Méthodes Numériques pour les Problèmes de Valeurs Initiales d'EDO, Convergence et Stabilité

Introduction

Dans ce chapitre, nous allons aborder la résolution numérique des équations différentielles ordinaires (EDO), en nous concentrant particulièrement sur les problèmes de valeurs initiales. Nous introduirons la terminologie associée, les méthodes numériques de base, et analyserons deux propriétés cruciales de ces méthodes : la convergence et la stabilité.

Problèmes de Valeurs Initiales d'EDO

Nous nous intéressons aux problèmes de la forme :

$$y'(t) = f(t, y)$$

avec une condition initiale :

$$y(t_0) = y_0$$

Ici, $y(t)$ est une fonction de dimension n. Nous pouvons écrire ce système sous forme vectorielle :

$$\begin{pmatrix} y_1'(t) \\ y_2'(t) \\ \vdots \\ y_n'(t) \end{pmatrix} = \begin{pmatrix} f_1(t, y_1, \ldots, y_n) \\ f_2(t, y_1, \ldots, y_n) \\ \vdots \\ f_n(t, y_1, \ldots, y_n) \end{pmatrix}$$

où $y_1, \ldots, y_n$ sont les composantes du vecteur $y(t)$, et $f_1, \ldots, f_n$ sont les composantes de la fonction vectorielle f. Ceci constitue un système de n EDO couplées.

Le problème est dit un problème de valeurs initiales car la donnée initiale $y(t_0) = y_0$ est spécifiée à un instant précis t_0.

L'ordre d'une EDO est donné par l'ordre le plus élevé de la dérivée apparaissant dans l'équation. L'équation $y'(t) = f(t, y)$ est du premier ordre car la dérivée la plus élevée est y'.

D'un point de vue numérique, nous pouvons nous concentrer entièrement sur les équations du premier ordre. Les EDO d'ordre supérieur peuvent être converties en systèmes d'EDO du premier ordre en introduisant des variables supplémentaires.

Example 0.1. Considérons la deuxième loi de Newton pour la position $y(t)$:

$$y''(t) = \frac{f(t, y, y')}{m}$$

avec les conditions initiales $y(0) = y_0$ et $y'(0) = v_0$. Pour convertir ceci en un système du premier ordre, introduisons une nouvelle variable $v = y'$. Alors $v' = y''$. Le système devient :

$$\begin{aligned} v'(t) &= \frac{f(t, y, v)}{m} \\ y'(t) &= v \end{aligned}$$

Les conditions initiales sont $y(0) = y_0$ et $v(0) = v_0$. Nous avons maintenant un système de deux EDO du premier ordre pour les variables y et v.

Exemple : Le Modèle de Lotka-Volterra

Le modèle de Lotka-Volterra est un système non linéaire à deux composantes qui peut expliquer les interactions prédateur-proie. Soit y_1 la population d'une espèce proie (par exemple, lapins) et y_2 la population

d'une espèce prédatrice (par exemple, renards). Le système d'EDO est donné par :

$$y_1'(t) = \alpha_1 y_1 - \beta_1 y_1 y_2$$
$$y_2'(t) = \beta_2 y_1 y_2 - \alpha_2 y_2$$

où $\alpha_1, \beta_1, \alpha_2, \beta_2$ sont des constantes positives. Le terme $\alpha_1 y_1$ représente la croissance exponentielle des proies en l'absence de prédateurs. Le terme $-\beta_1 y_1 y_2$ représente la diminution de la population de proies due à la prédation. Le terme $\beta_2 y_1 y_2$ représente la croissance de la population de prédateurs due à la consommation de proies. Le terme $-\alpha_2 y_2$ représente la décroissance exponentielle des prédateurs en l'absence de proies.

Voici un exemple de code Python utilisant la fonction 'odeint' de 'scipy.integrate' pour résoudre ce système :

Listing 1: Code Python pour le modèle Lotka-Volterra

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Constants in model
alpha1=1.0
beta1=0.1
alpha2=0.4
beta2=0.02

# Function that evaluates the RHS of the ODE.
# It has two components, representing the changes in prey and predator
   ↪ populations.
def f(y,t):
    return np.array([alpha1*y[0]-beta1*y[0]*y[1],
       ↪ beta2*y[0]*y[1]-alpha2*y[1]])

