1 Préambule

Nous allons ici chercher à résoudre numériquement des problèmes de Cauchy d'inconnue $x \in \mathcal{C}^1(I, \mathbb{R})$ de la forme :

$$(\mathcal{S}) : \left\{ \begin{array}{cc} x' & = f(x,t) \\ x(t_0) & x_0 \end{array} \right.$$

où f est une fonction définie sur $\Omega \subset \mathbb{R}^2$.
Pour f suffisamment régulière, on admettra le théorème de Cauchy-Lipschitz qui stipule qu'il existe une unique solution maximale à ce problème de Cauchy.

Nous raisonnerons ici dans $\mathbb{R}$ mais il est tout à fait possible de raisonner sur un problème de Cauchy d'inconnue $X \in \mathbb{R}^d$ de la forme:

$$(\mathcal{S}') : \left\{ \begin{array}{cc} X' & = F(X,t) \\ X(t_0) & X_0 \end{array} \right.$$

Les méthodes détaillés ci-dessous s'applique dans les deux cas, il faut simpliment rester vigilant quant aux objets manipulés (aussi bien mathématiquement qu'informatiquement).

2 Méthode d'Euler explicite

On subdivise l'intervalle de temps $[t_0, t_0 + T]$ en $N+1$ points $t_0 < t_1 < \cdots < t_N = t_0 + T$ puis on approche la relation :

$$x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} x'(t)dt = \int_{t_n}^{t_{n+1}} F(x(t),t)dt$$

La méthode d'Euler explicite consiste à approcher cette intégrale par la méthode du rectangle gauche,autrement dit à approcher :

$$\int_{t_n}^{t_{n+1}} F(x(t),t)dt \simeq (t_{n+1} - t_n)F(x(t_n), t_n)$$

En posant $h_n = t_{n+1} - t_n$, le pas de la subdivision (que nous prendrons en général constant à h) on a :

$$\boxed{\forall 0 \leq n \leq N-1,\ x_{n+1} = x_n + h_n F(x_n, t_n)}$$

On a donc :

```
1 def euler_explicite(f, x0, t):
2     N = len(t)
3     x = [x0]
4     for n in range(0, N-1):
5         h = t[n+1]-t[n]
6         p1 = f(x[n], t[n])
7         x.append(x[n] + h * p1)
8     return x
```

Il est évidemment possible de modifier ce code lorsque l'on travaille avec un système vectoriel $(\mathcal{S})'$. Il suffit d'être vigilant lors de la manipulation de listes de listes :

```
1 def euler_explicite_vect(F, X0, t):
2     N = len(t)
3     valeurs_X = [X0]
4     X=X0
5     for n in range(0, N-1):
6         h = t[n+1]-t[n]
7         X=X+h*F(X,t[n])
8         valeurs_X.append(X)
9     return valeurs_X
```

Théorème :
La méthode d'Euler explicite est convergente d'ordre 1.

Démonstration :

- On montre que le schéma est consistant d'ordre 1. C'est-à-dire que pour un N fixé, en notant l'erreur de consistance la donnée :

$$\varepsilon_n = \frac{x(t_{n+1}) - x_{n+1}}{h} = \frac{x(t_{n+1}) - x(t_n)}{h} - F(x(t_n), t_n)$$

On montre que $\sup_{n \leq N} \|\varepsilon_n\| \leq Ch$.
On a :

$$x(t_{n+1}) = x(t_n) + h \underbrace{x'(t_n)}_{=F(x(t_n),t_n)} + \frac{h^2}{2} x''(\theta_n) \quad \text{avec} \quad \theta_n \in]t_n, t_{n+1}[$$

On a donc :

$$\varepsilon_n = \frac{h}{2} x''(\theta_n) \underset{h \to 0}{=} O(h)$$

Ainsi :

$$\sup_{n \leq N} \|\varepsilon_n\| \leq \frac{\|x''\|}{2} h$$

- On montre que le schéma est stable.
La fonction F supposée suffisamment régulière est lipschitizienne par rapport à la variable d'état (hypothèse d'existence et d'unicité d'une solution maximale du problème de Cauchy), elle est donc stable.

- On montre de cela la convergence d'ordre 1 de la méthode. C'est-à-dire qu'on montre que :

$$\sup_{n \leq N} \|x(t_n) - x_n\| \leq C' h$$

On a l'erreur de consistance :

$$\varepsilon_n = \frac{x(t_{n+1}) - x(t_n)}{h} - F(x(t_n), t_n)$$

Donc :

$$x(t_{n+1}) = x(t_n) + hF(x(t_n), t_n) + h\varepsilon_n$$

On a donc que la suite $z_n = x(t_n)$ satisfait le schéma d'Euler explicite perturbé de perturbation $\nu_n = h\varepsilon_n$.
Par ailleurs on a :

$$x_{n+1} = x_n + hF(x_n, t_n)$$

Donc par stabilité, il existe $M > 0$ tel que :

$$\sup_{n \leq N} \|x(t_n) - x_n\| \leq M \left(\|x(t_0) - x_0\| + \sum_{n=0}^{N-1} h\|\varepsilon_n\| \right)$$

