EDP linéaires

Aide-mémoire.

Gloria Faccanoni

http://faccanoni.univ-tln.fr/enseignements.html

Année 2015 – 2016

Table des matières

Dernière mise-à-jour
Mardi 22 mars 2016

Gloria FACCANONI

IMATH Bâtiment M-117
Université de Toulon
Avenue de l'université
83957 LA GARDE - FRANCE

☎ 0033 (0)4 83 16 66 72

✉ gloria.faccanoni@univ-tln.fr
🛈 http://faccanoni.univ-tln.fr

Chapitre 1.

Méthode des caractéristiques

On s'intéresse au calcul de la solution exacte d'une équation différentielle aux dérivées partielles linéaire. On cherche

$$\begin{aligned} u\colon \mathbb{R}\times\mathbb{R}^+ &\to \mathbb{R} \\ (x,t) &\mapsto u(x,t) \end{aligned}$$

solution faible du problème

$$\begin{cases} \partial_t u(t,x) + a(t,x)\partial_x u(t,x) + b(t,x)u(t,x) = f(t,x), & x\in\mathbb{R}, t>0, \\ u(x,0) = g(x), & x\in\mathbb{R}. \end{cases}$$

Soit $(\hat{t},\hat{x})\in]0;+\infty[\times\mathbb{R}$ donné. On appelle **courbe caractéristique passant par** $(\hat{t},\hat{x})$ la solution (si elle existe) du problème de CAUCHY

$$\begin{cases} x'(t) = a(t,x(t)) \quad t>0, \\ x(\hat{t}) = \hat{x}. \end{cases}$$

En particulier, si $a(t,x) = A$ constante, alors $x(t) = \hat{x} + A(t-\hat{t})$.

Notons $v(t) \equiv u(t,x(t))$ la restriction de u à la courbe caractéristique. Alors $u(\hat{t},\hat{x}) = v(\hat{t})$ et l'on a

$$v'(t) = \partial_t u(t,x(t)) + x'(t)\partial_x u(t,x(t)) = \partial_t u(t,x(t)) + a(t,x(t))\partial_x u(t,x(t)).$$

Par conséquent,

- ▷ si $b(t,x) = f(t,x) = 0$ pour tout $(t,x)\in[0;+\infty[\times\mathbb{R}$ alors $v'(t) = 0$ donc

$$u(\hat{t},\hat{x}) = v(\hat{t}) = v(0) = u(0,x(0)) = g(x(0)).$$

  En particulier, si $a(t,x) = A$ constante, alors $x(t) = \hat{x} + A(t-\hat{t})$ et donc $u(\hat{t},\hat{x}) = g(\hat{x} - A\hat{t})$.

- ▷ si $f(t,x) = 0$ pour tout $(t,x)\in[0;+\infty[\times\mathbb{R}$ alors $v'(t) + b(t,x(t))v(t) = 0$ donc

$$v(t) = v(0)e^{-\int_0^t b(s,x(s))ds} = u(0,x(0))e^{-\int_0^t b(s,x(s))ds} = g(x(0))e^{-\int_0^t b(s,x(s))ds}.$$

  On a alors

$$u(\hat{t},\hat{x}) = v(\hat{t}) = g(x(0))e^{-\int_0^{\hat{t}} b(s,x(s))ds}.$$

  En particulier, si $b(t,x) = B$ constante, alors $u(\hat{t},\hat{x}) = g(x(0))e^{-B\hat{t}}$. Si, de plus, $a(t,x) = A$ constante, alors $x(t) = \hat{x} + A(t-\hat{t})$ et donc $u(\hat{t},\hat{x}) = g(\hat{x} - A\hat{t})e^{-B\hat{t}}$.