# Specify the range of time values where the ODE should be solved at
t=np.linspace(0,70,500)

# Initial conditions; set to the initial populations of prey and predators
yinit=np.array([10,1])

# Solve ODE using the "odeint" library in SciPy
y=odeint(f,yinit,t)

# Plot the solutions
plt.figure()
plt.plot(t,y[:,0], label="Proies")
plt.plot(t,y[:,1], label="Prédateurs")
plt.legend(loc="upper␣right")
plt.xlabel("Temps")
plt.ylabel("Population")
plt.savefig("lotka_volterra_plot.png")
```

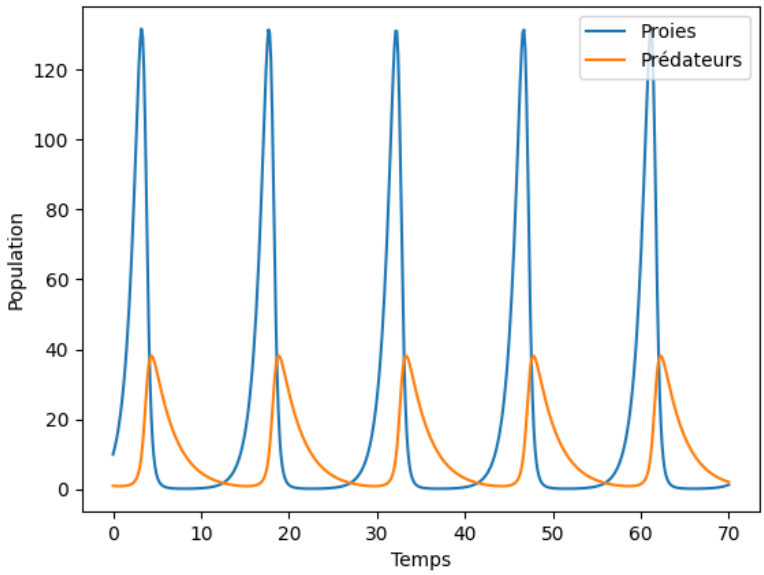

Figure 1: Solution du modèle de Lotka-Volterra montrant le comportement cyclique des populations de proies et de prédateurs.

L'exécution de ce code montre typiquement un comportement cyclique : lorsque la population de proies est élevée, les prédateurs ont de la nourriture et leur population augmente ; cela entraîne une diminution des proies ; la diminution des proies réduit la nourriture pour les prédateurs, entraînant une diminution de leur population ; cela permet aux proies de se rétablir, et le cycle se répète.

Méthodes d'Intégration à Un Pas

Pour résoudre numériquement l'EDO $y'(t) = f(t, y)$, nous cherchons à approximer la solution $y(t)$ à des points temporels discrets $t_k = t_0 + kh$, où h est le pas de temps et $k = 0, 1, 2, \ldots$. Nous notons y_k notre approximation de la solution exacte $y(t_k)$.

Une approche simple consiste à intégrer l'EDO sur l'intervalle $[t_k, t_{k+1}]$:

$$\int_{t_k}^{t_{k+1}} y'(s)ds = \int_{t_k}^{t_{k+1}} f(s, y(s))ds$$

Par le Théorème Fondamental de l'Analyse, l'intégrale du côté gauche est $y(t_{k+1}) - y(t_k)$, d'où :

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(s, y(s))ds$$

Les méthodes numériques consistent à approximer l'intégrale du côté droit. Ces méthodes peuvent souvent être exprimées sous la forme générale d'une méthode à un pas :

$$y_{k+1} = y_k + h\Phi(t_k, y_k, t_{k+1}, y_{k+1}; h)$$

où Φ est une fonction appelée incrément ou flux. Ces méthodes permettent de calculer y_{k+1} en fonction de y_k (et potentiellement de y_{k+1} lui-même).

Méthode d'Euler Explicite (ou Forward Euler)

En utilisant une quadrature de Newton-Cotes d'ordre 0 (point unique) au début de l'intervalle $[t_k, t_{k+1}]$, c'est-à-dire en approximant $f(s, y(s))$ par $f(t_k, y(t_k))$, l'intégrale devient :

$$\int_{t_k}^{t_{k+1}} f(s, y(s))ds \approx (t_{k+1} - t_k)f(t_k, y_k) = hf(t_k, y_k)$$

Ceci mène à la méthode d'Euler explicite :

$$y_{k+1} = y_k + hf(t_k, y_k)$$

Cette méthode est *explicite* car y_{k+1} est directement calculé en termes de y_k et t_k.

Méthode d'Euler Implicite (ou Backward Euler)

En utilisant une quadrature de Newton-Cotes d'ordre 0 à la fin de l'intervalle $[t_k, t_{k+1}]$, c'est-à-dire en approximant $f(s, y(s))$ par $f(t_{k+1}, y(t_{k+1}))$, l'intégrale devient :

$$\int_{t_k}^{t_{k+1}} f(s, y(s))ds \approx (t_{k+1} - t_k)f(t_{k+1}, y_{k+1}) = hf(t_{k+1}, y_{k+1})$$

Ceci mène à la méthode d'Euler implicite :

$$y_{k+1} = y_k + hf(t_{k+1}, y_{k+1})$$

Cette méthode est *implicite* car y_{k+1} apparaît des deux côtés de l'équation et nécessite généralement la résolution d'une équation (potentiellement non linéaire) pour trouver y_{k+1}.

Méthode du Trapèze

En utilisant une quadrature de Newton-Cotes d'ordre 1 (règle du trapèze) sur l'intervalle $[t_k, t_{k+1}]$, l'intégrale est approximée par :

$$\int_{t_k}^{t_{k+1}} f(s, y(s))ds \approx \frac{t_{k+1} - t_k}{2}[f(t_k, y_k) + f(t_{k+1}, y_{k+1})] = \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, y_{k+1})]$$

Ceci mène à la méthode du trapèze :

$$y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, y_{k+1})]$$

Cette méthode est également *implicite*.

Analyse de la Convergence

Nous voulons savoir si la solution numérique y_k converge vers la solution exacte $y(t_k)$ lorsque le pas de temps h tend vers zéro (et donc $k \to \infty$ pour atteindre un temps fixe t_k). La convergence est cruciale car elle garantit que nous pouvons atteindre une précision arbitraire en réduisant h.

