisonner sur une vitesse positive dont la valeur croît au fur et à mesure que le corps perd de l'altitude. Par ailleurs, une analyse dimensionnelle de l'équation précédente montre que le rapport m/λ est homogène à un temps caractéristique τ.

De fait, l'équation différentielle précédente peut se mettre sous la forme suivante.

$$\forall t \in \mathbb{R}^+ \qquad \dot{v}(t) = g - \frac{1}{\tau}v(t) \tag{2.2}$$

On s'intéresse à présent à la recherche de solutions numériques approchées à cette équation, $v(0) = 0$ (corps lâché sans vitesse initiale). Pour mettre en œuvre les *schémas numériques d'Euler*, la relation précédente est mise sous forme intégrée.

$$\forall t \in \mathbb{R}^+ \qquad \int_0^t \dot{v}(u)\,\mathrm{d}u = \int_0^t \left(g - \frac{1}{\tau}v(u)\right)\mathrm{d}u$$

Le membre de gauche s'intègre immédiatement sous la forme :

$$\int_0^t \dot{v}(u)\,\mathrm{d}u = v(t) - v(0)$$

Le membre de droite ne s'intègre que partiellement. L'équation finalement obtenue se met sous la forme intégrale suivante.

$$\forall t \in \mathbb{R}^+ \qquad v(t) - v(0) = gt - \frac{1}{\tau}\int_0^t v(u)\,\mathrm{d}u \tag{2.3}$$

La condition initiale $v(0) = 0$ permet de simplifier le membre de gauche. Mais pour des raisons pédagogiques, il peut être intéressant de conserver la présence de ce terme en vue de préparer la mise en œuvre des techniques numériques.

2.3 Schémas d'Euler

L'idée des schémas numériques d'Euler est de remplacer l'intégrale précédente par une valeur approchée. Cependant, une telle approximation ne peut être faite sans précaution. Elle sera d'autant plus raisonnable que le terme sous le signe intégral varie peu sur l'intervalle d'intégration. En pratique, il y a peu de chance que cette propriété soit vérifiée. La vitesse peut varier de manière importante sur l'intervalle d'observation du phénomène. En revanche, si cet intervalle est divisé en de nombreux sous-intervalles suffisamment petits pour que la vitesse y varie peu, l'approximation devient plus raisonnable.

Notons t_{max} l'instant final d'observation de la chute. Découpons la durée $\Delta t = t_{max} - 0$ en n_t intervalles de mêmes durées $\delta t = \Delta t / n_t$. Définissons les instants :

$$\forall i \in \{0, 1, \ldots, n\}, \quad t_i = i \times \delta t$$

Plus n_t est grand, plus δt est petit et plus il y a de valeurs d'instants t_i. En augmentant la valeur de n_t, on peut raisonnablement penser que les intervalles temporels deviennent suffisamment petits pour que la vitesse y varie peu. L'idée des *schémas d'Euler* est d'exploiter cette analyse pour donner une forme pratique à l'équation (2.3) en intégrant sur un intervalle de la forme $[t_i, t_{i+1}]$, $i = \{0, 1, \ldots, n-1\}$. Ainsi, il est possible d'écrire, pour tout entier i prenant ses valeurs dans l'ensemble $\{0, 1, \ldots, n-1\}$:

$$v(t_{i+1}) - v(t_i) = g \times \delta t - \frac{1}{\tau}\int_{t_i}^{t_{i+1}} v(u)\,\mathrm{d}u$$

Si sur chaque intervalle $[t_i, t_{i+1}]$, la vitesse varie peu, elle varie quand même. La question est alors de savoir quelle valeur choisir pour l'approcher.

- Le *schéma d'Euler explicite* propose de remplacer $v(u)$ par $v(t_i)$ sur cet intervalle. L'équation intégrale prend alors la forme approchée suivante.

$$v(t_{i+1}) - v(t_i) \approx \left(g - \frac{v(t_i)}{\tau}\right) \times \delta t$$

- Le *schéma d'Euler implicite* propose de remplacer $v(u)$ par $v(t_{i+1})$ sur cet intervalle. L'équation intégrale prend alors la forme approchée suivante.

$$v(t_{i+1}) - v(t_i) \approx \left(g - \frac{v(t_{i+1})}{\tau}\right) \times \delta t$$

Ces relations permettent de calculer, de proche en proche, les *valeurs approchées* de la vitesse aux différents instants t_i. Ces valeurs ne sont pas les valeurs exactes et seront notées $v_i \approx v(t_i)$ de sorte que les relations établies ci-dessus se traduisent par les égalités suivantes.

$$\begin{cases} v_{i+1} - v_i = \left(g - \frac{v_i}{\tau}\right) \times \delta t & \text{(Euler explicite)} \\ v_{i+1} - v_i = \left(g - \frac{v_{i+1}}{\tau}\right) \times \delta t & \text{(Euler implicite)} \end{cases}$$

En isolant v_{i+1}, on obtient des relations de récurrence qui permettent le calcul de proche en proche des valeurs approchées.

$$\begin{cases} v_{i+1} = \left(1 - \frac{\delta t}{\tau}\right) v_i + g \times \delta t & \text{(Euler explicite)} \\ v_{i+1} = \frac{1}{1+\frac{\delta t}{\tau}} \left(v_i + g \times \delta t\right) & \text{(Euler implicite)} \end{cases}$$

2.4 Méthodologie de tracé

Pour tracer la courbe représentative de la loi horaire de v avec Python, on doit :

- définir les paramètres physiques g et h en précisant leurs valeurs numériques ;
- préciser l'intervalle de temps $[t_{\min}, t_{\max}]$ sur lequel on désire tracer la courbe ;
- discrétiser cet intervalle de temps ;
- évaluer de proche en proche les valeurs approchées de vitesses ;
- tracer la courbe et l'afficher.