Dans le cas général, on doit résoudre l'EDO $v'(t) + b(t,x(t))v(t) = f(t,x(t))$. Comme on connaît déjà la solution de l'équation homogène, il ne reste à calculer qu'une solution particulière de l'équation complète et on trouve

$$\begin{aligned} u(\hat{t},\hat{x}) = v(\hat{t}) &= v(0)e^{-\int_0^{\hat{t}} b(s,x(s))ds} + \int_0^{\hat{t}} f(s,x(s))e^{-\int_s^{\hat{t}} b(r,x(r))dr}\,ds \\ &= g(x(0))e^{-\int_0^{\hat{t}} b(s,x(s))ds} + \int_0^{\hat{t}} f(s,x(s))e^{-\int_s^{\hat{t}} b(r,x(r))dr}\,ds. \end{aligned}$$

Exercice 1.1 En utilisant la méthode des caractéristiques, calculer la solution faible $u(t,x)$ pour $t \geq 0$ et $x \in \mathbb{R}$ du problème de CAUCHY suivant :

$$\begin{cases} \partial_t u + x\partial_x u = 0 & x \in \mathbb{R},\ t > 0 \\ u(0,x) = x^4 & x \in \mathbb{R}, \end{cases}$$

Correction
$a(t,x) = x$, $b(t,x) = f(t,x) = 0$, $g(x) = x^4$ donc pour $(\hat{t},\hat{x})$ donné on a $x(t) = \hat{x}e^{t-\hat{t}}$ et donc

$$u(\hat{t},\hat{x}) = g(x(0)) = g(\hat{x}e^{-\hat{t}}) = \left(\hat{x}e^{-\hat{t}}\right)^4$$

Exercice 1.2 En utilisant la méthode des caractéristiques, calculer la solution faible $u(t,x)$ pour $t \geq 0$ et $x \in \mathbb{R}$ du problème de CAUCHY suivant :

$$\begin{cases} (t+1)\partial_t u + \partial_x u = 0 & x \in \mathbb{R},\ t > 0 \\ u(0,x) = x & x \in \mathbb{R}, \end{cases}$$

Correction
$a(t,x) = \frac{1}{t+1}$, $b(t,x) = f(t,x) = 0$, $g(x) = x$ donc pour $(\hat{t},\hat{x})$ donné on a $x(t) = \hat{x} + \ln\left(\frac{t+1}{\hat{t}+1}\right)$ et donc

$$u(\hat{t},\hat{x}) = g(x(0)) = g(\hat{x} - \ln(\hat{t}+1)) = \hat{x} - \ln(\hat{t}+1)$$

Exercice 1.3 En utilisant la méthode des caractéristiques, calculer la solution faible $u(t,x)$ pour $t \geq 0$ et $x \in \mathbb{R}$ du problème de CAUCHY suivant :

$$\begin{cases} \partial_t u + xt\partial_x u = t & x \in \mathbb{R},\ t > 0 \\ u(0,x) = x^2 & x \in \mathbb{R}, \end{cases}$$

Correction
$a(t,x) = xt$, $b(t,x) = 0$, $f(t,x) = t$, $g(x) = x^2$ donc pour $(\hat{t},\hat{x})$ donné on a $x(t) = \hat{x}e^{(t^2-\hat{t}^2)/2}$ et donc

$$u(\hat{t},\hat{x}) = g(x(0)) + \int_0^t f(s,x(s))ds = g(\hat{x}e^{-\hat{t}^2/2}) + \int_0^t s\,ds = \hat{x}^2 e^{-\hat{t}^2} + \frac{t^2}{2}.$$

Exercice 1.4 En utilisant la méthode des caractéristiques, calculer la solution faible $u(t,x)$ pour $t \geq 0$ et $x \in \mathbb{R}$ du problème de CAUCHY suivant :

$$\begin{cases} \partial_t u + \partial_x u + u = e^{-t+2x} & x \in \mathbb{R},\ t > 0 \\ u(0,x) = 0 & x \in \mathbb{R}, \end{cases}$$

Correction
$a(t,x) = 1$, $b(t,x) = 1$, $f(t,x) = e^{-t+2x}$, $g(x) = 0$ donc pour $(\hat{t},\hat{x})$ donné on a $x(t) = \hat{x} + t - \hat{t}$ et donc

$$u(\hat{t},\hat{x}) = g(x(0))e^{-\hat{t}} + \int_0^{\hat{t}} e^{-s+2x(s)}e^{-(\hat{t}-s)}ds = g(\hat{x}-\hat{t})e^{-\hat{t}} + \frac{1}{2}e^{2\hat{x}-\hat{t}} = \frac{1}{2}e^{2\hat{x}-\hat{t}}.$$

Chapitre 2.