L'analyse de convergence est subtile car les erreurs s'accumulent au fil du temps.

Definition 0.2 (Erreur Globale)**.** L'erreur globale au pas k est définie comme la différence entre la solution exacte et la solution numérique au temps t_k:

$$E_k = y(t_k) - y_k$$

Definition 0.3 (Erreur de Troncature Locale)**.** L'erreur de troncature locale au pas k, notée hT_k, est l'erreur introduite en un seul pas en appliquant la méthode numérique à la solution exacte. T_k est cette erreur divisée par h. Pour une méthode à un pas explicite, elle est donnée par :

$$T_k = \frac{y(t_{k+1}) - y(t_k)}{h} - \Phi(t_k, y(t_k))$$

hT_k représente l'erreur commise en passant de $y(t_k)$ à l'approximation $y(t_{k+1})$ en utilisant la méthode numérique, en partant de la solution exacte au temps t_k.

Nous pouvons relier l'erreur globale d'un pas au suivant. Pour la méthode d'Euler explicite : $y_{k+1} = y_k + hf(t_k, y_k)$ De la définition de l'erreur de troncature, nous avons : $y(t_{k+1}) = y(t_k) + hf(t_k, y(t_k)) + hT_k$ En soustrayant la première équation de la seconde : $y(t_{k+1}) - y_{k+1} = (y(t_k) - y_k) + h(f(t_k, y(t_k)) - f(t_k, y_k)) + hT_k$ $E_{k+1} = E_k + h(f(t_k, y(t_k)) - f(t_k, y_k)) + hT_k$ Ceci montre comment l'erreur globale E_{k+1} dépend de l'erreur globale précédente E_k et de l'erreur de troncature locale hT_k. Le terme $h(f(t_k, y(t_k)) - f(t_k, y_k))$ représente l'amplification de l'erreur précédente par la fonction f.

Definition 0.4 (Condition de Lipschitz)**.** Une fonction $f(t, y)$ satisfait une condition de Lipschitz par rapport à y sur un domaine si il existe une constante $L_f > 0$ telle que pour tout t dans un intervalle donné et pour tout u, v dans le domaine de y:

$$\|f(t, u) - f(t, v)\| \leq L_f \|u - v\|$$

La constante L_f est appelée constante de Lipschitz. Géométriquement, cela borne la pente de la fonction f par rapport à y. Si f est continûment différentiable par rapport à y, alors L_f peut être choisi comme le maximum de $\|\partial f/\partial y\|$.

Example 0.5.

- $g(x) = |x|$. $|g(x) - g(y)| = ||x| - |y|| \leq |x - y|$ (par l'inégalité triangulaire inverse). Donc $L_f = 1$.
- $g(x) = \sqrt{x}$ sur $[0, 1]$. $|g(x) - g(0)|/|x - 0| = \sqrt{x}/x = 1/\sqrt{x}$. Lorsque $x \to 0^+$, ceci tend vers l'infini. La condition de Lipschitz n'est pas satisfaite sur un intervalle incluant 0.

Theorem 0.6 (Convergence de la Méthode d'Euler Explicite)**.** Supposons que $f(t, y)$ satisfait la condition de Lipschitz par rapport à y avec la constante L_f. Alors l'erreur globale de la méthode d'Euler explicite au pas k est bornée par :

$$\|E_k\| \leq e^{L_f(t_k - t_0)} \frac{1}{L_f} \max_{0 \leq j < k} \|hT_j\|$$

ou de manière équivalente (en utilisant la définition de T_j):

$$\|E_k\| \leq e^{L_f(t_k - t_0)} \frac{h}{L_f} \max_{0 \leq j < k} \|T_j\|$$

Preuve (Esquisse de Preuve)**.** Nous avons l'inégalité pour l'erreur globale : $E_{k+1} = E_k + h(f(t_k, y(t_k)) - f(t_k, y_k)) + hT_k$ En prenant la norme et en utilisant l'inégalité triangulaire et la condition de Lipschitz ($y(t_k) - y_k = E_k$): $\|E_{k+1}\| \leq \|E_k\| + h\|f(t_k, y_k + E_k) - f(t_k, y_k)\| + \|hT_k\|$ $\|E_{k+1}\| \leq \|E_k\| + hL_f\|E_k\| + h\|T_k\|$ $\|E_{k+1}\| \leq (1 + hL_f)\|E_k\| + h \max_{0 \leq j \leq k} \|T_j\|$ En appliquant cette relation récursivement à partir de $E_0 = 0$: $\|E_1\| \leq (1 + hL_f)\|E_0\| + h \max \|T_j\| = h \max \|T_j\|$ $\|E_2\| \leq (1 + hL_f)\|E_1\| + h \max \|T_j\| \leq (1 + hL_f)h \max \|T_j\| + h \max \|T_j\| = h \max \|T_j\|(1 + 1 + hL_f)$ Ceci ne semble pas exactement la dérivation montrée dans la vidéo. Reprenons à partir de $\|E_{k+1}\| \leq (1 + hL_f)\|E_k\| + h \max_{0 \leq j < k+1} \|T_j\|$. Soit $C = h \max_{0 \leq j < k+1} \|T_j\|$. $\|E_{k+1}\| \leq (1 + hL_f)\|E_k\| + C$. Pour $k = 0$: $\|E_1\| \leq (1 + hL_f)\|E_0\| + C = C$ (car $E_0 = 0$). Pour $k = 1$: $\|E_2\| \leq (1 + hL_f)\|E_1\| + C \leq (1 + hL_f)C + C = C(1 + (1 + hL_f))$. Pour $k = 2$: $\|E_3\| \leq (1 + hL_f)\|E_2\| + C \leq (1 + hL_f)C(1 + (1 + hL_f)) + C = C(1 + (1 + hL_f) + (1 + hL_f)^2)$.