2.5 Codage

Le début du script comporte les mêmes importations que celles rencontrées dans l'activité 1.

```
# importation de numpy
# alias np
import numpy as np
# importation de matplotlib.pyplot
# alias plt
import matplotlib.pyplot as plt
```

À présent, il nous faut construire l'équivalent de la loi horaire définie par une fonction dans l'activité 1. Une *fonction* Python remplit ce rôle. Elle reçoit un ensemble d'arguments nécessaires au calcul des différentes valeurs approchées de la vitesse et retourne un tableau de ces valeurs. D'un point de vue informatique, le calcul de proche en proche associé aux relations de récurrence fait appel à une *structure répétitive* (ou *boucle*). Dans le script suivant illustrant la construction du tableau à l'aide du schéma d'Euler explicite, la boucle `for` parcourt une liste d'entiers de `0` à `n_t-1`. C'est la fonction `range(n_t)` qui définit cette liste. Noter que la dernière valeur définie par cette fonction est `n_t-1` et non `n_t`. D'autres utilisations de la fonction `range` reçoivent plusieurs arguments. Si `a` et `b` sont deux nombres entiers :

- `range(a,b)` définit un objet itérable `[a,a+1,a+2,...,b-1]` ;
- `range(b)` définit un objet itérable `[0,1,2,...,b-1]` ;
- `range(a,b,c)` définit un objet itérable `[a,a+c,a+2c,...]` ;

La création d'un tableau se fait à l'aide de l'instruction `np.array`, du module `numpy`. En tapant

```
tab_v = np.array(n_t)
```

on crée un tableau vide à `n_t+1` éléments. Les éléments d'un tableau sont accessibles par l'intermédiaire de leur rang dans le tableau, en commençant par le rang 0. Ainsi, à chaque rang dans le tableau `tab_v` est associée une valeur approchée de vitesse selon les correspondances suivantes.

$$\begin{aligned} \texttt{tab\_v[0]} &\longleftrightarrow v_0 \\ \texttt{tab\_v[1]} &\longleftrightarrow v_1 \\ &\cdots \\ \texttt{tab\_v[n\_t]} &\longleftrightarrow v_{n_t} \end{aligned}$$

Le script suivant présente la fonction `euler_explicite` qui calcule et retourne le tableau des vitesses.

```
def eulerExp(t_min,t_max,n_t,g,h,v0,tau):
  # création d'un tableau vide
  # des vitesses
  # à (n_t + 1) éléments
  tab_v = np.zeros(n_t)
  # affectation de la vitesse initiale
  # au premier élement du tableau
  tab_v[0] = v0
  # calcul de delta_t
  delta_t = (t_max - t_min) / n_t
  # boucle de calcul
  # des vitesses approchées
  for i in range(n_t-1):
    tab_v[i+1] = (1 - delta_t / tau) *
    ↪  tab_v[i] + g * delta_t
  return tab_v
```

D'autres choix de codage sont possibles. Le choix fait ici est celui de la simplicité et d'un lien étroit avec les expressions mathématiques établies plus haut.
Le script suivant définit les paramètres physiques du problème.

```
# paramètres physiques
g = 9.81 # accélération de la pesanteur
m = 1.0 # masse
lamb = 0.1 # coef. de frottement
tau = m / lamb
v0 = 0.0 # vitesse initiale
h = 5.0 # hauteur initiale
```

Puis les tableaux des instants et des vitesses sont calculés.

```
t_min = 0.0 # instant initial
t_max = 5.0 * tau # instant final
n_t = 100

tab_t = np.linspace(t_min,t_max,n_t)
tab_v = eulerExp(t_min,t_max,n_t,g,h,v0,tau)
```

Enfin, la courbe et quelques informations sont affichées.

```
plt.plot(tab_t,tab_v,'r-',label="h = " +
 ↪  str(h) + " m")
plt.xlabel('Temps (en s)')
plt.ylabel('Vitesse de $M$ (en m/s)')
plt.title("Chute libre avec frottements
 ↪  fluides et sans vitesse initiale")
plt.text(0.7 * t_max,10.0,r'$h = ' + str(h) +
 ↪  '$ m')
plt.text(0.7 * t_max,5.0,r'$t_{max} = ' +
 ↪  str(t_max) + '$ s')
plt.text(0.7 * t_max,15.0,r'$m = ' + str(m) +
 ↪  '$ kg')
plt.text(0.7 * t_max,20.0,r'$\lambda = ' +
 ↪  str(lamb) + '$ kg/s')# h = 10.0 # hauteur
 ↪  initiale
plt.show()
```

Le résultat graphique est présenté figure 2.1.

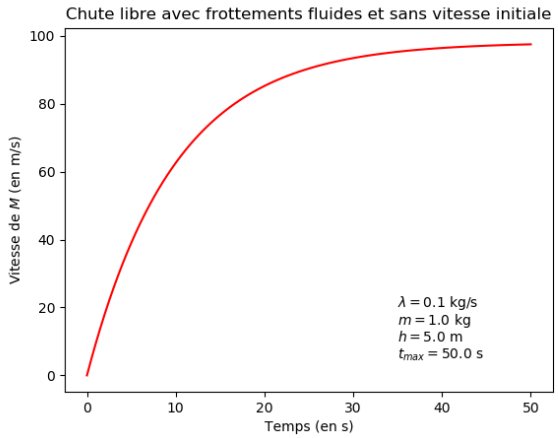

FIGURE 2.1

La mise en œuvre du schéma d'Euler implicite suit exactement la même procédure de codage. Ce travail est proposé en fin de cette activité.

2.6 Solution exacte

L'équation (2.2) admet une solution analytique de la forme :

$$v(t) = g\tau\left(1 - e^{-t/\tau}\right)$$

Son évolution temporelle peut être tracée en Python en adoptant la même méthodologie que dans l'activité 1. Le script suivant présente une solution dans laquelle la fonction v est à compléter.

```
import numpy as np
import matplotlib.pyplot as plt

# fonction associée à la loi horaire
# de la vitesse
def v(tab_t,g,h,tau):
  return g * tau * (1.0 - np.exp(-tab_t /
   ↪  tau))

# paramètres physiques
g = 9.81 # accélération de la pesanteur
m = 1.0 # masse
lamb = 0.1 # coef. de frottement
tau = m / lamb
v0 = 0.0 # vitesse initiale
h = 5.0 # hauteur initiale

# tracé des trajectoires
t_min = 0.0 # instant initial
t_max = 5.0 * tau # instant final
n_t = 100
# tableaux
tab_t = np.linspace(t_min,t_max,n_t)
tab_v = v(tab_t,g,h,tau)
# tracé
plt.plot(tab_t,tab_v,'g--',label="Solution
 ↪  exacte")
plt.xlabel('Temps (en s)')
plt.ylabel('Vitesse de $M$ (en m/s)')
plt.title("Chute libre avec frottements
 ↪  fluides et sans vitesse initiale")
plt.text(0.7 * t_max,40.0,r'$h = ' + str(h) +
 ↪  '$ m')
plt.text(0.7 * t_max,35.0,r'$t_{max} = ' +
 ↪  str(t_max) + '$ s')
plt.text(0.7 * t_max,45.0,r'$m = ' + str(m) +
 ↪  '$ kg')
plt.text(0.7 * t_max,50.0,r'$\lambda = ' +
 ↪  str(lamb) + '$ kg/s')
plt.legend(loc=4)
plt.show()
```

Le résultat graphique est donné figure 2.2.