Méthode des différences finies

À part dans quelques cas très particulier, il est impossible de calculer explicitement des solutions de modèles issus de la physique. Il est donc nécessaire d'avoir recours au calcul numérique sur ordinateur pour estimer qualitativement et quantitativement ces solutions. Le principe de toutes les méthodes de résolution numérique des équations aux dérivées partielles est d'obtenir des valeurs numériques discrètes (c'est-à-dire en nombre fini) qui *approchent* (en un sens convenable à préciser) la solution exacte. Dans ce procédé il faut bien être conscient de deux points fondamentaux : premièrement, on ne calcul pas des solutions exactes mais approchées ; deuxièmement, on *discrétise* le problème en représentant des fonctions par un nombre finis de valeurs, c'est-à-dire que l'on passe du *continu* au *discret*.

Il existe de nombreuses méthodes d'approximation numérique des solutions d'équations aux dérivées partielles. Pour simplifier la présentation, nous nous limiterons dans ce chapitre à la dimension un d'espace.

2.1. Principe de la méthode des différences finies

Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction de classe $\mathscr{C}^1(\mathbb{R})$. Comme

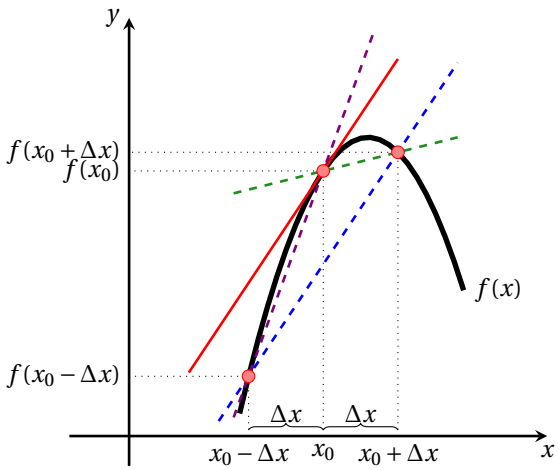
$$f'(x_0) = \lim_{h\to 0} \frac{f(x_0+h) - f(x_0)}{h}$$

il est naturel d'introduire les approximations

$$f'(x_0) \approx \frac{f(x_0+\Delta x) - f(x_0)}{\Delta x}, \tag{2.1}$$

$$f'(x_0) \approx \frac{f(x_0) - f(x_0-\Delta x)}{\Delta x}, \tag{2.2}$$

$$f'(x_0) \approx \frac{f(x_0+\Delta x) - f(x_0-\Delta x)}{2\Delta x}. \tag{2.3}$$



De manière analogue, la dérivée seconde peut être approchée par

$$f''(x_0) \approx \frac{f(x_0+\Delta x) - 2f(x_0) + f(x_0-\Delta x)}{(\Delta x)^2}.$$

Considérons un développement de Taylor en x autour du point x_0

$$f(x_0 \pm \Delta x) = f(x_0) \pm \Delta x f'(x_0) + (\Delta x)^2 f''(x_0) + O((\Delta x)^3),$$

alors

$$\frac{f(x_0+\Delta x) - f(x_0)}{\Delta x} = \frac{f(x_0) + \Delta x f'(x_0) + (\Delta x)^2 f''(x_0) + O((\Delta x)^3) - f(x_0)}{\Delta x} = f'(x_0) + O(\Delta x),$$

$$\frac{f(x_0) - f(x_0-\Delta x)}{\Delta x} = \frac{f(x_0) - f(x_0) + \Delta x f'(x_0) - (\Delta x)^2 f''(x_0) + O((\Delta x)^3)}{\Delta x} = f'(x_0) + O(\Delta x),$$

$$\frac{f(x_0+\Delta x) - f(x_0-\Delta x)}{2\Delta x} = \frac{f(x_0) + \Delta x f'(x_0) + (\Delta x)^2 f''(x_0) + O((\Delta x)^3) - f(x_0) + \Delta x f'(x_0) - (\Delta x)^2 f''(x_0)}{2\Delta x}$$

$$= f'(x_0) + O((\Delta x)^2),$$

et pour l'approximation de la dérivée seconde on a

$$\frac{f(x_0+\Delta x) - 2f(x_0) + f(x_0-\Delta x)}{(\Delta x)^2}$$
$$= \frac{f(x_0) + \Delta x f'(x_0) + (\Delta x)^2 f''(x_0) + O((\Delta x)^3) - 2f(x_0) + f(x_0) - \Delta x f'(x_0) + (\Delta x)^2 f''(x_0)}{(\Delta x)^2}$$
$$= f''(x_0) + O((\Delta x)^2).$$

Si Δx est «petit», ces formules sont des «bonnes» approximations.