En général : $\|E_k\| \leq C \sum_{j=0}^{k-1}(1+hL_f)^j$. Cette somme est une série géométrique de raison $r = 1+hL_f$. $\sum_{j=0}^{k-1} r^j = \frac{r^k-1}{r-1} = \frac{(1+hL_f)^k-1}{(1+hL_f)-1} = \frac{(1+hL_f)^k-1}{hL_f}$. Donc, $\|E_k\| \leq h \max_{0\leq j<k} \|T_j\| \frac{(1+hL_f)^k-1}{hL_f} = \frac{1}{L_f} \max_{0\leq j<k} \|hT_j\|((1+hL_f)^k - 1)$. En utilisant l'inégalité $1+x \leq e^x$, nous avons $(1+hL_f)^k \leq (e^{hL_f})^k = e^{khL_f}$. De plus, $t_k = t_0 + kh$, donc $kh = t_k - t_0$. $(1+hL_f)^k - 1 \leq e^{L_f(t_k-t_0)} - 1 \leq e^{L_f(t_k-t_0)}$ (pour $t_k \geq t_0$). Donc, $\|E_k\| \leq \frac{e^{L_f(t_k-t_0)}}{L_f} \max_{0\leq j<k} \|hT_j\| = \frac{e^{L_f(t_k-t_0)}h}{L_f} \max_{0\leq j<k} \|T_j\|$. Ceci établit le théorème. □

Definition 0.7 (Ordre de Précision)**.** Une méthode numérique a un ordre de précision p si son erreur de troncature locale hT_k est d'ordre $O(h^{p+1})$, ce qui signifie que $\|T_k\| = O(h^p)$ pour des pas de temps h petits. Le théorème de convergence implique alors que l'erreur globale $\|E_k\|$ est d'ordre $O(h^p)$.

Ordre de Précision des Méthodes

- **Méthode d'Euler Explicite :** $T_k = \frac{y(t_{k+1})-y(t_k)}{h} - f(t_k, y(t_k))$. Utilisant $y'(t_k) = f(t_k, y(t_k))$ et un développement de Taylor de $y(t_{k+1})$ autour de t_k: $y(t_{k+1}) = y(t_k) + hy'(t_k) + \frac{h^2}{2}y''(t_k) + O(h^3)$ $T_k = \frac{(y(t_k)+hy'(t_k)+h^2/2y''(t_k)+O(h^3))-y(t_k)}{h} - y'(t_k)$ $T_k = \frac{hy'(t_k)+h^2/2y''(t_k)}{h} - y'(t_k) + O(h^2)$ $T_k = y'(t_k) + \frac{h}{2}y''(t_k) - y'(t_k) + O(h^2) = \frac{h}{2}y''(t_k) + O(h^2)$. Donc $T_k = O(h)$, et l'erreur globale est $O(h)$. La méthode d'Euler explicite est d'ordre 1.
- **Méthode d'Euler Implicite :** Par une analyse similaire utilisant un développement de Taylor autour de t_{k+1}, on trouve que $T_k = -\frac{h}{2}y''(t_{k+1}) + O(h^2)$. Donc $T_k = O(h)$, et l'erreur globale est $O(h)$. La méthode d'Euler implicite est également d'ordre 1.
- **Méthode du Trapèze :** L'erreur de troncature pour la méthode du trapèze peut être liée à l'erreur de la règle du trapèze pour l'intégration. L'erreur de la règle du trapèze pour $\int_a^b g(x)dx$ est $O((b-a)^3)$. Ici, l'intervalle est $[t_k, t_{k+1}]$, de longueur h. L'erreur de quadrature est donc $O(h^3)$. Comme la formule de la méthode du trapèze a un facteur $1/h$ devant l'intégrale approximée, l'erreur de troncature T_k est $O(h^2)$. L'erreur globale est donc $O(h^2)$. La méthode du trapèze est d'ordre 2.

Example 0.8. Comparaison de l'erreur globale pour $y' = y, y(0) = 1$ (solution exacte $y(t) = e^t$) à $t = 1$.

h	Erreur Globale (Euler Explicite)	Erreur Globale (Méthode du Trapèze)
0.1	0.0517	0.00177
0.05	0.0259	0.00044
0.025	0.0130	0.00011
0.0125	0.0065	0.000027

Lorsque h est divisé par 2 (passant de 0.1 à 0.05), l'erreur d'Euler est approximativement divisée par 2 (0.0517 / 2 ≈ 0.0259), ce qui est cohérent avec un ordre 1. Pour la méthode du trapèze, l'erreur est approximativement divisée par 4 (0.00177 / 4 ≈ 0.00044), ce qui est cohérent avec un ordre 2 ($2^2 = 4$).

Analyse de la Stabilité

En plus de la convergence (précision pour $h \to 0$), il est crucial d'analyser la stabilité pour des valeurs finies de h. Idéalement, nous voulons que notre méthode numérique reproduise les propriétés de stabilité de l'EDO sous-jacente.

Stabilité d'une EDO

Definition 0.9 (Stabilité d'une EDO)**.** L'EDO $y'(t) = f(t, y)$ est stable si pour tout $\epsilon > 0$, il existe $\delta > 0$

tel que si $\|y(t_0) - \hat{y}(t_0)\| < \delta$, alors $\|y(t) - \hat{y}(t)\| < \epsilon$ pour tout $t \geq t_0$, où $y(t)$ et $\hat{y}(t)$ sont deux solutions de l'EDO avec des conditions initiales $y(t_0)$ et $\hat{y}(t_0)$.

Une petite perturbation de la condition initiale entraîne seulement une petite perturbation de la solution à tout instant ultérieur.