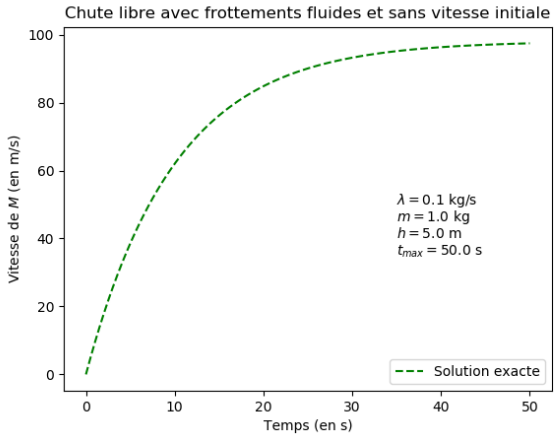

FIGURE 2.2

2.7 Solution Python

D'autres méthodes numériques, plus efficaces que les schémas d'Euler, permettent d'approcher la solution d'une

équation différentielle. C'est le cas de la commande `odeint` du module `integrate` du package `scipy`[1]. `odeint` met en œuvre un mélange de méthodes numériques parmi lesquelles la méthode de Runge Kutta d'ordre 4 et des méthodes à pas adaptatifs pour résoudre des équations différentielles du premier ordre. Dans la suite, nous n'utiliserons que cette commande du module `scipy.integrate`. C'est pourquoi nous pouvons charger directement l'instruction pour éviter le recours à la notation pointée.

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

Cette commande reçoit au minimum trois arguments. Le premier argument fait référence à l'équation différentielle[2]. Le deuxième paramètre est un tableau de la ou des conditions initiales. Le troisième argument est le tableau des instants où la solution approchée est déterminée. Le résultat de l'appel à la fonction `odeint` est un tableau dont la dimension dépend du nombre d'équations différentielles passées en argument. Un quatrième argument peut être ajouté qui indique à `odeint` les valeurs de paramètres présents dans les équations différentielles.
Illustrons notre propos avec l'équation 2.2 rappelée ci-dessous.

$$\forall t \in \mathbb{R}^+ \qquad \dot{v}(t) = g - \frac{1}{\tau} v(t)$$

Le second membre de cette équation permet de définir une fonction qui est le premier argument de `odeint`. En Python, cette fonction est définie avec au minimum deux arguments : une variable[3] associée à la grandeur recherchée et le tableau des instants `tab_t`. Les éventuels autres arguments sont les paramètres physiques de l'équation.

```
def eqDif(v,tab_t,g,tau):
    return g - v / tau
```

La résolution numérique de l'équation différentielle procède de la même logique que celle présentée jusqu'ici avec les autres méthodes numériques.

```
# paramètres physiques
g = 9.81 # accélération de la pesanteur
m = 1.0 # masse
lamb = 0.5 # coefficient de frottement
tau = m / lamb # temps caractéristique
h = 5.0 # hauteur initiale
# vitesse initiale
v0 = 0.0
```

Le script suivant précise les différentes étapes menant à la résolution. Le résultat de l'appel à `odeint` est un tableau qui contient autant de valeurs que le tableau `tab_t`. Ce tableau est affecté à la variable `tab_v`.

```
# résolution numérique
t_min = 0.0
t_max = 5 * tau
n_t = 100
tab_t = np.linspace(t_min,t_max,n_t)
tab_v = odeint(eqDif,v0,tab_t, args=(g,h))
```

La fin du script permet l'affichage de la courbe solution et de quelques commentaires. Noter l'utilisation de la fonction `max` pour trouver la valeur maximale présente dans le tableau `tab_v`. Cette valeur est utilisée pour placer les commentaires.

```
v_max = max(tab_v)

plt.plot(tab_t,tab_v,label="Méthode odeint")
plt.xlabel('Temps (en s)')
plt.ylabel('Vitesse de $M$ (en m/s)')
plt.title("Évolution de la vitesse")
plt.text(0.7 * t_max,0.8 * v_max,r'$h = ' +
 ↪  str(h) + '$ m')
plt.text(0.7 * t_max,0.7 * v_max,r'$t_{max} =
 ↪  ' + str(t_max) + '$ s')
plt.text(0.7 * t_max,0.6 * v_max,r'$m = ' +
 ↪  str(m) + '$ kg')
plt.text(0.7 * t_max,0.5 * v_max,r'$\lambda =
 ↪  ' + str(lamb) + '$ kg/s')
plt.legend(loc=4)
plt.show()
```

Le résultat graphique est donné figure 2.3

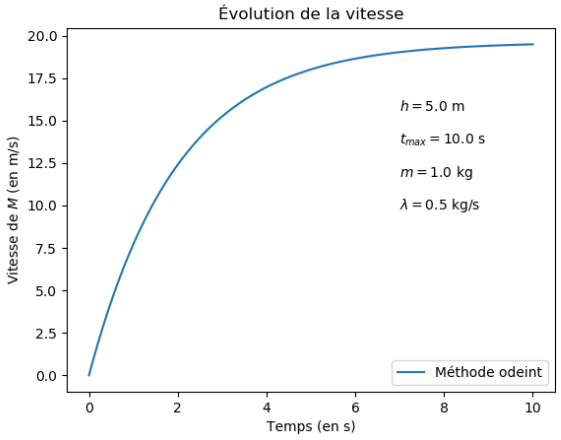

FIGURE 2.3

La méthode `odeint` fournit des résultats numériques tout à fait pertinents. Elle est plus précise que les schémas d'Euler dont l'objet était de présenter une technique numérique simple, à la base de nombreuses autres méthodes.

2.8 À vous de jouer

▶ **Question 1.** Mettre en œuvre le schéma d'Euler implicite pour tracer l'évolution temporelle de la vitesse.
Sur un même graphique, tracer les courbes obtenues par

1. Le package `scipy`
2. Celle-ci peut d'ailleurs être vectorielle, ce qui permet la résolution de certaines équations différentielles d'ordre supérieur à 1. C'est souvent le cas en physique.
3. Nous verrons plus loin qu'il peut s'agir d'un ensemble de variables, sous la forme d'un tableau.

les deux schémas d'Euler et la courbe théorique. Observer les différences en prenant un nombre de points de calculs peu élevé. La figure 2.4 est une illustration du résultat attendu.