2.2. Généralisation aux équations aux dérivées partielles

Nous nous limitons pour le moment à la dimension un d'espace et considérons une équation aux dérivées partielles $F(u) = 0$ définie pour $(x,t) \in \mathbb{R} \times \mathbb{R}^+$ avec une condition initiale $u(x,0) = g(x)$ pour $x \in \mathbb{R}$ (remarquons que $F(u)$ est une notation pour une fonction de u et de ses dérivées partielles en tout point). Pour discrétiser le domaine $\mathbb{R} \times \mathbb{R}^+$, on introduit un pas d'espace $\Delta x > 0$ et un pas de temps $\Delta t > 0$ et on définit les nœuds d'un maillage régulier

$$(x_j, t^n) = (j\Delta x, n\Delta t) \qquad \text{pour } j \in \mathbb{Z},\ n \in \mathbb{N}.$$

On note u_j^n la valeur d'une solution discrète approchée au point (x_j, t^n) et $u(x,t)$ la solution exacte (inconnue). Le principe de la méthode des différences finies est de remplacer les dérivées par des différences finies en utilisant des formules de Taylor dans lesquelles on néglige les restes.

Dans tous schéma il y a bien sûr une donnée initiale pour démarrer les itération en n : les valeurs initiales $(u_j^0)_{j\in\mathbb{Z}}$ sont définies par exemple par $u_j^0 = g(x_j)$ où g est la donnée initiale de l'équation.

S'il y a un second membre $f(x,t)$ dans l'équation aux dérivées partielles, alors les schémas se modifient en remplaçant zéro au second membre par une approximation consistante de $f(x,t)$ au point (x_j, t^n).

Si l'équation est définie sur un domaine borné, par exemple $x \in [\alpha;\beta]$, le maillage spatiale sera restreint à cet intervalle c'est-à-dire $j \in \{0,1,\dots,N\}$ avec $x_0 = \alpha$ et $x_N = \beta$ et $\Delta x = (\beta-\alpha)/(N+1)$. Il faut de plus ajouter des conditions aux limites qui peuvent être de plusieurs types.

Par exemple, si on a des conditions aux limites de Dirichlet

$$u(\alpha,t) = L, \quad u(\beta,t) = R, \qquad \text{pour } t \in \mathbb{R}_*^+,$$

elles se traduisent au niveau discret en

$$u_0^n = L, \quad u_{N+1}^n = R, \qquad \text{pour } n \in \mathbb{N}.$$

Si on a des conditions de Neumann

$$\partial_x u(\alpha,t) = L, \quad \partial_x u(\beta,t) = R, \qquad \text{pour } t \in \mathbb{R}_*^+,$$

elles se traduisent au niveau discret en

$$\frac{u_1^n - u_0^n}{\Delta x} = L, \quad \frac{u_N^n - u_{N-1}^n}{\Delta x} = R, \qquad \text{pour } n \in \mathbb{N}.$$

Si on a des conditions périodiques

$$u(x+\beta,t) = u(x+\alpha,t), \qquad \text{pour } x \in [\alpha;\beta],\ t \in \mathbb{R}_*^+$$

elles se traduisent au niveau discret en

$$u_0^n = u_N^n, \qquad \text{pour } n \in \mathbb{N},$$

et plus généralement $u_j^n = u_{N+j}^n$.

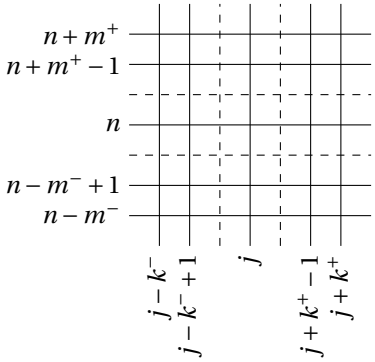


FIGURE 2.1. – Exemple de stencil pour un schéma à $m^- + m^+ + 1$ niveaux et $k^- + k^+ + 1$ points.