Definition 0.10 (Stabilité Asymptotique d'une EDO)**.** L'EDO est asymptotiquement stable si elle est stable et si de plus $\|y(t) - \hat{y}(t)\| \to 0$ lorsque $t \to \infty$.

Dans ce cas, les petites perturbations décroissent au fil du temps.

Le Problème Test : $y' = \lambda y$

Pour analyser la stabilité numérique, on utilise classiquement le problème test scalaire $y' = \lambda y$, où λ est un paramètre complexe. La solution exacte est $y(t) = y(0)e^{\lambda t}$. La différence entre deux solutions est $y(t) - \hat{y}(t) = (y(0) - \hat{y}(0))e^{\lambda t}$. La stabilité dépend du comportement de $e^{\lambda t}$ lorsque $t \to \infty$. Soit $\lambda = a + ib$. $|e^{\lambda t}| = |e^{(a+ib)t}| = |e^{at}e^{ibt}| = e^{at}|e^{ibt}| = e^{at}$.

- Si $\text{Re}(\lambda) = a < 0$, $e^{at} \to 0$ quand $t \to \infty$. La différence entre les solutions tend vers 0. L'EDO est asymptotiquement stable.
- Si $\text{Re}(\lambda) = a = 0$, $e^{at} = e^0 = 1$. La différence reste bornée (égale à la différence initiale). L'EDO est stable mais pas asymptotiquement stable.
- Si $\text{Re}(\lambda) = a > 0$, $e^{at} \to \infty$ quand $t \to \infty$. La différence entre les solutions croît de manière exponentielle. L'EDO est instable.

Pour un système linéaire $y' = Ay$, où A est une matrice, la stabilité est déterminée par les valeurs propres λ_i de A. Le système est stable si $\text{Re}(\lambda_i) \leq 0$ pour toutes les valeurs propres, et asymptotiquement stable si $\text{Re}(\lambda_i) < 0$ pour toutes les valeurs propres.

Stabilité Numérique

Definition 0.11 (Stabilité Numérique)**.** Une approximation numérique est stable si pour tout $\epsilon > 0$, il existe $\delta > 0$ tel que si $\|y_0 - \hat{y}_0\| < \delta$, alors $\|y_k(y_0) - y_k(\hat{y}_0)\| < \epsilon$ pour tout $k \geq 0$, où $y_k(y_0)$ et $y_k(\hat{y}_0)$ sont les solutions numériques pour les conditions initiales y_0 et $\hat{y}_0$.

Nous souhaitons que notre méthode numérique soit stable lorsque l'EDO elle-même est stable, c'est-à-dire pour $\text{Re}(\lambda) \leq 0$ dans le cas du problème test.

Stabilité de la Méthode d'Euler Explicite

Appliquons la méthode d'Euler explicite au problème test $y' = \lambda y$: $y_{k+1} = y_k + hf(t_k, y_k) = y_k + h(\lambda y_k) = (1+h\lambda)y_k$ En partant de y_0, on obtient $y_k = (1+h\lambda)^k y_0$. Le facteur d'amplification numérique est $G = 1+h\lambda$. Pour que y_k ne croisse pas indéfiniment lorsque l'EDO est stable (i.e., pour $\text{Re}(\lambda) \leq 0$), il faut que $|G| \leq 1$. Soit $z = h\lambda$. La condition de stabilité est $|1 + z| \leq 1$. Dans le plan complexe z, ceci correspond à un disque de rayon 1 centré en $(-1, 0)$. C'est la région de stabilité de la méthode d'Euler explicite. Pour que la méthode soit stable lorsque l'EDO est stable (c'est-à-dire pour tout z tel que $\text{Re}(z/h) \leq 0$, soit $\text{Re}(z) \leq 0$), la région de stabilité numérique (le disque) doit contenir le demi-plan gauche ($\text{Re}(z) \leq 0$). Ce n'est pas le cas. La méthode d'Euler explicite est donc *conditionnellement stable*. Pour une valeur de λ avec $\text{Re}(\lambda) < 0$, la stabilité numérique n'est garantie que si h est suffisamment petit pour que $h\lambda$ tombe dans le disque de stabilité. Si λ est réel et négatif ($\lambda < 0$), alors $z = h\lambda < 0$. La condition $|1 + z| \leq 1$ pour $z < 0$ devient $|1 + h\lambda| \leq 1$. Si $1 + h\lambda \geq 0$, $1 + h\lambda \leq 1 \implies h\lambda \leq 0$, toujours vrai pour $\lambda < 0, h > 0$. Si $1 + h\lambda < 0$, $-(1 + h\lambda) \leq 1 \implies -1 - h\lambda \leq 1 \implies -2 \leq h\lambda$. Donc, $-2 \leq h\lambda \leq 0$. Pour $\lambda < 0$, ceci implique $h \leq -2/\lambda = 2/|\lambda|$. Le pas de temps h est restreint.

Voici un programme Python pour illustrer la stabilité/instabilité de la méthode d'Euler explicite :

Listing 2: Code Python pour Euler Explicite stable

