Méthode d'Euler pour la résolution d'équations différentielles

C. Charignon

Table des matières

Première partie
Cours

1 Notations

Soit I un intervalle et $F : \mathbb{R} \times I \to \mathbb{R}$. Soit (E) l'équation :

$$(E) : \quad \forall t \in \mathbb{R},\ f'(t) = F\big(f(t), t\big)\,.$$

d'inconnue $f \in \mathcal{C}^1(I, \mathbb{R})$.
Remarque :

- La plupart des équation différentielles que vous rencontrez en prépa rentrent dans ce cadre. Mais par exemple l'équation $e^{f'(t)} + \cos(f'(t)) = 2f(t) + \ln(t)$ non !
- On suppose pour simplifier F définie sur $\mathbb{R} \times I$, mais en pratique la question du domaine de définition peut se poser.

On fixe $h \in \mathbb{R}^{+*}$ qu'on appellera le pas de résolution. On fixe $t_0 \in I$ et y_0. On peut démontrer (théorème de Cauchy-Lipschitz) que, à condition que F soit suffisamment régulière (localement lipschitzienne selon sa première variable), (E) admet une unique solution f vérifiant en outre $f(t_0) = y_0$. On suppose ces hypothèses vérifiées, et on conserve la notation f pour la suite.

Le but est de calculer une valeur approchée de f.

Pour tout $i \in \mathbb{N}$, on pose $t_i = t_0 + ih$. On notera y_i la valeur approchée de $f(t_i)$ qu'on va calculer.

2 Description de la méthode d'Euler

2.1 Principe

On utilise un développement limité d'ordre 1 :

$$\begin{aligned} \forall t \in \mathbb{R}, \quad f(t+h) &\underset{h\to 0}{=} f(t) + hf'(t) + O(h^2) \\ &\underset{h\to 0}{=} f(t) + hF\big(f(t), t\big) + O(h^2). \end{aligned}$$

Ainsi pour tout $i \in \mathbb{N}$,

$$f(t_{i+1}) \underset{h\to 0}{=} f(t_i) + hF(f(t_i), t_i) + O(h^2).$$

Ce qui nous incite à définir la suite $\big(y_i\big)_{i\in\mathbb{N}}$ par :

$$\begin{cases} y_0 = y_0 \\ \forall i \in \mathbb{N}, \quad y_{i+1} = y_i + hF(y_i, t_i) \end{cases}$$

En gros, on fait donc une erreur de l'ordre de $O(h^2)$ à chaque étape ; l'étude mathématique de la précision de la méthode d'Euler sera menée partie 4.

2.2 Programmation

J'ai choisi ci-dessous de prendre en entrée t_f l'instant de fin de simulation et h la taille d'un pas. On pourrait aussi prendre le nombre de pas en entrée (si N est ce nombre, poser $h = \frac{t_f - t_0}{N}$), mais c'est moins pratique pour l'utilisateur final.

```
def euler(t0, tf, y0, F,h):
    t, y = t0, y0
    tt=[t]
    ty=[y]
    while t<tf:
        y+= h*F(y, t) # C'est ici qu'on utilise le DL1
        t+=h
        tt.append(t)
        ty.append(y)
    return tt, ty
```

Pour tracer la courbe :

```
import matplotlib.pyplot as plt
def dessin(t0, tf, y0, F,h):
    tt, ty = euler(t0, tf, y0, F,h)
    plt.plot(tt, ty, label="Approx par Euler")
    plt.legend()
```

Il faudra après l'appel à `dessin` encore lancer une `plt.show()` pour voir le graphe. Ceci permet d'afficher plusieurs graphes dans la même figure.

3 Exemples simples

3.1 Exponentielle

Appliquons la méthode d'Euler pour résoudre l'équation $f' = f$ avec la condition initiale $f(0) = 1$. Cette fois nous prenons $F : (a, b) \mapsto a$.

Pour appliquer notre programme :

```
dessin(0, 2, 1, lambda (a,b):a, 0.1)
dessin(0, 2, 1, lambda (a,b):a, 0.01)
plt.show()
```

Analysons mathématiquement cette situation. On obtient la suite $(y_i)_{i\in\mathbb{N}}$ définie par : $\begin{cases} y_0 = 1 \\ \forall i \in \mathbb{N},\ y_{i+1} = y_i + hy_i \end{cases}$.

D'où pour tout $i \in \mathbb{N}$, $y_i = (1 + h)^i$.

Par exemple, pour approcher $f(1)$ (qui devrait donc valoir e), prenons $n \in \mathbb{N}^*$, $h = \frac{1}{n}$ et calculons y_n. Ce devrait être une approximation de $f(0 + n \times h)$, c'est-à-dire $f(1)$.