Or, par consistance, on a :

$$\sup_{n \leq N} \|\varepsilon_n\| \leq Ch$$

D'où

$$\sup_{n \leq N} \|x(t_n) - x_n\| \leq M \left(\underbrace{\|x(t_0) - x_0\|}_{=0} + Ch \times \underbrace{hN}_{=T} \right)$$

Donc on a :

$$\sup_{n \leq N} \|x(t_n) - x_n\| \leq \underbrace{MCT}_{=C'} \times h$$

D'où le résultat.

3 Méthode d'Euler implicite

On subdivise l'intervalle de temps $[t_0, t_0+T]$ en $N+1$ points $t_0 < t_1 < \cdots < t_N = t_0+T$ puis on approche la relation :

$$x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} x'(t)dt = \int_{t_n}^{t_{n+1}} F(x(t),t)dt$$

La méthode d'Euler implicite consiste à approcher cette intégrale par la méthode du rectangle droit,autrement dit à approcher :

$$\int_{t_n}^{t_{n+1}} F(x(t),t)dt \simeq (t_{n+1}-t_n)F(x(t_{n+1}),t_{n+1})$$

En posant $h_n = t_{n+1} - t_n$, le pas de la subdivision (que nous prendrons en général constant à h) on a :

$$\boxed{\forall 0 \leq n \leq N-1,\ x_{n+1} = x_n + h_n F(x_{n+1}, t_{n+1})}$$

La valeur de x_{n+1} étant dans les membres de droite et de gauche, l'idée est de calculer à chaque itération les zéros de la fonction :

$$G(u) = u - x_n - h_n F(u, t_{n+1})$$

Notamment à l'aide d'une méthode de Newton (ou autre) pour le cas unidimensionnel et de Newton-Raphson pour le cas pluridimensionnel (c.f Résolution numérique d'équations).

Une fonction `newton` est également intégrée dans le module `scipy.optimize`.

On a donc :

```
1  from scipy.optimize import newton
2
3  def euler_implicite(f, X0, t):
4      N = len(t)
5      x = [x0]
6      for n in range(0, N-1):
7          h = t[n+1]-t[n]
8          s = newton(lambda u: u-x[n]-f(u, t[n+1])*h, x[n])
9          x.append(s)
10     return x
```

Il est évidemment possible de modifier ce code lorsque l'on travaille avec un système vectoriel $(\mathcal{S})'$. Il suffit d'être vigilant lors de la manipulation de listes de listes :

```
1  def euler_implicite_vect(F, X0, t):
2      N = len(t)
3      valeurs_X = [X0]
4      X=X0
5      for n in range(0, N-1):
6          h = t[n+1]-t[n]
7          s = newton(lambda U: U-X[n]-F(U, t[n+1])*h, X[n])
8          valeurs_X.append(X)
9      return valeurs_X
```

Théorème :
La méthode d'Euler implicite est convergente d'ordre 1.

Démonstration :

- On montre que le schéma est consistant d'ordre 1. C'est-à-dire que pour un N fixé, en notant l'erreur de consistance la donnée :

$$\varepsilon_n = \frac{x(t_{n+1}) - x_{n+1}}{h} = \frac{x(t_{n+1}) - x(t_n)}{h} - F(x(t_{n+1}), t_{n+1})$$

On montre que $\sup_{n\leq N}\|\varepsilon_n\| \leq Ch$.

On a :

$$x(t_n) = x(t_{n+1}) - h \underbrace{x'(t_{n+1})}_{=F(x(t_{n+1}),t_{n+1})} + \frac{h^2}{2}x''(\theta_n) \quad \text{avec} \quad \theta_n \in]t_n, t_{n+1}[$$

On a donc :

$$\varepsilon_n = \frac{h}{2}x''(\theta_n) \underset{h\to 0}{=} O(h)$$

Ainsi :

$$\sup_{n\leq N}\|\varepsilon_n\| \leq \frac{\|x''\|}{2}h$$

- On montre que le schéma est stable.
  La fonction F supposée suffisamment régulière est lipschitizienne par rapport à la variable d'état (hypothèse d'existence et d'unicité d'une solution maximale du problème de Cauchy), elle est donc stable.
- On montre de cela la convergence d'ordre 1 de la méthode. C'est-à-dire qu'on montre que :

$$\sup_{n\leq N}\|x(t_n) - x_n\| \leq C'h$$

On a l'erreur de consistance :

$$\varepsilon_n = \frac{x(t_{n+1}) - x(t_n)}{h} - F(x(t_{n+1}), t_{n+1})$$

Donc :

$$x(t_{n+1}) = x(t_n) + hF(x(t_{n+1}), t_{n+1}) + h\varepsilon_n$$

On a donc que la suite $z_n = x(t_n)$ satisfait le schéma d'Euler implicite perturbé de perturbation $\nu_n = h\varepsilon_n$.
Par ailleurs on a :

$$x_{n+1} = x_n + hF(x_{n+1}, t_{n+1})$$

Donc par stabilité, il existe $M > 0$ tel que :

$$\sup_{n\leq N}\|x(t_n) - x_n\| \leq M\left(\|x(t_0) - x_0\| + \sum_{n=0}^{N-1} h\|\varepsilon_n\|\right)$$