```
import matplotlib.pyplot as plt
import numpy as np
from math import exp

# Initial variables and constants
y=1.0
t=0.0
h=0.1

# Choose the constant in the ODE, dy/dt=lambda*y.
# We need -2 <= h*lambda <= 0 for stability for real lambda.
lam=-5.0 # h*lambda = -0.5 (stable)

# Apply Forward Euler step until t>=1
t_end = 1.0
t_values = [t]
y_values = [y]
y_exact_values = [exp(lam*t)]

while t < t_end:
    # Analytical solution
    yexact=exp(lam*t)

    # Euler step
    y = y+h*(lam*y)

    # Update time
    t=t+h
    t_values.append(t)
    y_values.append(y)
    y_exact_values.append(exp(lam*t))

# Plot the solutions
plt.figure()
plt.plot(t_values, y_exact_values, label='Exact')
plt.plot(t_values, y_values, 'o-', label='Numerical')
plt.legend(loc="upper␣right")
plt.xlabel("t")
plt.ylabel("y")
plt.savefig("euler_stab_lambda_neg5.png")
```

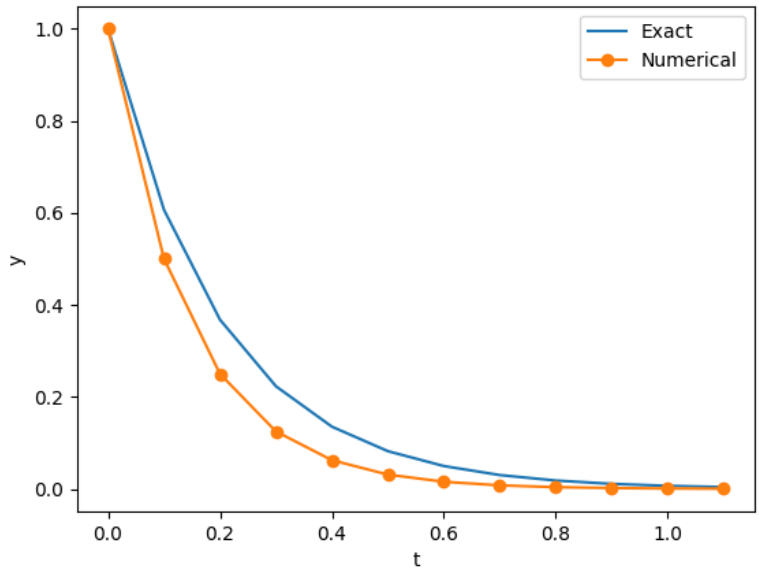

Figure 2: Solution pour Euler Explicite avec $\lambda = -5$ ($h = 0.1$, $h\lambda = -0.5$). Stable.

Listing 3: Code Python pour Euler Explicite oscillatoire stable