On obtient alors $\forall n \in \mathbb{N}$, $y_n = (1+\frac{1}{n})^n$. Un calcul classique de limites montre que $\lim_{n\to\infty} y_n = \mathrm{e}$. Ainsi, la méthode d'Euler fonctionne sur cet exemple simple.

3.2 Calcul d'une primitive

Soit u une fonction continue sur I dont nous voulons calculer une primitive. Cela revient à résoudre l'équation $f' = u$. Prenons par exemple la primitive qui s'annule en t_0, c'est-à-dire la fonction $x \mapsto \int_{t_0}^{x} f(t)\mathrm{d}t$.

Nous pouvons donc appliquer notre programme à la fonction $F : (a, b) \mapsto u(b)$.

On obtiendra une suite $(y_i)_{i\in\mathbb{N}}$ définie par $\begin{cases} y_0 = 0 \\ \forall i \in \mathbb{N},\ y_{i+1} = y_i + u(t_i) \end{cases}$.

D'où $\forall n \in \mathbb{N}$, $y_n = \sum_{i=0}^{n-1} u(t_i)$.

On reconnaît alors la méthode des rectangles à gauche !

En particulier, on sait que y_n fournit une approximation de $\int_{t_0}^{t_n} u(t)\mathrm{d}t$ c'est-à-dire de $f(t_n)$. On sait de plus que la précision est en $O\big(h(b-a)\big)$ à condition que f soit de classe $\mathcal{C}^1$.

La méthode d'Euler fonctionne donc dans ce cas simple.

4 Convergence

Passons à la partie technique : démontrons que la méthode d'Euler fonctionne, au moins avec certaines conditions sur F. Et dans le même temps, calculons sa précision.

Notations :

- $(y_n)_{n\in\mathbb{N}}$ la suite des valeurs approchées de $f(t_n)$ obtenue par la méthode d'Euler
- $\forall n \in \mathbb{N}$, $e_n = |y_n - f(t_n)|$ l'erreur à l'instant t_n. C'est ce que nous voulons estimer.
- Pour tout $x \in I$, notons R_x le reste de la formule de Taylor à l'ordre 1 en x. Ainsi, nous avons pour tout $x \in I$ et tout $h \in \mathbb{R}^{+*}$, $f(x + h) = f(x) + hf'(x) + R_x(h)$, et nous savons que $R_x(h) = O_{h\to0}(h^2)$.
  Notons également, L la constante cachée dans le $O(h^2)$. Autrement dit, on a $\forall x \in I$, $\forall h \in \mathbb{R}^{+*}$ tel que $x + h \in I$, $\big|R_x(h)\big| \leqslant Lh^2$.
  Remarque : Par la formule de Taylor-Lagrange, $L = \frac{1}{2}\left\|f\right\|_\infty$.

En outre, on suppose qu'il existe une constante K telle que $\forall t \in I,\ \forall x, y \in \mathbb{R},\ \big|F(x,t) - F(y,t)\big| \leqslant K|y-x|$. En gros, F est lipschitzienne selon sa première variable, et la constante de lipschitzianité ne dépend pas de t. On fixe cette constante K dans la suite.

Commençons par écrire les relations de récurrences vérifiées par les suites $\big(y_n\big)_{n\in\mathbb{N}}$ et $\big(f(t_n)\big)_{n\in\mathbb{N}}$:

$$\forall n \in \mathbb{N},\ \begin{cases} y_{n+1} = y_n + hF\big(y_n, t_n\big) \\ f(t_{n+1}) = f(t_n) + hF\big(f(t_n), t_n\big) + R_{t_n}(h) \end{cases}.$$

Une soustraction nous fournit la relation de récurrence vérifiée par e : $\forall n \in \mathbb{N}$:

$$\begin{aligned} y_{n+1} - f(t_{n+1}) &= y_n - f(t_n) + h\left(F\big(y_n, t_n\big) - F\big(f(t_n), t_n\big)\right) + R_{t_n}(h) \\ \text{donc } \Big|y_{n+1} - f(t_{n+1})\Big| &\leqslant \big|y_n - f(t_n)\big| + h\Big|F\big(y_n, t_n\big) - F\big(f(t_n), t_n\big)\Big| + \Big|R_{t_n}(h)\Big| \\ \text{donc } e_{n+1} &\leqslant e_n + hKe_n + Lh^2. \end{aligned}$$

Finalement,

$$\forall n \in \mathbb{N}, e_{n+1} \leqslant (1+Kh)e_n + Lh^2. \tag{1}$$

On constate que e est inférieure à une suite arithmético-géométrique. On peut donc utiliser les méthodes classiques, à savoir :

- trouver une constante λ vérifiant la relation de récurrence, puis étudier $e - \lambda$; (λ est un point fixe de la fonction itératrice...)
- ou étudier la suite $\left(\dfrac{e_n}{(1+Kh)^n}\right)_{n\in\mathbb{N}}$ (variation de la constante)

Remarque : Le terme Lh^2 s'appelle l'erreur de « consistance ». Le fait que F est globalement K-lipschitzienneévite que l'erreur de consistance ne soit démesurément amplifiée à chaque étape. On dit que la méthode est « stable ».