Or, par consistance, on a :

$$\sup_{n\leq N}\|\varepsilon_n\| \leq Ch$$

D'où

$$\sup_{n\leq N}\|x(t_n) - x_n\| \leq M\left(\underbrace{\|x(t_0) - x_0\|}_{=0} + Ch \times \underbrace{hN}_{=T}\right)$$

Donc on a :

$$\sup_{n\leq N}\|x(t_n) - x_n\| \leq \underbrace{MCT}_{=C'} \times h$$

D'où le résultat.

4 Méthode Runge-Kutta 4

En notant M_n le point de coordonnées $(t_n, x(t_n))$, le segment $[M_nM_{n+1}]$ a en général une pente plus proche de $x\left(t_n + \frac{h_n}{2}\right)$ (pente de la tangente au point milieu) que de $x'(t_n)$, pente de la tangente en M_n utilisée dans la méthode d'Euler.

D'où l'idée de remplacer la relation d'Euler :

$$x_{n+1} = x_n + h_nF(x_n, t_n)$$

Par :

$$x_{n+1} = x_n + h_nF\left(x\left(t_n + \frac{h_n}{2}\right), t_n + \frac{h_n}{2}\right)$$

Cependant, comme on ne connait pas $x\left(t_n + \frac{h_n}{2}\right)$, on en cherche une approximation notée $x_{n+\frac{1}{2}}$; le schéma d'Euler nous suggère de prendre :

$$x_{n+\frac{1}{2}} = x_n + \frac{h_n}{2} F(x_n, t_n)$$

On itérant ce processus on définit le schéma :

$$\boxed{x_{n+1} = x_n + \frac{h}{6}(k_1^n + 2k_2^n + 2k_3^n + k_4^n)}$$

où :

$$\boxed{\begin{array}{rcl} k_1^n & = & F(x_n, t_n) \\ k_2^n & = & F\left(x_n + \frac{h_n}{2}k_1^n \ , \ t_n + \frac{h_n}{2}\right) \\ k_3^n & = & F\left(x_n + \frac{h_n}{2}k_2^n \ , \ t_n + \frac{h_n}{2}\right) \\ k_4^n & = & F\left(x_n + hk_3^n \ , \ t_{n+1}\right) \end{array}}$$

On a donc :

```
1  def rk4(f, x0, t):
2      N = len(t)
3      x = [x0]
4      for n in range(0, N-1):
5          h = t[n+1]-t[n]
6          k1 = f(x[n], t[n])
7          k2 = f(x[n] + h * k1 / 2, t[n] + h / 2)
8          k3 = f(x[n] + h * k2 / 2, t[n] + h / 2)
9          k4 = f(x[n] + h * k3, t[n+1])
10         x.append(x[n] + h * (k1+2*k2+2*k3+k4) / 6)
11     return x
```

Théorème :
La méthode de Runge-Kutta est convergente d'ordre 4.

5 Le module `scipy.integrate`

Le module `scipy.integrate` contient une fonction nommée `odeint` qui permet le calcul approché d'une solution d'une EDO. Cette fonction est dédiée à la résolution des systèmes numériques, mais s'applique aussi aux équations scalaires (qui somme toute ne sont que des systèmes différentiels d'ordre 1) et aux équations différentielles d'ordre supérieur.
Par exemple, pour résoudre l'équation :

$$\frac{ds}{dt} = \frac{e(t) - s}{\tau} \quad \text{avec} \quad s(0) = \frac{q_0}{C}$$

```
1  def f(s, t):
2      return (e(t)-s) / tau
3
4  t = np.linspace(0, 0.01, 200)
5  s = odeint(f, q0/C, t)
6  plot (t, s)
```

Et pour résoudre l'équation vectorielle suivante :

$$\begin{cases} \frac{di}{dt} = \frac{-Ri - s - e(t)}{L} \\ \frac{ds}{dt} = \frac{i}{C} \end{cases} \quad \text{avec} \quad i(0) = 0 \ \text{ et } \ s(0) = 0$$

```
def f(x, t):
    [i, s] = x
    return [(-R*i-s-e(t))/L, i/C]

t = np.linspace(0, .02, 200)
x = odeint(f, [0, 0], t)
plot (t, x[:, 1])
```