```
import matplotlib.pyplot as plt
import numpy as np
from math import exp

# Initial variables and constants
y=1.0
t=0.0
h=0.1

# Choose the constant in the ODE, dy/dt=lambda*y.
# We need -2 <= h*lambda <= 0 for stability for real lambda.
lam=-12.5 # h*lambda = -1.25 (stable, but oscillatory)

# Apply Forward Euler step until t>=1
t_end = 1.0
t_values = [t]
y_values = [y]
y_exact_values = [exp(lam*t)]

while t < t_end:
    # Analytical solution
    yexact=exp(lam*t)

    # Euler step
    y = y+h*(lam*y)

    # Update time
    t=t+h
    t_values.append(t)
```

```
    y_values.append(y)
    y_exact_values.append(exp(lam*t))

# Plot the solutions
plt.figure()
plt.plot(t_values, y_exact_values, label='Exact')
plt.plot(t_values, y_values, 'o-', label='Numerical')
plt.legend(loc="upper right")
plt.xlabel("t")
plt.ylabel("y")
plt.savefig("euler_stab_lambda_neg12_5.png")
```

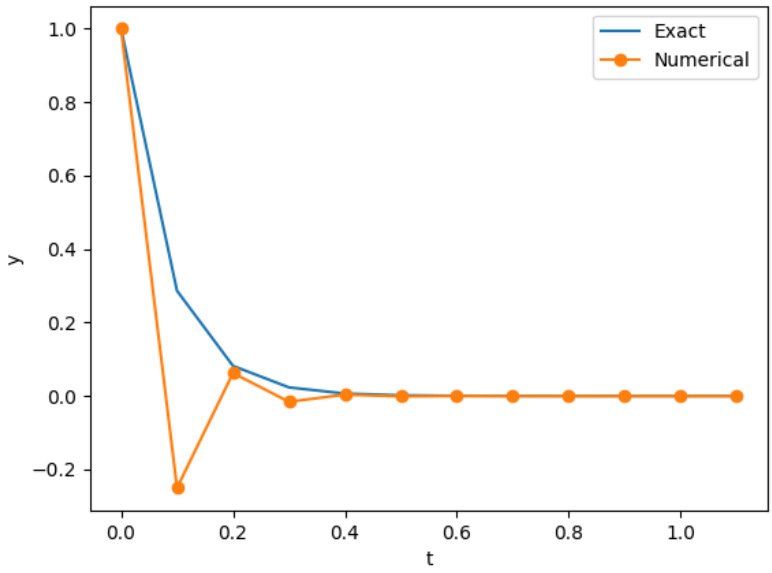

Figure 3: Solution pour Euler Explicite avec $\lambda = -12.5$ ($h = 0.1$, $h\lambda = -1.25$). Stable mais avec oscillations.

Listing 4: Code Python pour Euler Explicite instable

```
import matplotlib.pyplot as plt
import numpy as np
from math import exp

# Initial variables and constants
y=1.0
t=0.0
h=0.1

# Choose the constant in the ODE, dy/dt=lambda*y.
# We need -2 <= h*lambda <= 0 for stability for real lambda.
lam=-21.0 # h*lambda = -2.1 (unstable)

# Apply Forward Euler step until t>=1
t_end = 1.0
t_values = [t]
```

```
y_values = [y]
y_exact_values = [exp(lam*t)]

while t < t_end:
    # Analytical solution
    yexact=exp(lam*t)