J'utilise ci-dessous la deuxième méthode : la relation de récurrence «homogène» serait ici $e_{n+1} \leqslant (1+Kh)e_n$ qui a pour solution $cte(1+Kh)^n$. La méthode de la variation de la constante consiste alors à étudier $\frac{e_n}{(1+Kh)^n}$. Bref, je divise la relation de récurrence 1 par $(1+Kh)^{n+1}$:

$$\forall n \in \mathbb{N},\ \frac{e_{n+1}}{(1+Kh)^{n+1}} \leqslant \frac{e_n}{(1+Kh)^n} + \frac{Lh^2}{(1+Kh)^{n+1}}$$

D'où :

$$\forall n \in \mathbb{N},\ \frac{e_n}{(1+Kh)^n} \leqslant \sum_{k=1}^{n} \frac{Lh^2}{(1+Kh)^k}$$

La somme est une brave somme géométrique de raison $\frac{1}{(1+Kh)^k}$. Elle vaut $Lh^2\dfrac{1-(1+Kh)^{-(n+1)}}{1-(1+Kh)^{-1}}$.

$$\forall n \in \mathbb{N},\ e_n \leqslant (1+Kh)^n\,\frac{Lh}{K} - \frac{Lh}{K} = h\frac{L}{K}\left((1+Kh)^n - 1\right).$$

On trouve au final :

En utilisant la majoration $\forall x \in \mathbb{R}^{+*}$, $\ln(1+x) \leqslant x$, on obtient la forme un peu plus simple suivante :

$$\forall n \in \mathbb{N},\ e_n \leqslant h\frac{L}{K}\left(\mathrm{e}^{nKh} - 1\right).$$

Par exemple, appliquons cette formule au dernier pas de la simulation : notons $t_f = n \times h$. L'erreur est alors majorée par $h\frac{L}{K}\left(\mathrm{e}^{Kt_f} - 1\right)$.

Si nous fixons t_f, c'est une erreur en $O(h)$.

En revanche, si on fixe h et qu'on considère t_f comme paramètre, l'erreur est en $O(\mathrm{e}^{t_f})$! La méthode n'est pas fiable sur des temps longs...

Remarque : On verra en exercice d'autres formules que la formule d'Euler, plus précises. Supposons qu'on trouve g telle qu'il existe $p > 2$ tel que pour tout $n \in \mathbb{N}$, $f(t+h) = g\big(f(t), t, h\big) + O(h^p)$. On définira alors la suite $\big(y_n\big)_{n\in\mathbb{N}}$ par $y_{n+1} = g\big(y_n, t_n, h\big)$.

Pour peu qu'il existe une constante K telle que g vérifie $\forall (a,b) \in \mathbb{R},\ \forall t \in \mathbb{R},\ |g(a,t,h) - g(b,t,h)| \leqslant |b-a|\,(1+Kh)$, on peut alors reprendre les calculs précédents, et pour peu que g soit lipschitzienne, on obtiendra que l'erreur finale est en $O(h^{p-1})$.

5 Équation d'ordre 2

La plupart des équations rencontrées en physique, notamment celles venant du principe fondamental de la dynamique, sont d'ordre 2. Voyons comment adapter la méthode d'Euler à cette situation.

Nous gardons les notations I, h, et $(t_i)_{i\in\mathbb{Z}}$ des parties précédentes.
Nous fixons dans cette partie une fonction $F : \mathbb{R}\times\mathbb{R}\times I \to \mathbb{R}$, et nous appelons (E) l'équation suivante :

$$\forall t \in I,\ f''(t) = F\big(f'(t), f(t), t\big)$$

d'inconnue $f \in \mathcal{C}^2(I,\mathbb{R})$.