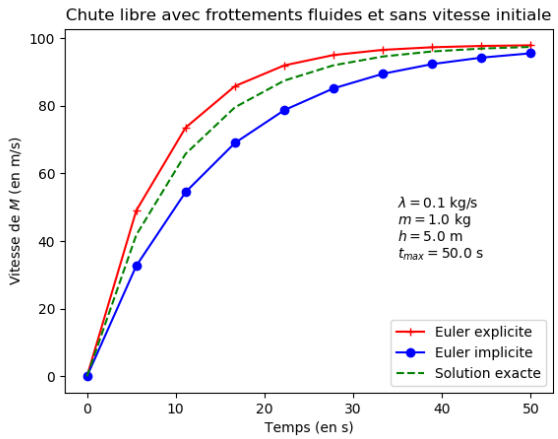

FIGURE 2.4

▶ **Question 2.** La solution de l'équation 2.2 :

$$v(t) = g\tau\left(1 - e^{-t/\tau}\right)$$

permet de trouver l'évolution de $z(t)$. Attention à ne pas oublier que $v = -\dot{z}$ de sorte qu'après intégration :

$$\forall\, t \in \mathbb{R}^+, \quad z(t) = h + g\tau^2\left(1 - \frac{t}{\tau} - e^{-t/\tau}\right)$$

On peut également voir z comme la solution de l'équation différentielle suivante :

$$\forall\, t \in \mathbb{R}^+, \quad \dot{z}(t) = -gt - \frac{1}{\tau}\left(z(t) - h\right) \tag{2.4}$$

avec la condition initiale $z(0) = h$.

▷ **2.1.** Mettre en œuvre les schémas d'Euler, la méthode `odeint` pour trouver des solutions approchées de cette équation différentielle.

▷ **2.2.** Tracer les courbes associées aux trois solution numériques et à la solution exacte.

▷ **2.3.** Comparer les méthodes numériques pour de petites valeurs de `n_t`.

▷ **2.4.** Observer le comportement des solutions sur des intervalles de temps différents.

▶ **Question 3.** Les résultats précédents peuvent être exploités pour faire une étude énergétique numérique.

▷ **3.1.** Comment définir deux tableaux `tab_Ec` et `tab_Ep` contenant respectivement les valeurs numériques des énergies cinétiques et potentielles ?

▷ **3.2.** Tracer les évolutions temporelles des énergies cinétiques et temporelles ainsi que celle de l'énergie mécanique. Commenter.

Activité 3

Trajectoire

3.1 Position du problème

Les activités précédentes ont permis d'acquérir les méthodes élémentaires de tracé de courbes et de résolution numérique d'équations différentielles simples. Ces compétences peuvent être mises à profit pour étudier un mouvement plan et tracer la trajectoire d'un corps à partir de la solution approchée d'un système différentiel.
La trajectoire plane du point matériel peut aisément être numériquement tracée dans le cas de la chute libre avec vitesse initiale, sans frottement. La prise en compte de frottements fluides est abordé sous forme d'exercice pour compléter la prise en main de Python.

3.2 Mise en équation

Le point matériel M de masse m, soumis à la seule force de pesanteur, est initialement lancé avec une vitesse v_0 faisant un angle α avec l'axe horizontal (Ox), depuis le point de coordonnées $(0,h)$. Son vecteur vitesse initiale se décompose dans la base $(\hat{e}_x,\hat{e}_z)$ sous la forme :

$$\vec{v}(0) = v_0\cos\alpha\,\hat{e}_x + v_0\sin\alpha\,\hat{e}_z$$

À un instant $t \geqslant 0$, les coordonnées du point M, notées $x(t)$ et $(z(t)$, vérifient le système différentiel suivant.

$$\begin{cases}\ddot{x}(t) = 0\\ \ddot{z}(t) = -g\end{cases} \tag{3.1}$$

La solution de (3.1) est :

$$\forall\, t\in\mathbb{R}^+ \qquad \begin{cases}x(t) = v_0\,t\cos\alpha\\ z(t) = h + v_0\,t\sin\alpha - \frac{1}{2}g\,t^2\end{cases} \tag{3.2}$$

3.3 Méthodologie de tracé

Tracer la trajectoire revient à tracer $z(t)$ en fonction de $x(t)$ pour $t \geqslant 0$. En Python, cela revient d'abord à construire les tableaux de valeurs des coordonnées x et z pour t variant dans un intervalle fixé. La méthodologie est très proche de celle présentée dans l'activité. Il convient simplement d'ajouter la construction du tableau associé aux valeurs de x.

- Définir les paramètres physiques du problème en précisant leurs valeurs numériques.
- Définir deux fonctions $x\colon x\mapsto v_0\,t\cos\alpha$ et $z\colon t\mapsto h + v_0\,t\sin\alpha - g\,t^2/2$.
- Préciser l'intervalle de temps $[\,t_{\min},t_{\max}\,]$ sur lequel on désire tracer la courbe.
- Tracer la courbe et l'afficher.

3.4 Codage

Le début du script reprend en grande partie celui écrit dans l'activité 1. Il est complété par la définition de la fonction `x` et par la construction du tableau `tab_x`.

```
import numpy as np
import matplotlib.pyplot as plt
# fonctions associée aux lois horaires
def x(t,g,v0,alpha):
  return v0 * np.cos(alpha) * t
def z(t,g,h,v0,alpha):
    return h + v0 * np.sin(alpha) * t - g *
    ↪  t**2 / 2
# paramètres physiques
g = 9.81 # accélération de la pesanteur
h = 5.0 # altitude initiale
v0 = 10.0 # vitesse initiale
# angle en degrés
alpha_deg = 45.0
# Python calcule en radians
alpha = alpha_deg * np.pi / 180.0
# intervalle temporel d'étude
t_min = 0.0 # instant initial
t_max = 1.0 # instant final
n_t = 100 # nombre de points
# tableaux
tab_t = np.linspace(t_min,t_max,n_t)
tab_x = x(tab_t,g,v0,alpha)
tab_z = z(tab_t,g,h,v0,alpha)
```

Tracer la trajectoire se fait en utilisant la fonction `plot` avec comme arguments les deux tableaux `tab_x` et `tab_z`.

```
plt.plot(tab_x,tab_z,'r-')
```

La fin du script ajoute les commentaires.

```
plt.xlabel('Abscisse (en s)')
plt.ylabel('Cote (en m)')
plt.title("Chute libre avec vitesse
 ↪  initiale")
plt.text(0.7 * t_max,35.0,r'$t_{max} = ' +
 ↪  str(t_max) + '$ s')
plt.text(0.7 * t_max,40.0,r'$h = ' + str(h) +
 ↪  '$ m')
plt.text(0.7 * t_max,45.0,r'$v_{0} = ' +
 ↪  str(v0) + '$ m/s')
plt.text(0.7 * t_max,50.0,r'$\alpha = ' +
 ↪  str(alpha_deg) + '$°')
plt.show()
```