Définition 1 (Niveaux et stencil) Un schéma est dit à m niveaux s'il ne fait intervenir que m indices de temps. Les schémas les plus populaires sont des schémas à deux ou trois niveaux.

La collection des couples (j', n') qui interviennent dans l'équation discrète au point (j, n) est appelé *stencil* du schéma (qu'on peut essayer de traduire par support). En général, plus le stencil est large, plus le schéma est coûteux et difficile à programmer.

2.3. Consistance et ordre de précision

De manière générale, un schéma aux différences finies est défini, pour tous les indices possibles $j \in \mathbb{Z}$ et $n \in \mathbb{N}$, par la formule

$$F_{\Delta x,\Delta t}\left(\left\{u_{j+k}^{n+m}\right\}_{\substack{k^-\le k\le k^+\\ m^-\le m\le m^+}}\right) = 0$$

où les entiers k^-, k^+, m^- et m^+ définissent la largeur du stencil du schéma.

Un des buts de l'analyse numérique est de comparer et de sélectionner les meilleurs schémas suivant des critères de précision, de coût ou de robustesse.

Définition 2 (Erreur de troncature) Considérons le schéma aux différence finies $F_{\Delta x,\Delta t}(\{u_{j+k}^{n+m}\})$ pour l'approximation de l'équation aux dérivées partielles $F(u) = 0$. Soit $u(x, t)$ une solution suffisamment régulière de cette équation. On appelle *erreur de troncature* du schéma la quantité

$$\tau_j^n \equiv F_{\Delta x,\Delta t}\left(\left\{u(x + k\Delta x, t + m\Delta t)\right\}_{\substack{k^-\le k\le k^+\\ m^-\le m\le m^+}}\right).$$

Concrètement on calcule l'erreur de troncature d'un schéma en remplaçant u_{j+k}^{n+m} dans la formule par $u(x + k\Delta x, t + m\Delta t)$.

Définition 3 (Consistance) Le schéma aux différence finies $F_{\Delta x,\Delta t}(\{u_{j+k}^{n+m}\})$ est dit *consistant* avec l'équation aux dérivées partielles $F(u) = 0$ si l'*erreur de troncature* du schéma tend vers zéro, uniformément par rapport à (x, t), lorsque Δx et Δt tendent vers zéro indépendamment.

Définition 4 (Ordre de consistance) Le schéma aux différence finies $F_{\Delta x,\Delta t}(\{u_{j+k}^{n+m}\})$ est *précis à l'ordre p en espace et à l'ordre q en temps* avec l'équation aux dérivées partielles $F(u) = 0$ si l'*erreur de troncature* du schéma tend vers zéro comme $O((\Delta x)^p + (\Delta t)^q)$ lorsque Δx et Δt tendent vers zéro.

2.4. Convergence

La convergence d'un schéma aux différences finies est une propriété naturelle qui assure que, pour des valeurs suffisamment petites des pas d'espace et de temps, la solution numérique calculée sera proche de la solution exacte du problème de départ.

Définition 5 (Convergence) Le schéma aux différence finies $F_{\Delta x,\Delta t}(\{u_{j+k}^{n+m}\})$ utilisé pour la résolution numérique de l'équation aux dérivées partielles $F(u)=0$ est *convergent* si, pour toute solution u de l'équation $F(u)=0$, la suite u_j^n converge vers $u(x_j,t^n)$ avec $(\Delta x,\Delta t)\to(0,0)$.

Malheureusement la notion de consistance ne suffit pas à garantir que le schéma soit convergente comme on verra sur des exemples. Pour introduire un critère (très pratique) qui permet de voir si un schéma donné est convergent nous allons introduire la notion de stabilité.

2.5. Stabilité

Autre les outils qui permettent de comparer les performances des différents schémas, on doit également choisir les pas Δx et Δt de sorte que le schéma correspondant donnera une solution approchée correcte, au sens où une petite perturbation de la donnée initiale g n'induira par une perturbation trop grande sur la solution calculée au temps final. Cette idée, déjà rencontrée pour la définition de problème bien posé, est à la base du concept de stabilité pour les schémas aux différences finies.

Soit $u^n\equiv(u_j^n)_{1\le j\le N-1}$ la solution numérique d'un schéma.