Ainsi, pour calculer $f''(t)$ il nous faut connaître $f(t)$, t et $f'(t)$.
Le plus simple est de calculer en même temps des valeurs approchées des $f(t_i)$ mais aussi des $f'(t_i)$. Et pour ce, nous utilisons les deux DL1 suivant :

$$\forall i\in\mathbb{N},\quad \begin{cases} f(t_i+h) \underset{h\to 0}{=} f(t_i) + hf'(t_i) + O(h^2) \\ f'(t_i+h) \underset{h\to 0}{=} f'(t_i) + hf''(t_i) + O(h^2) \end{cases}$$

Le second se réécrit ainsi :

$$f'(t_i+h) \underset{h\to 0}{=} f'(t_i) + hF\big(f'(t_i), f(t_i), t_i\big) + O(h^2).$$

On définit alors deux suites par récurrence. La première $(y_n)_{n\in\mathbb{N}}$ contiendra comme précédemment les valeurs approchées de $(f(t_n))_{n\in\mathbb{N}}$. La seconde, qu'on notera $(z_n)_{n\in\mathbb{N}}$ contiendra les valeurs approchées de $(f'(t_n))_{n\in\mathbb{N}}$. On les définit par :

$$\begin{cases} y_0 = f(t_0) \text{ et } z_0 = f'(t_0) \\ \forall i\in\mathbb{N}, \begin{cases} y_{i+1} = y_i + hz_i \\ z_{i+1} = z_i + hF(z_i, y_i, t_i) \end{cases} \end{cases}$$

Notons que nous devons connaître $f(t_0)$ et $f'(t_0)$: la condition initiale pour une équation d'ordre 2 porte aussi sur la dérivée.

On peut imaginer le code suivant :

```
def eulerOrdre2(t0,tf, y0, z0, F):
    y, z = y0, z0
    t=t0
    tt, ty, tz = [t], [y], [z]
    while t < tf:
        t+=h
        y+=h*z
        z+= h* F(z, y, t)
        tt.append(t)
        ty.append(y)
        tz.append(z)
    return tt, ty, tz
```

Remarque : J'ai choisi de garder en mémoire, et de renvoyer aussi les valeurs de la dérivée. Si le but est juste de tracer la courbe de f, c'est inutile.

Mais ce code ne correspond pas véritablement à la méthode d'Euler ! En effet, `y` est mis à jour avant `z`, donc au moment de mettre `z` à jour, on utilise la nouvelle valeur de `y`, soit y_{i+1} si on a déjà fait i tours de boucle au lieu de y_i.
Autrement dit les suites obtenues vérifient :

$$\begin{cases} y_0 = f(t_0) \text{ et } z_0 = f'(t_0) \\ \forall i\in\mathbb{N}, \begin{cases} y_{i+1} = y_i + hz_i \\ z_{i+1} = z_i + hF(z_i, y_{i+1}, t_i) \end{cases} \end{cases}$$

En fait, ce code n'est pas moins efficace que la vraie méthode d'Euler, au contraire. On l'appelle la méthode d'Euler « semi-implicite ». Néanmoins, si nous voulons programmer la vraie méthode d'Euler, nous pouvons proposer le code suivant, sachant que pour tout tableau `t`, la commande `t[-1]` renvoie le dernier élément de `t`.

```python
def eulerOrdre2(t0,tf, y0, z0, F):
    y, z = y0, z0
    t=t0
    tt, ty, tz = [t], [y], [z]
    while t < tf:
        y+=h*z
        z+= h* F(z, ty[-1], t)
        t+=h
        tt.append(t)
        ty.append(y)
        tz.append(z)
    return tt, ty, tz
```

6 Système d'équations

Il faut également savoir gérer le cas d'un système d'équations différentielles. C'est le même principe qu'au paragraphe précédent puisque finalement une équation d'ordre 2 est un système d'équations où les inconnues sont f et f'.

Voir la première partie du premier TP.

7 Vectorisation, et introduction des tableaux numpy

On peut proposer une écriture un peu plus compacte des codes précédents (équations d'ordre $\geqslant 2$ ou systèmes d'équations).

On traite ci-dessous l'exemple d'une équation d'ordre 2.

L'idée est de rassembler `y` et `z` dans un tableau de deux éléments. Mathématiquement, cela revient à utiliser le vecteur $\begin{pmatrix} y_i \\ z_i \end{pmatrix}$. On a la relation de récurrence suivante :

$$\forall i \in \mathbb{N}, \quad \begin{pmatrix} y_{i+1} \\ z_{i+1} \end{pmatrix} = \begin{pmatrix} y_i \\ z_i \end{pmatrix} + h \cdot \begin{pmatrix} z_i \\ F(z_i, y_i, t_i) \end{pmatrix}.$$

Remarque : Au niveau du développement limité, on a

$$\begin{aligned} \begin{pmatrix} f(t_i+h) \\ f'(t_i+h) \end{pmatrix} &\underset{h\to 0}{=} \begin{pmatrix} f(t_i) \\ f'(t_i) \end{pmatrix} + h \begin{pmatrix} f'(t_i) \\ f''(t_i) \end{pmatrix} + O(h^2) \\ &\underset{h\to 0}{=} \begin{pmatrix} f(t_i) \\ f'(t_i) \end{pmatrix} + h \begin{pmatrix} f'(t_i) \\ F\big(f'(t_i), f(t_i), t_i\big) \end{pmatrix} + O(h^2). \end{aligned}$$

On pose $G : \left(\begin{pmatrix} a \\ b \end{pmatrix}, t \right) \mapsto \begin{pmatrix} b \\ F(b,a,t) \end{pmatrix}$, et on note $\forall i \in \mathbb{Z}$, $X_i = \begin{pmatrix} y_i \\ z_i \end{pmatrix}$. Alors notre relation de récurrence devient :

$$\forall i \in \quad X_{i+1} = X_i + hG(X_i, t_i)$$

C'est donc exactement la même formule que pour une équation d'ordre 1, simplement avec la fonction G au lieu de F.