Le résultat graphique est donné figure 3.1

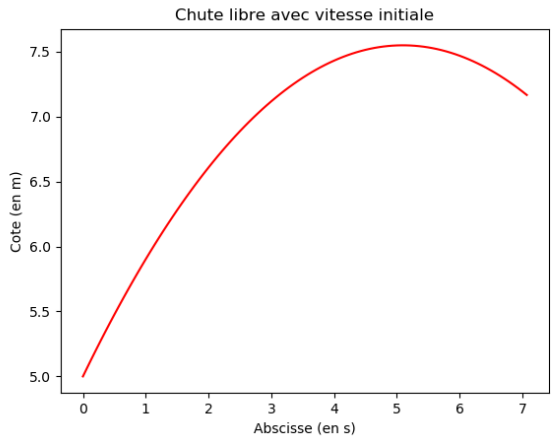

FIGURE 3.1

3.5 Système différentiel et Python

L'exemple de la chute libre est suffisamment simple pour permettre l'écriture d'une solution analytique du système différentiel. Mais de nombreuses situations physiques sont modélisées par plusieurs équations différentielles couplées qui peuvent souvent se mettre sous la forme suivante.

$$\forall t \in \mathbb{R}^+, \quad \begin{cases} \ddot{x}(t) = f_x(x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t), t) \\ \ddot{y}(t) = f_y(x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t), t) \\ \ddot{z}(t) = f_z(x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t), t) \end{cases}$$

En physique, un tel système est complété par les conditions initiales.

$$\begin{array}{lll} x(0) = x_0 & y(0) = y_0 & z(0) = z_0 \\ \dot{x}(0) = v_{0x} & \dot{y}(0) = v_{0y} & \dot{z}(0) = v_{0z} \end{array}$$

Dans l'exemple de la chute libre, deux équations rendent compte du phénomène physique. Les fonctions f_x et f_z sont :

$$\begin{cases} f_x(x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t), t) = 0 \\ f_z(x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t), t) = -g \end{cases}$$

Leurs expressions sont très simples. Nous rencontrerons dans une activité ultérieure une situation de couplage des coordonnées en les deux équations.
En Python, un tel système différentiel n'est pas résolu numériquement directement sous cette forme. Le système faisant intervenir des dérivées d'ordre supérieur à 1 doit d'abord être transformé en un système du premier ordre [1]. Cette transformation est simple à réaliser en introduisant de nouvelles fonctions [2] qui « absorbent » une partie des dérivées. Posons :

$$p = \dot{x} \qquad q = \dot{y} \qquad r = \dot{z}$$

Ainsi, le problème physique peut être formellement décrit par le système différentiel équivalent du premier ordre suivant :

$$\forall t \in \mathbb{R}^+, \quad \begin{cases} \dot{x}(t) = p(t) \\ \dot{p}(t) = f_x(x(t), y(t), z(t), p(t), q(t), r(t), t) \\ \dot{y}(t) = q(t) \\ \dot{q}(t) = f_y(x(t), y(t), z(t), p(t), q(t), r(t), t) \\ \dot{z}(t) = r(t) \\ \dot{r}(t) = f_z(x(t), y(t), z(t), p(t), q(t), r(t), t) \end{cases}$$

muni des conditions initiales :

$$\begin{array}{lll} x(0) = x_0 & y(0) = y_0 & z(0) = z_0 \\ p(0) = v_{0x} & q(0) = v_{0y} & r(0) = v_{0z} \end{array}$$

Sous cette forme, la commande `odeint` est à même de donner une solution numérique du problème. La principale difficulté est de définir le système différentiel en vue de son utilisation avec `odeint`. En pratique, ce système n'est rien d'autre qu'une relation vectorielle de la forme :

$$\dot{X}(t) = F(X(t), t) \qquad \text{avec} \qquad X(t) = \begin{pmatrix} x(t) \\ p(t) \\ y(t) \\ q(t) \\ z(t) \\ r(t) \end{pmatrix}$$

Pour utiliser la fonction `odeint`, il suffit donc de définir la fonction vectorielle F. Le système suivant définit cette fonction dans le cas de la chute libre sans frottement.

$$\forall t \in \mathbb{R}^+, \quad F(X(t), t) = \begin{pmatrix} p(t) \\ 0 \\ r(t) \\ -g \end{pmatrix} \qquad \text{avec} \qquad X(t) = \begin{pmatrix} x(t) \\ p(t) \\ z(t) \\ r(t) \end{pmatrix}$$

Le script suivant traduit cette formulation mathématique en code Python, en définissant la fonction `F`.

```
def F(X,t,g):
  x,p,z,r = X
  return [p,0,r,-g]
```

Dans ce script, la fonction renvoie un tableau à deux dimensions. Chaque colonne du tableau contient les valeurs numériques associées respectivement à x, p, z, r et comporte autant de valeurs que d'instants définis dans le tableau `tab_t`. Python offre un moyen simple de récupérer chacune des colonnes pour les affecter à des variables idoines. Chaque colonne porte un numéro qui commence à 0. Ainsi, à la colonne 0 est associé le tableau des valeurs de x, à la colonne 1 celui des valeurs de p, à la colonne 2 celui des valeurs de z et à la colonne 3 celui des valeurs de r. L'opération de sélection d'une colonne en Python est appelée *slicing*, littéralement *tranchage*. Le script suivant illustre cette opération de récupération des tableaux de valeurs de x et de z, après avoir utilisé la commande `odeint`.

1. Cette analyse, *a priori* mathématique, est à rapprocher des modalités d'étude des mouvements en mécanique analytique et de la notion d'espace de phases.
2. auxquelles il est souvent possible de donner un sens physique

```
# paramètres physiques
g = 9.81 # accélération de la pesanteur
x0 = 0.0
z0 = 5.0 # altitude initiale
v0 = 10.0 # vitesse initiale
# angle en degrés

alpha_deg = 45.0
# Python calcule en radians
alpha = alpha_deg * np.pi / 180.0
cond_init = [x0,v0 * np.cos(alpha),z0,v0 *
 ↪  np.sin(alpha)]
# intervalle temporel d'étude
t_min = 0.0 # instant initial
t_max = 1.0 # instant final
n_t = 3 # nombre de points
# tableaux
tab_t = np.linspace(t_min,t_max,n_t)
sol_num = odeint(F,cond_init,tab_t,args=(g,))
tab_x = sol_num[:,0] # tableaux des x
tab_z = sol_num[:,2] # tableaux des z
```

Le tracé avec les options est immédiat. La figure 3.2 illustre la mise en œuvre du script précédent.

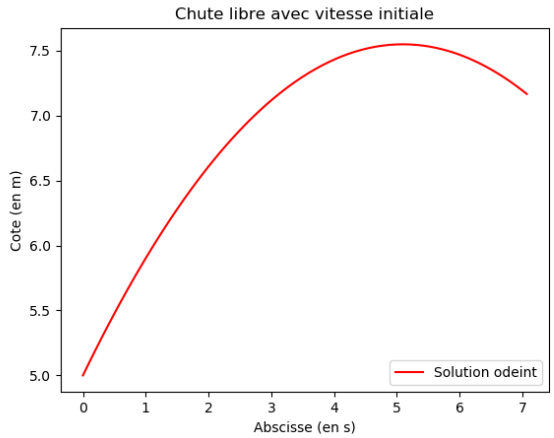

FIGURE 3.2

3.6 À vous de jouer

▶ **Question 1.** En vous aidant du dernier script, tracer l'hodographe du corps. Le résultat est *a priori* évident.