Définition 6 (Stabilité) Un schéma aux différences finies est dit *stable* pour la norme $\|\cdot\|$ s'il existe une constante $K>0$ indépendante de Δx et Δt (lorsque ces valeurs tendent vers zéro) telle que

$$\|u^n\|\le K\|u^0\| \qquad \text{pour tout } n\ge 0,$$

quelle que soit la donnée initiale u^0. Si cette inégalité n'a lieu que pour des pas Δx et Δt astreints à certaines inégalités, on dit que le schéma est *conditionnellement stable.*

On définit les normes classiques

$$\|u^n\|_p=\left(\sum_{j=1}^{N-1}\Delta x\,|u_j^n|^p\right)^{\frac{1}{p}} \qquad \text{pour } 1\le p<+\infty,$$
$$\|u^n\|_\infty=\max_{1\le j\le N-1}|u_j^n|.$$

Définition 7 (Schéma linéaire) Un schéma aux différences finies est dit linéaire si la formule $F_{\Delta x,\Delta t}(\{u_{j+k}^{n+m}\})=0$ qui le définit est linéaire par rapport à ses arguments u_{j+k}^{n+m}.

La stabilité d'un schéma linéaire à deux niveaux est facile à interpréter. En effet, par linéarité tout schéma linéaire à deux niveaux peut s'écrire sous la forme condensée

$$\mathbb{A}u^n=u^{n+1},$$

où $\mathbb{A}$ est une matrice (dite d'itération) et on obtient $\mathbb{A}^n u^0=u^{n+1}$ (attention, la notation $\mathbb{A}^n$ désigne ici la puissance n-ème de $\mathbb{A}$) et par conséquent la stabilité du schéma est équivalente à

$$\|\mathbb{A}^n u^0\|\le K\|u^0\|, \qquad \forall n\ge 0,\ \forall u^0\in\mathbb{R}^{N-1}.$$

Introduisant la norme matricielle subordonnée $\|\mathbb{M}\|=\sup_{u\in\mathbb{R}^{N-1},u\neq 0}\frac{\|\mathbb{M}u\|}{\|u\|}$, la stabilité du schéma est équivalente à

$$\|\mathbb{A}^n\|\le K \qquad \forall n\ge 0$$

qui veut dire que la suite des puissances de $\mathbb{A}$ est bornée.

Définition 8 (Principe du maximum discret - stabilité L^∞) Un schéma aux différences finies vérifie le principe du maximum discret si pour tout $n \geq 0$ et tout $1 \leq j \leq N-1$ on a

$$\min_{0\leq j\leq N}\left(u_j^0\right) \leq u_j^n \leq \max_{0\leq j\leq N}\left(u_j^0\right)$$

quelle que soit la donné initiale u^0.

Stabilité L^2

La norme L^2 se prête bien à l'étude de la stabilité grâce à l'outil très puissant de l'analyse de Fourier. Supposons désormais que les conditions aux limites pour l'équation aux dérivées partielles sont des conditions aux limites de périodicité. À chaque vecteur $u^n \equiv (u_j^n)_{1\leq j\leq N-1}$ on associe une fonction $u^n(x)$, constante par morceaux, périodique, définie sur $[\alpha;\beta]$ par

$$u^n(x) = u_j^n \qquad \text{si } x_{j-1/2} < x < x_{j+1/2}$$

avec $x_{j+1/2} = \alpha + (j+1/2)\Delta x$ pour $0 \leq j \leq N$, $x_{-1/2} = \alpha$ et $x_{N+1/2} = \beta$. Ainsi définie, la fonction $u^n(x)$ appartient à $L^2([\alpha;\beta])$, elle peut donc se décomposer en la somme de Fourier

$$u^n(x) = \sum_{k\in\mathbb{Z}} \hat{u}^n(k) e^{2i\pi kx}$$

avec $\hat{u}^n(k) = \int_\alpha^\beta u^n(x) e^{-2i\pi kx} \mathrm{d}x$ et la formule de Plancherel

$$\int_\alpha^\beta |u^n(x)|^2 \mathrm{d}x = \sum_{k\in\mathbb{Z}} |\hat{u}^n(k)|^2.$$

Remarquons que même si $u^n(x)$ est une fonction réelle, les coefficients $\hat{u}^n(k)$ de la série de Fourier sont complexes. Une propriété importante pour l'étude de stabilité de la transformée de Fourier des fonctions périodiques est la suivantes : si on note $v^n(x) = u^n(x+\Delta x)$ alors $\hat{v}^n(k) = \hat{u}^n(k) e^{2i\pi k\Delta x}$.