De même, si on note $\forall t \in I$, $X(t) = \begin{pmatrix} f(t) \\ f'(t) \end{pmatrix}$, l'équation (E) devient :

$$\forall t \in I, \quad X'(t) = G\big(X(t), t\big).$$

De cette manière, n'importe quelle équation d'ordre 2 peut être considérée comme une équation d'ordre 1, mais dont l'inconnue est une fonction vectorielle.

Au niveau du code, on peut imaginer :

```python
def eulerOrdre2(t0, tf, y0, z0, F):

    def G(X, t):
       return [ X[1], F(X[1], X[0], t)]

    X=[y0, z0]
    t=t0
    tX, tt = [X], [t]
    while t < tf:
        X = X + h * G(X, t)
        t+=h
        tt.append(t)
        tX.append(X)
    return tt, tX
```

Remarque : On pourrait aussi laisser à l'utilisateur le travail de définir lui-même la fonction G.

Mais ça ne fonctionne pas ! En effet, les opérations $+$ et $*$ sur les tableaux ne sont pas les opérations mathématiques sur les vecteurs.

C'est le moment de parler des tableaux de numpy.

8 Interlude : les tableaux numpy

La bibliothèque numpy crée un nouveau type de tableaux (type `ndarray`). Voici ses principales caractéristiques :

- Non redimensionnables : pas de append ni de pop.
- Typés : le contenu de toutes les cases doit être du même type, et ce type ne pourra être changé par la suite.
- Les opérations `+` et `*` ne sont pas programmées de la même manière pour les tableaux numpy. En effet, `+` est l'addition terne à terme, comme en maths. De même `*` implémente le produit par une constant ou le produit terme selon qu'on l'utilise entre deux tableaux ou entre un tableau et un nombre.

En pratique :

- Lorsque vous utiliserez un tableau numpy, vous devrez à l'avance créer le bon nombre de cases. Utiliser `np.zeros` pour créer un tableau rempli de zéros avec un nombre de case prédéfini.
- La fonction de conversion permettant de passer d'un tableau quelconque à un tableau numpy est `np.array`.

Programmation : Maintenant, programmons une version vectorisée de la méthode d'Euler d'ordre 2.

9 Variantes de la méthode d'Euler

9.1 Euler implicite

Voici un exemple de variante de la méthode d'Euler. Pour tout $i \in \mathbb{N}$, au lieu d'utiliser un développement limité en t_i pour estimer $f(t_i + h)$, on va utiliser un développement limité basé en t_{i+1}. Autrement dit, on utilise :

$$f(t_i + h) \underset{h\to 0}{=} f(t_i) + hf'(t_i + h) + O(h^2).$$

Ainsi, nous définissons cette fois la suite $\big(y_n\big)_{n\in\mathbb{N}}$ ainsi :

$$\begin{cases} y_0 = f(t_0) \\ \forall n \in \mathbb{N},\ y_{n+1} = y_n + hF(y_{n+1}, t_{n+1}) \end{cases}$$

Mais cette relation de récurrence ne permet pas de calculer immédiatement y_{n+1} en fonction de y_n ! Il s'agit d'une équation à résoudre pour trouver y_{n+1}. Pour la résoudre on peut utiliser la méthode de Newton. Il faudra alors envoyer au programme `newton` vu au pénultième chapitre la fonction $x \mapsto y_n - hF(x, t_{n+1}) - x$. Bien entendu comme point de départ on choisit y_n. Comme paramètre de précision, on peut utiliser h.

Ceci donne :

```
def eulerImplicite(t0, tf, y0, F,h):
    t, y = t0, y0
    tt=[t]
    ty=[y]
    while t<tf:
        t+=h
        y= newton( lambda x : y-h*F(x,t)-x, y, h)
        tt.append(t)
        ty.append(y)
    return tt, ty
```

La méthode d'Euler explicite est en général en retard sur la vraie solution, puisqu'elle utilise la dérivée à l'instant précédente. Au contraire, la méthode implicite sera en avance car elle utilise la dérivée à l'instant suivant. Les deux méthodes ont a priori le même ordre de grandeur de précision.