▶ **Question 2.** Comment modifier le script précédent en présence de frottements fluides ?

Activité 4

Oscillateurs

4.1 Position du problème

Nous sommes à présent en mesure d'étudier des situations physiques plus complexes que la chute libre. L'étude des oscillations occupe une place importante en physique. Cette activité vise à réinvestir les compétences acquises dans les précédentes activités en vue de tracer des lois horaires et des portraits de phase. Une ouverture vers les oscillations non-linéaires permet de dépasser le cadre des situations pour lesquelles une solution analytique existe

4.2 Notations

On s'intéresse tout d'abord au mouvement à un seul degré de liberté d'un point matériel soumis à une force de rappel élastique et à une force de frottements fluides. Le corps n'est soumis à aucune autre force. Cette situation peut être modélisée par l'équation différentielle du second ordre suivante.

$$\forall t \in \mathbb{R}^+, \quad \ddot{x}(t) + 2\zeta\omega_0\dot{x}(t) + \omega_0^2 x(t) = 0 \qquad (4.1)$$

Dans cette relation, ω_0 désigne une *pulsation caractéristique* et ζ est une quantité positive sans dimension, appelée *taux d'amortissement*.
Les *conditions initiales* sont notées :

$$x(0) = x_0 \qquad \dot{x}(0) = v_0$$

En posant $p(t) = \dot{x}(t)$, l'équation (4.1) est équivalente au système différentiel du premier ordre suivant :

$$\forall t \in \mathbb{R}^+, \quad \begin{cases} \dot{x}(t) = p(t) \\ \dot{p}(t) = -2\zeta\omega_0 p(t) + \omega_0^2 x(t) \end{cases}$$

muni des conditions initiales :

$$x(0) = x_0 \qquad p(0) = v_0$$

L'objet de cette activité est de résoudre numériquement ce système puis d'étudier des systèmes oscillants plus complexes. L'outil numérique se révèlera ici particulièrement efficace pour atteindre ces objectifs.

4.3 À vous de jouer

▶ **Question 1.** Proposer un script complet qui donne l'évolution temporelle de x pour différentes valeurs de ζ. Ces valeurs seront choisies de manière à mettre en évidence différents comportements oscillatoires. Les courbes seront tracées sur un même graphique.

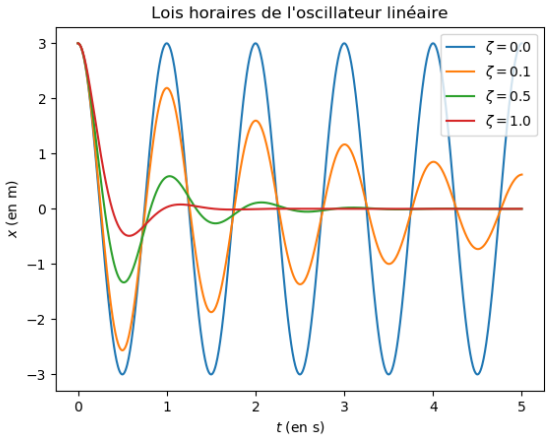

FIGURE 4.1

▶ **Question 2.** Tracer le portrait de phase de l'oscillateur en complétant le code précédent.

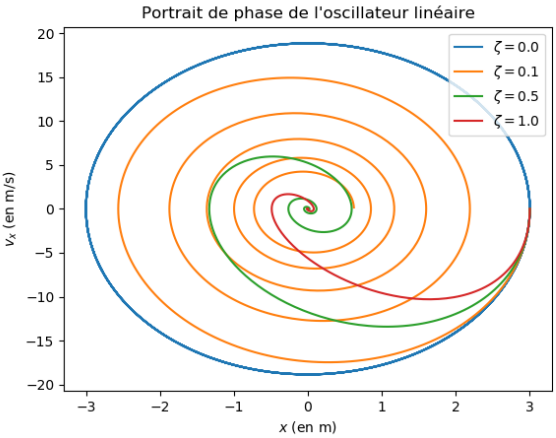

FIGURE 4.2

▶ **Question 3.** Pour disposer de plusieurs tracés dans une même fenêtre graphique, Python dispose de la commande `subplot`. Le script suivant illustre son utilisation.

```
import numpy as np
import matplotlib.pyplot as plt
# fonction à tracer
def f(x,k):
  return np.cos(k * x)
def g(x,k):
  return np.sin(k * x)
# données de tracé
x_min = 0.0
x_max = 2 * np.pi
n_x = 100
tab_x = np.linspace(x_min,x_max,n_x)
```

```
# liste de valeurs de k
lst_k = [1,2,3]
# boucle de tracés
# pour différentes valeurs de k
for k in lst_k:
  plt.subplot(2,1,1)
  y = f(tab_x,k)
  plt.plot(tab_x,y,label="$k = " + str(k)
   ↪ +"$")
  plt.subplot(2,1,2)
  y = g(tab_x,k)
  plt.plot(tab_x,y,label="$k = " + str(k)
   ↪ +"$")
plt.show()
```

La figure 4.3 illustre l'exécution de ce script.

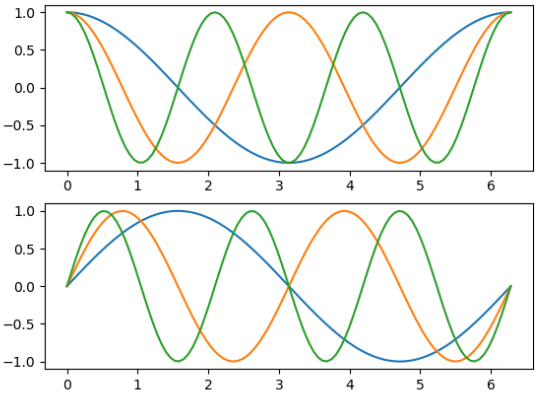

FIGURE 4.3

Proposer un script qui affiche dans une même fenêtre, l'évolution temporelle de x, le portrait de phase et, sur un même graphique, l'évolution temporelle des énergies cinétique, potentielle et mécanique.

▶ **Question 4.** Proposer une modélisation des oscillations d'un pendule. Exploiter toutes vos compétences numériques pour présenter une solution numérique du problème.