    # Euler step
    y = y+h*(lam*y)

    # Update time
    t=t+h
    t_values.append(t)
    y_values.append(y)
    y_exact_values.append(exp(lam*t))

# Plot the solutions
plt.figure()
plt.plot(t_values, y_exact_values, label='Exact')
plt.plot(t_values, y_values, 'o-', label='Numerical')
plt.legend(loc="upper right")
plt.xlabel("t")
plt.ylabel("y")
plt.savefig("euler_stab_lambda_neg21.png")
```

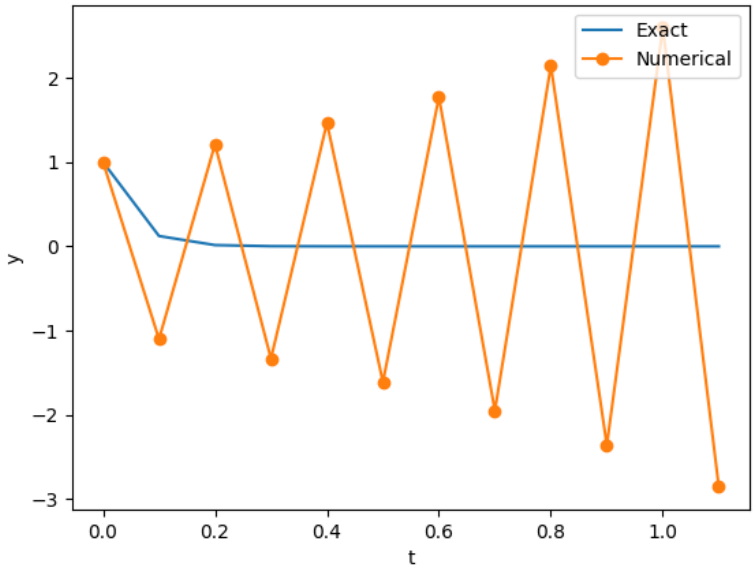

Figure 4: Solution pour Euler Explicite avec $\lambda = -21$ ($h = 0.1$, $h\lambda = -2.1$). Instable.

Ces exemples montrent que lorsque $h\lambda$ est dans la région de stabilité (par exemple, -0.5 et -1.25), la solution numérique reste stable et converge vers la solution exacte (décroissance vers 0). Lorsque $h\lambda$ est en dehors de la région de stabilité (par exemple, -2.1), la solution numérique devient instable et diverge. L'oscillation observée pour $h\lambda = -1.25$ est due au fait que le facteur d'amplification $1 + h\lambda$ est négatif, ce qui fait alterner le signe de y_k à chaque pas, tout en décroissant en magnitude.

Stabilité de la Méthode d'Euler Implicite

Appliquons la méthode d'Euler implicite au problème test $y' = \lambda y$: $y_{k+1} = y_k + hf(t_{k+1}, y_{k+1}) = y_k + h(\lambda y_{k+1})$ $y_{k+1}(1 - h\lambda) = y_k$ $y_{k+1} = \frac{1}{1-h\lambda} y_k$ Le facteur d'amplification numérique est $G = \frac{1}{1-h\lambda}$. Pour la stabilité, il faut $|G| \leq 1$, c'est-à-dire $|\frac{1}{1-h\lambda}| \leq 1 \iff |1-h\lambda| \geq 1$. Soit $z = h\lambda$. La condition est $|1-z| \geq 1$. Dans le plan complexe z, $|1-z|$ représente la distance entre 1 et z. La condition $|1-z| \geq 1$ signifie que z doit être à l'extérieur ou sur le cercle de rayon 1 centré en $(1, 0)$. C'est la région de stabilité de la méthode d'Euler implicite. Cette région de stabilité ($|1-z| \geq 1$) contient tout le demi-plan gauche ($\text{Re}(z) \leq 0$). La méthode d'Euler implicite est donc *inconditionnellement stable* pour les EDO stables ($\text{Re}(\lambda) \leq 0$). Cela signifie que pour toute EDO stable, nous n'avons pas de restriction sur le pas de temps h pour garantir la stabilité numérique.

Explicite vs Implicite pour la Stabilité

En général, les méthodes implicites ont souvent des régions de stabilité plus grandes (voire incluant tout le demi-plan gauche) que les méthodes explicites. Cela permet d'utiliser de plus grands pas de temps h pour les problèmes stables, notamment les problèmes "raides" (stiff) où $|\lambda|$ est grand pour certaines composantes ou modes. Cependant, les méthodes implicites sont plus coûteuses par pas car elles nécessitent la résolution d'un système d'équations (linéaire ou non) pour y_{k+1}. Le choix de la méthode dépend souvent d'un compromis entre le coût par pas et la taille du pas permis par la stabilité.

Conclusion

Ce chapitre a introduit les concepts fondamentaux des problèmes de valeurs initiales d'EDO et a présenté des méthodes numériques de base comme les méthodes d'Euler explicite et implicite et la méthode du trapèze. Nous avons défini et analysé la convergence, montrant comment l'erreur globale est liée à l'erreur de troncature locale et à la condition de Lipschitz. Enfin, nous avons examiné la stabilité numérique, en utilisant le problème test $y' = \lambda y$ pour comprendre les régions de stabilité et la différence cruciale entre les méthodes conditionnellement stables (comme Euler explicite) et inconditionnellement stables (comme Euler implicite) en fonction du pas de temps choisi.