Cependant, la méthode d'Euler implicite est réputée être plus stable, c'est-à-dire moins amplifier les erreurs.

9.2 Schema prédicteure-correcteur

On améliore maintenant la méthode d'Euler implicite.

Intuitivement, la moyenne entre Euler implicite et explicite devrait être plus précise. Et en effet, les formules de Taylor-Young à l'ordre 2 nous donnent :

$$\forall i \in \mathbb{N},\ f(t_{i+1}) = f(t_i) + \frac{1}{2}\left(f'(t_i) + f'(t_{i+1})\right) + O(h^3).$$

Ainsi, cette méthode aura un ordre de précision de plus que les méthodes classiques.

N.B. Dans le cas du calcul d'une primitive, la méthode d'Euler explicite revient comme on l'a déjà dit à la méthode des rectangles à gauche. La méthode implicite aux rectangles à droite. Et celle ci-dessus revient à la méthode des trapèzes.

On peut l'implémenter ainsi :

```
def trapèzesImplicite(t0, tf, y0, F,h):
    t, y = t0, y0
    tt=[t]
    ty=[y]
    while t<tf:
        t+=h
        y= newton( lambda x : y - h/2*(F(x,t)+F(y, t-h)) - x, y, h)
        tt.append(t)
        ty.append(y)
    return tt, ty
```

Il existe aussi une variante permettant de se passer de Newton : dans la relation de récurrence $y_{n+1} = y_n + \frac{h}{2}\left(F(y_n, t_n) + F(y_{n+1}, t_{n+1}\right)$, on peut remplacer le y_{n+1} à droite par une estimation obtenue par un Euler explicite ordinaire. Autrement dit, on peut remplacer cette relation de récurrence par :

$$y_{n+1} = y_n + \frac{h}{2}\left(F(y_n, t_n) + F(y_n + hF(y_n, t_n), t_{n+1}\right).$$

On appelle ceci un schéma « prédicteur-correcteur ». La phase de prédiction est le premier calcul de y_{n+1} par Euler explicite, et la phase de correction est l'utilisation de la formule des trapèzes pour obtenir une meilleur estimation à partir de celle-ci.

Remarque : L'erreur de consistance de la méthode d'Euler est en $O(h^2)$. Ainsi, la première estimation de y_{n+1} est précise en $O(h^2)$. On lui applique F, qu'on a supposé lipschitzienne, l'erreur reste en $O(h^2)$. Puis on multiplie par h : l'erreur est alors en $O(h^3)$. L'erreur de consistance de la formule des trapèzes est aussi en $O(h^3)$, au final, la formule ci-dessus a une erreur en $O(h^3)$.

```
def prédicteur_correcteur(t0, tf, y0, F,h):
    t, y = t0, y0
    tt=[t]
    ty=[y]
    while t<tf:
```

```
        t+=h
        y = y + h/2 * (F(y,t) + F(y+h*F(y,t-h), t))
        tt.append(t)
        ty.append(y)
    return tt, ty
```

Le cours s'arrête ici en signalant que les variantes de la méthode d'Euler peuvent fournir de nouveaux sujet de concours pendant encore plusieurs années...

10 Un exemple physique simple

ç

10.1 Présentation du problème

Un objet ponctuel de masse m et de volume $\mathcal{V}$ est attaché à un point O, fixe dans un référentiel galiléen, par un ressort. Ce ressort peut s'étirer mais non se tordre. On note k sa constante de raideur et l_0 son élongation au repos.

À tout instant t, on note $M(t)$ la position de l'objet, $\vec{v}(t)$ sa vitesse, et $\vec{a}(t)$ son accélération. On notera aussi $l(t)$ la distance $OM(t)$.

Le tout est dans une bassine d'eau, j'appelle H l'ordonnée de la surface de l'eau. Lorsque l'objet est dans l'eau, il subit la poussée d'Archimède et une force de frottement égale à $-\alpha \left\|\vec{v}\right\| \vec{v}$, où α est une constante donnée.

Soit $(\vec{e}_x, \vec{e}_y, \vec{e}_z)$ une base orthonormée directe telle que $\vec{y} = -\frac{1}{g}\vec{g}$.

L'objet a une charge électrique q et il est dans un champ magnétique $\vec{B}$ dirigé par $\vec{e}_z$.

On suppose que $\vec{v}(0) \in \text{Vect}\left(\vec{e}_x, \vec{e}_y\right)$. Dès lors, le mouvement de M sera plan et contenu dans le plan $\left(0, \vec{e}_x, \vec{e}_y\right)$.

À chaque instant t, on note $(x(t), y(t))$ les coordonnées de $\vec{OM}(t)$ dans la base $\left(\vec{e}_x, \vec{e}_y\right)$.