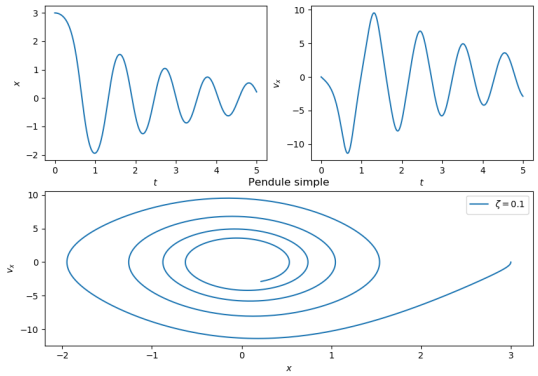

FIGURE 4.4

▶ **Question 5.** Le système est forcé par une excitation sinusoïdale. Illustrer le phénomène de résonance.

▶ **Question 6.** L'oscillateur de Van der Pol peut être modélisé par une équation différentielle du second ordre de la forme :

$$\forall t \in \mathbb{R}^+, \quad \ddot{x}(t) - \varepsilon\omega_0\left[1 - x^2(t)\right]\dot{x}(t) + \omega_0^2 x(t) = 0$$

Illustrer son comportement en fonction des paramètres du problème. En particulier, mettre en évidence l'existence d'un cycle limite indépendant des conditions initiales.

Activité 5

Canon de newton

5.1 Position du problème

On attribue à Newton une expérience de pensée dont l'un des objectifs premiers était de montrer que la gravitation constituait la composante majeure de la force de pesanteur. Dans cette expérience, Newton place un canon au sommet d'une « très haute » montagne. Ce canon tire des boulets qui, suivant leurs vitesses initiales, soit tombent sur la Terre, soit se satellisent, soit partent dans l'espace.

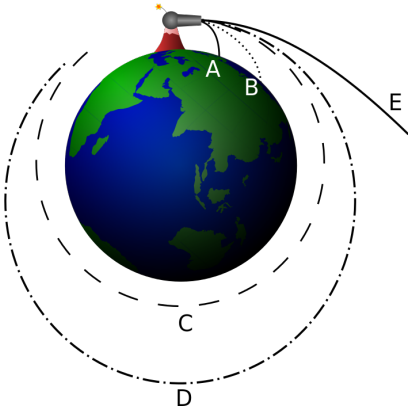

FIGURE 5.1 – Retombée sur Terre (A, B) - Satellisation (C, D) - Vers l'infini et au-delà (E)

L'objet de cette activité est de simuler numériquement cette expérience de pensée. Il sera l'occasion de confronter les vitesses de mise en orbite d'un corps. De nombreux sites web proposent une illustration de cette expérience de pensée. Vous pouvez par exemple tester le site suivant : `https://sites.google.com/site/physicsflash/home/gravity`.

5.2 Mise en équation

Le problème précédent est une application directe des équations régissant le mouvement d'un corps dans le champ gravitationnel d'un corps massif immobile, en l'occurence la Terre dans le cas présent. En désignant par M la position du centre de masse du boulet et par O le centre de la Terre, les lois de la mécanique permettent d'établir les résultats suivants.

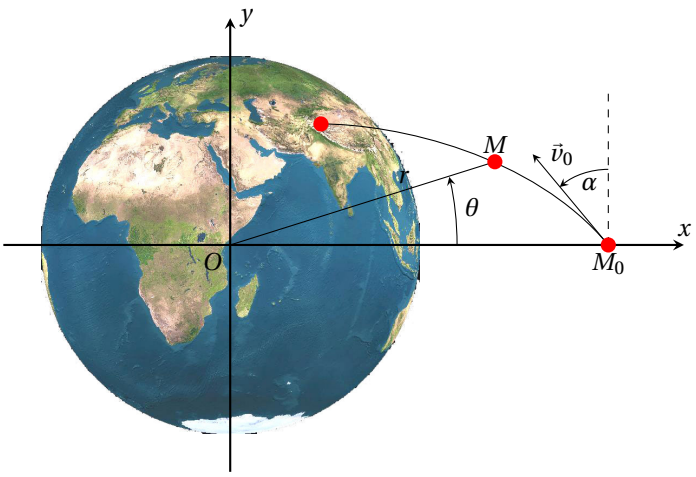

FIGURE 5.2 – Notations

- Le mouvement de M est plan. Le plan du mouvement est orthogonal au vecteur moment cinétique constant $\vec{L}$ et passe par O. Par choix, ce plan constitue le plan (xOy), l'axe (Oz) étant défini comme l'axe porté par $\vec{L}$.
- Dans le plan du mouvement, le point M peut être repéré par ses coordonnées polaires (r,θ). Celles-ci vérifient les équations dynamiques suivantes.

$$\ddot{r} = \frac{L^2}{m^2 r^3} - \frac{GM}{r^2} \qquad \dot{\theta} = \frac{L}{mr^2}$$

  où G désigne la constante de gravitation universelle.
- Ce sont ces équations dont il convient de donner une solution numérique afin de construire la trajectoire du point M.

5.3 À vous de jouer

La Terre est assimilée à une sphère de rayon R, de masse M. Son centre définit l'origine d'un référentiel supposé galiléen. En outre, on suppose la Terre immobile.
Un canon est placé à une altitude h. Des boulets sont tirés avec un vecteur vitesse initial $\vec{v}_0$ dont la norme et la direction peuvent être modifiées à souhait. Chaque boulet a une masse m et est soumis à la seule force de gravitation exercée par la Terre.
Le travail demandé pourra se décomposer de la manière suivante.

▶ **Question 1.** Adimensionner les équations du mouvement.

▶ **Question 2.** Définir le système différentiel et les conditions initiales associées au problème adimensionné.

▶ **Question 3.** Résoudre numériquement le système.

▶ **Question 4.** Tracer la planète et les trajectoires des boulets.

▶ **Question 5.** Déterminer les conditions sur $\vec{v}_0$ (direction et intensité) pour observer la retombée du boulet sur la Terre ou sa mise en orbite. Comparer les valeurs aux valeurs connues sur Terre (vitesse de satellisation, vitesse de libération).

5.4 Données numériques

Masse de la Terre
$M = 5{,}98 \times 10^{24}\,\mathrm{kg}$

Rayon terrestre
$R = 6370\,\mathrm{km}$

Constante de gravitation universelle
$G = 6{,}67 \times 10^{-11}\,\mathrm{m^3 \cdot kg^{-1} \cdot s^{-2}}$

Vitesse de satellisation minimale — *La vitesse de satellisation minimale est la vitesse théoriquement communiquée à un corps au départ d'un astre pour le satelliser au plus près de ce dernier sur une orbite circulaire.*
$v_s = 7{,}9\,\mathrm{km \cdot s^{-1}}$

Vitesse de libération — *La vitesse de libération est la vitesse minimale que doit atteindre un corps pour échapper définitivement à l'attraction gravitationnelle d'un astre (planète, étoile, etc.) et s'en éloigner indéfiniment.*
$v_s = 11{,}2\,\mathrm{km \cdot s^{-1}}$

Vitesse de la lumière dans le vide
$c = 2{,}998 \times 10^{8}\,\mathrm{m \cdot s^{-1}}$

Activité 6

Petit vade-mecum

6.1 Tracé d'une courbe à partir d'une fonction

Tracé de la courbe représentative de la fonction $f : x \mapsto e^{-x^2}$ sur un intervalle $[x_{min}, x_{max}]$. Par défaut, Python trace une courbe en reliant les points calculés. Noter l'appel aux deux modules `numpy` et `matplotlib.pyplot`.