Définition 9 («Recette» pour un schéma à deux niveaux) On injecte dans le schéma un mode de Fourier, on obtient ainsi

$$u_j^n = A(k)^n e^{2i\pi kx_j}$$

et on en déduit la valeur du facteur d'amplification $A(k)$. Rappelons que pour l'instant nous nous sommes limité au cas scalaire, c'est-à-dire que $A(k)$ est un nombre complexe. On appelle *condition de stabilité de Von Neumann* l'inégalité

$$|A(k)| \leq 1 \qquad \text{pour tout mode } k \in \mathbb{Z}.$$

Si la condition de stabilité de Von Neumann est satisfaite (avec éventuellement des restrictions sur Δx et Δt), alors le schéma est stable pour la norme L^2, sinon il est instable.

Dans la plupart des cas, on va trouver des restrictions sur Δx et Δt pour obtenir la stabilité au sens L^2 du schéma. Comme Δx est initialement fixé, ceci nous oblige à nous donner un pas de temps Δt petit. Plus cette condition de stabilité est restrictive, plus le schéma sera couteux à utiliser d'un point de vue du temps de calcul. Au contrario, les schémas inconditionnellement stables ne nécessitent aucune restriction particulière et donc peuvent être à priori utilisés pour une valeur quelconque de Δt. Ceci ne signifie pas pour autant qu'ils seront des «bons» schémas, et notamment que la solution calculée sera proche de la solution exacte. En effet, un choix trop grand de Δt donne une mauvaise approximation de la dérivée partielle par rapport au temps. En pratique, un schéma instable est inutilisable car, même si on part d'une donnée initiale spécialement préparée de manière à ce qu'aucun des modes de Fourier instables ne soit excité par elle, les inévitables erreurs d'arrondi vont créer des composantes non nulles (bien que très petites) de la solution sur ces modes instables. La croissance exponentielle de ces modes instables entraîne qu'après seulement quelque pas en temps ces modes deviennent énormes et polluent complètement le reste de la solution numérique.

2.6. Stabilité + Consistance = Convergence

Théorème 1 (Théorème de Lax) Soit $u(x,t)$ la solution suffisamment régulière de l'équation aux dérivée partielles $F(u)=0$ avec des conditions aux limites appropriées. Soit u_j^n la solution numérique discrète obtenue par un schéma aux différences finies avec la donnée initiale $u_j^0 = g(x_j)$. Si le schéma est linéaire, à deux niveaux, consistant et stable pour une norme $\|\cdot\|$, alors le schéma est convergent au sens où

$$\forall T>0, \qquad \lim_{\Delta x,\Delta t\to 0}\left(\sup_{t^n\le T}\|u_j^n - u(x_j,t^n)\|\right)=0.$$

De plus, si le schéma est précis à l'ordre p en espace et à l'ordre q en temps, alors pour tout $T>0$ il existe une constante $C_T>0$ telle que

$$\sup_{t^n\le T}\|u_j^n - u(x_j,t^n)\| \le C_T\left((\Delta x)^p + (\Delta t)^q\right).$$

D'un point de vue pratique ce théorème est très rassurant : si l'on utilise un schéma consistant (ils sont construit pour cela en général) et que l'on n'observe pas d'oscillations numériques (c'est-à-dire qu'il est stable), alors la solution numérique est proche de la solution exacte (le schéma converge).

2.7. Équation équivalente : diffusion et dispersion

Pour comparer divers schémas consistants et stables (donc convergents) d'un point de vue pratique, un concept pertinent (quoique formel) est celui d'équation équivalente.

Définition 10 (Équation équivalente) On appelle équation équivalente d'un schéma l'équation obtenue en ajoutant au modèle étudié la partie principale (c'est-à-dire le terme d'ordre le plus bas) de l'erreur de troncature.

Tous les schéma qu'on va voir sont consistants. Cependant, si on ajoute à l'équation la partie principale de l'erreur de troncature d'un schéma, alors ces schémas non seulement sont encore consistants avec cette nouvelle équation «équivalente», mais sont même strictement plus précis pour cette équation équivalente. En d'autre termes, les schémas sont «plus consistants» avec l'équation équivalente qu'avec l'équation qu'on veut approcher.