On commence par définir les différentes constantes en variables globales.

```
g = 9.81
m = 1.5 # en kg
α = 0.1
l0 =.5 #en m
k = 2
h = -.2
q = .1 # en volt
vol = 1e-3 #en m^3
 = 1e3 #en kg/m^3
ey = np.array([0,1])
B = 0.11 # en tesla
```

10.2 Bilan des forces

```
poids =  np.array([0 , -m*g ])


def norme(u):
    """ Entrée : un vecteur u, de type ndarray
        Sortie : sa norme"""
    return np.sqrt(sum(u*u))

def frottement(v):
    """ Entrée : v le *vecteur* vitesse, de type ndarray """
    return  - * norme(v) * v

def rappel(M):
    """ Entrée : M, position de l'objet."""

    OM=M # Le vecteur OM a les mêmes coordonnées que le point M
    er= (1/norme(OM)) * OM
```

```
    l = norme(OM)

    return k*(l0-l)*er

def lorentz(v):
    """ Entrée : le vecteur vitesse v """
    return q * B * np.array( [v[1], - v[0]])

poussée_Archi =  *g*vol * ey
```

Remarquez que le poids et la poussée d'Archimède sont des constantes alors que les autres forces sont des fonctions.

10.3 Accélération

En appliquant le pfd, on obtient :

```
def accélération(M, v): # C'est le F du cours

    somme_forces =  lorentz(v) + rappel(M) + poids

    if M[1] < h  : #dans l'eau
         somme_forces += frottement(v)+ poussée_Archi

    #pfd
    return 1/m * somme_forces
```

Le fait d'avoir enregistré nos forces dans des tableaux numpy permet de les additionner par un simple `+`.

10.4 Euler

On programme alors Euler comme d'habitude. Le programme `accélération` ci-dessus va remplacer le `F` utilisé dans les premiers programmes de ce chapitre.

```
def eulerPenduleRessortMouillé(t0 , dt, tf, M0, v0):
    tt=[t0]
    tM=[M0]
    M=np.array(M0)
    v=np.array(v0)
    t=t0

    while t < tf :
        #nouvelle valeur de M càd M(t+dt).
        M = M + dt*v # Ne pas utiliser += lorsque M est un tableau numpy. Sinon on va garder
            ↪ toujours le même tableau, et tM contiendra len(tM) fois la même chose
        tM.append(M)

        #nouvelle valeur de v
        v = v + dt*accélération(M,v)
        # Rema : c'est en fait la méthode d'Euler semi-implicite car on utilise la nouvelle
            ↪ valeur de M, càd M(t+dt)

        #nouvelle valeur de t
        t+=dt
        tt.append(t)
    return tM, tt
```

Et une procédure pour afficher la trajectoire calculée :

```
def dessinRessortMouillé(t0 , dt, tf, M0, v0):
        tM, _ = eulerPenduleRessortMouillé(t0 , dt, tf, M0, v0)
        # On ne va pas tracé x en fonction de t, ni y en fonction de t, mais plutôt la
            ↪ trajectoire, càd l'ensemble des points (x(t), y(t)) pour t   [t0,tf]

        tx = [ M[0] for M in tM]
        ty = [ M[1] for M in tM]
```

```
7
8           plt.plot(tx, ty, label="Trajectoire")
9           plt.plot( [-1.5*l0, 1.5*l0], [h,h], color="blue")
10          plt.show()
```

Deuxième partie

Exercices

Exercices : méthode d'Euler

1 Variante sur la méthode d'Euler

On considère une équation différentielle autonome d'ordre 2, où ni la dérivée première ni le temps n'interviennent : soit $F : \mathbb{R} \to \mathbb{R}$ et soit E l'équation $f'' = F \circ f$ d'inconnue $f \in \mathcal{C}^2(\mathbb{R})$.

1.1 Méthode de Verlet

1. Démontrer que pour tout $t \in \mathbb{R}$, à condition que f soit de classe $\mathcal{C}^4$, on a
$$f(t+h) \underset{h\to 0}{=} 2f(t) - f(t-h) + h^2 F\big(f(t)\big) + O\left(h^4\right).$$
Indication : Partir d'un développement limité à l'ordre 3 de $f(t+h)$ ainsi que de $f(t-h)$.
2. Soit t_0 l'instant de début de simulation. Pour initialiser toutes les variables, nous devrons calculer $f(t_0 - h)$ en connaissant $f'(t_0)$. Proposer une formule adéquate. Bien réfléchir à la précision nécessaire.
3. En déduire une variante du programme vu en cours pour calculer une approximation de f.

1.2 Application : équation du pendule

Un pendule de masse m est attaché par un fil de longueur ℓ à un point O, fixe dans un référentiel galiléen. On note $\vec{e}_z$ le vecteur unitaire colinéaire et de même direction que la gravité.