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.exp(-x * x)

x_min, x_max = -4.0, 5.0
n_x = 100
x = np.linspace(x_min,x_max,n_x)
y = f(x)

plt.plot(x,y)
plt.show()
```

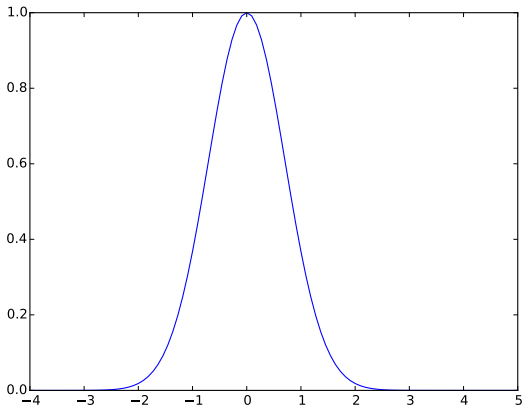

FIGURE 6.1

Pour faire apparaître les points, on peut ajouter une option à `plt.plot`.

- `'b-'` trace la courbe en bleu (option `b`) et relie les points (option `-`).
- `'ro'` affiche des points (option `o`) en rouge (option `r`).

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.exp(-x * x)

x_min, x_max = -4.0, 5.0
n_x = 100
x = np.linspace(x_min,x_max,n_x)
y = f(x)

plt.plot(x,y)
plt.show()
```

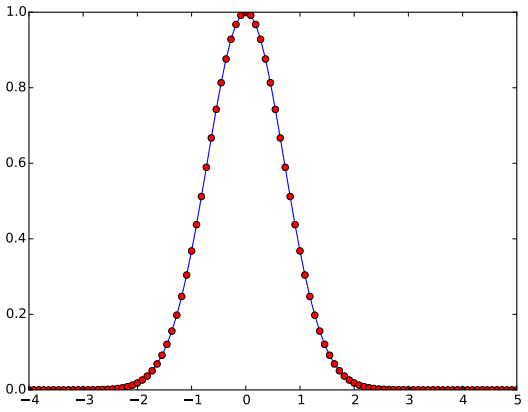

FIGURE 6.2

6.2 Tracé d'une courbe à partir de données tabulaires

En physique, les données numériques sont d'origine expérimentales. Obtenues à partir de mesures (acquisitions de données via un CAN), elles peuvent être enregistrées dans un fichier. Python peut lire les données pour ensuite afficher une courbe.
Le format `.csv` est fréquemment adopté pour enregistrer des données ou les exporter depuis un logiciel. En pratique, les données sont stockées dans un fichier sous forme de flottants, séparés par un point virgule. Le module `csv` de Python fournit quelques fonctions pratiques pour manipuler les fichiers `.csv`.
Dans l'exemple suivant, le fichier de données `data.csv` contient des lignes de deux flottants séparés par un point-virgule.

```
import matplotlib.pyplot as plt
import csv

source = open('data.csv', 'r')

x, y = [], []
for row in csv.reader(source,delimiter=';'):
  x1, y1 = map(float,row)
  x.append(x1)
  y.append(y1)
```

```
plt.plot(x,y,'b-')
plt.plot(x,y 'ro')
plt.show()
```

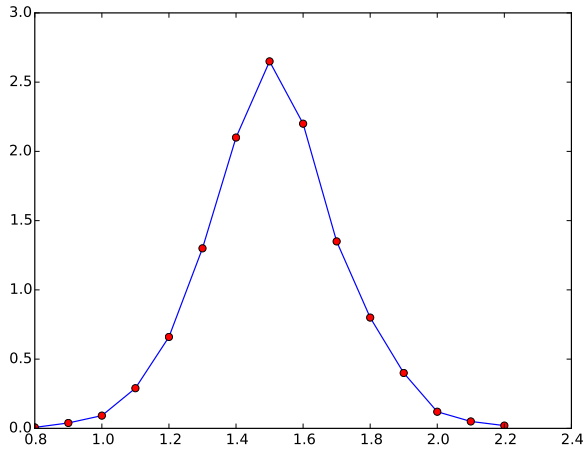

FIGURE 6.3

```
plt.plot(xFit,yFit,label='fit',color='r')
plt.plot(x,y,'bo')
plt.show()
```

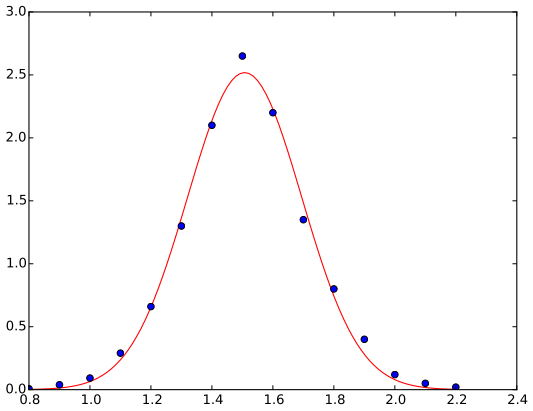

FIGURE 6.4

6.3 Ajustements

Il est parfois nécessaire de déterminer un modèle mathématique rend compte de l'évolution d'un phénomène physique. Par exemple, une série de mesures a fourni un ensemble de couples de points (x_i, y_i) qui semblent s'aligner. Comment déterminer l'équation de la droite de régression associée? Python dispose d'une fonction `curve_fit` du module `from scipy.optimize` qui permet tout type d'ajustement : ajustement linéaire, ajustement gaussien, etc. L'exemple suivant présente un ajustement gaussien de données. Il reprend une partie du code vu ci-dessus. Noter la conversion des données en tableau (fonction `ar`).

```
import matplotlib.pyplot as plt
import csv
import numpy as np
from scipy.optimize import curve_fit
from scipy import asarray as ar

def gauss(x,a,x0,sigma):
    return a*np.exp(- (x - x0)**2 / sigma**2
    ↪  / 2)

source = open('data.csv','r')
x, y = [], []
for row in csv.reader(source,delimiter=';'):
  x1, y1 = map(float,row)
  x.append(x1)
  y.append(y1)

x, y = ar(x), ar(y)
xmin, xmax, n_x = min(x), max(x), 100

popt, pcov = curve_fit(gauss,x,y)
xFit = np.linspace(xmin,xmax,n_x)
yFit = gauss(xFit,*popt)
```