Pour tout $t \in \mathbb{R}$, on note $M(t)$ la position de la masse à l'instant t, et $\theta(t)$ une mesure de l'angle $\left(\vec{e}_z, \vec{OM}(t)\right)$.

1. Écrire l'équation différentielle vérifiée par θ.
2. Tracer le graphe de θ pour différentes valeurs de m, l, $\theta(0)$ et $\dot{\theta}(0)$. On comparera les méthodes d'Euler et de Verlet.
3. Quelles formules permettent d'obtenir les coordonnées (x, y) d'une position du pendule à partir de la valeur de θ ?
4. Écrire une procédure qui trace la trajectoire du pendule.
5. Écrire une fonction prenant en entrée l, $\theta(0)$ et $\dot{\theta}(0)$ et t_f et renvoyant la hauteur maximale atteinte par le pendule.
6. Écrire une fonction prenant en entrée l, $\theta(0)$ et $\dot{\theta}(0)$ et t_f et renvoyant le nombre d'oscillations effectuées.
Indication : Réfléchir au moyen de détecter le début et la fin d'une oscillation.
7. Modifier la fonction précédente pour obtenir également la durée de chaque oscillation. À quel point le pendule est-il asynchrone ?

2 Système d'équations différentielles

Le principal prédateur du glomorphe à pois est le lynx à collier de Mélanésie. On étudie la population de ces deux espèces en milieu clos. On note, à tout instant t, $x(t)$ et $y(t)$ la population de chaque espèce. On suppose qu'il existe des constantes $a, b, c, d \in (\mathbb{R}^{+*})^4$ telle que :

$$\begin{cases} x' = ax - bxy \\ y' = -cy + dxy \end{cases}$$

Interprétation : xy est proportionnel à la probabilité qu'une proie rencontre un prédateur. Et lorsque cela arrive, le nombre de proies diminue, et le nombre de prédateurs augmente.

1. Identifier x et y.
2. Adapter la méthode d'Euler pour calculer des valeurs approchées de x et y.
Quel semble être le comportement de $x(t)$ et $y(t)$ lorsque $t \to \infty$?
3. *(Si le cours le tableaux numpy a été fait)* On note pour tout $t \in \mathbb{R}$, $X(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$. Écrire l'équation différentielle vérifiée par X. Écrire une fonction prenant en entrée $X(t)$, a, b, c et d et renvoyant $X'(t)$. En déduire une version plus compacte de la fonction précédente.

4. **Méthode d'Euler semi-implicite :** La méthode d'Euler normale consiste à calculer les deux suites $\big(x_n\big)_{n\in\mathbb{N}}$ et $\big(y_n\big)_{n\in\mathbb{N}}$ telles que $\begin{cases} x_{n+1} = h(ax_n - bx_n y_n) \\ y_{n+1} = h(-cy_n + dx_n y_n) \end{cases}$.

   On appelle la méthode d'Euler semi-implicite celle consistant à employer la relation de récurrence suivante : $\begin{cases} x_{n+1} = h(ax_n - bx_n y_n) \\ y_{n+1} = h(-cy_n + dx_{n+1} y_n) \end{cases}$.

   (a) Aviez-vous utilisé la méthode normale ou la méthode semi-implicite ?

   (b) Comparer expérimentalement les deux.

5. **Exploitation de la modélisation :**

   (a) Calculer la valeur moyenne de x et de y au cours du temps.

   (b) On suppose à présent que les deux espèces sont des poissons. On veut étudier l'influence de la pêche : on rajoute un terme en $-\epsilon x$ dans la première équation et $-\epsilon y$ dans la seconde (on pêche indifféremment proies ou prédateurs, plus il y en a et plus on en pêche).

   Calculer les nouvelles valeurs moyennes. Commentaires ?

   (c) **Étude théorique des trajectoires :**

   i. On pose $F : \begin{array}{ccc} \mathbb{R}^2 & \to & \mathbb{R} \\ (u,v) & \mapsto & bv + du - a\ln(v) - c\ln(u) \end{array}$. Démontrer que la fonction $t \mapsto F(x(t), y(t))$ est constante.

   On peut donc dire que la quantité F est conservée au fil du temps. Ainsi, F a le même rôle que l'énergie dans un problème physique. On dit que F est une intégrale première du système.

   ii. Dans le fichier joint vous trouverez une fonction pour tracer les courbes de niveau de F, c'est-à-dire les courbes sur lesquelles F est constante.

   En traçant simultanément ces courbes de niveau et la trajectoire de (x, y), préciser vos conjectures quant à l'évolution de $\big(x(t), y(t)\big)$ quand t tend vers $+\infty$.