Cette équation va nous donner des renseignements précieux sur le comportement numérique du schéma. Le coefficient de diffusion (c'est-à-dire le coefficient de la dérivée seconde) de l'équation équivalente est appelé diffusion numérique. S'il est grand on dit que le schéma est diffusif (ou dissipatif). Le comportement typique d'un schéma diffusif est sa tendance à étaler artificiellement les données initiales au cours du temps. Si le schéma est précis d'ordre 2 alors l'équation équivalente ne contient pas de terme de diffusion mais un terme du troisième ordre, dit dispersif. Le comportement typique d'un schéma dispersif est qu'il produit des oscillation lorsque la solution est discontinue. En effet, le terme dispersif modifie la vitesse de propagation des modes de Fourier de la solution (particulièrement des modes de fréquence élevée), alors qu'un terme diffusif ne fait qu'atténuer son amplitude.

Chapitre 3.

Résolution numérique de l'équation de transport

On considère le problème de transport en une dimension d'espace dans le domaine borné $[0;L]$ avec une vitesse c constante non nulle et des conditions aux limites de périodicité

$$\begin{cases} \partial_t u(x,t) + c\partial_x u(x,t) = 0 & \text{pour } (x,t) \in [0;L] \times [0;T], \\ u(x+L,t) = u(x,t) & \text{pour } (x,t) \in [0;L] \times [0;T], \\ u(x,0) = g(x) & \text{pour } x \in [0;L]. \end{cases}$$

On souhaite calculer la valeur de la solution u en un ensemble discret de points en espace et en temps. Plus précisément, en fixant un pas d'espace $\Delta x = L/N > 0$ (N entier positif) et un pas de temps $\Delta t > 0$, on cherche à calculer $u_j^n \approx u(j\Delta x, n\Delta t)$ la valeur d'une solution discrète approchée au point (x_j, t^n). On sait que $u(x,t) = g(x-ct)$ est la solution exacte. Les conditions aux limites de périodicité conduisent aux égalités $u_1^n = u_{N+1}^n$ pour tout $n \geq 0$, par conséquent l'inconnue discrète à chaque pas de temps est un vecteur $u^n = (u_j^n)_{1\leq j\leq N}$.

Notre stratégie consiste à remplacer des opérateurs différentiels par des quotients aux différences finies. En utilisant différentes façon d'évaluer les dérivées partielles, beaucoup de choix de schémas sont possibles. Nous voulons en étudier ici quelques uns.

Soit $\alpha := c\frac{\Delta t}{\Delta x}$. On considère les schémas aux différences finies suivants :

❶ le schéma décentré à gauche

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} + c\frac{u_j^n - u_{j-1}^n}{\Delta x} = 0 \qquad \textit{i.e.} \qquad u_j^{n+1} = u_j^n - \alpha(u_j^n - u_{j-1}^n)$$

❷ le schéma décentré à droite

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} + c\frac{u_{j+1}^n - u_j^n}{\Delta x} = 0 \qquad \textit{i.e.} \qquad u_j^{n+1} = u_j^n - \alpha(u_{j+1}^n - u_j^n)$$

❸ le schéma centré

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} = 0 \qquad \textit{i.e.} \qquad u_j^{n+1} = u_j^n - \alpha\frac{u_{j+1}^n - u_{j-1}^n}{2}$$

❹ le schéma upwind (décentré amont)

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} + \frac{c+|c|}{2\Delta x}(u_j^n - u_{j-1}^n) + \frac{c-|c|}{2\Delta x}(u_{j+1}^n - u_j^n) = 0$$

i.e.

$$u_j^{n+1} = u_j^n - \left(\frac{\alpha+|\alpha|}{2}(u_j^n - u_{j-1}^n) + \frac{\alpha-|\alpha|}{2}(u_{j+1}^n - u_j^n)\right) = \begin{cases} u_j^n - \alpha(u_j^n - u_{j-1}^n) & \text{si } \alpha > 0, \\ u_j^n - \alpha(u_{j+1}^n - u_j^n) & \text{si } \alpha < 0. \end{cases}$$

❺ le schéma de Lax-Friedrichs

$$\frac{u_j^{n+1} - \frac{u_{j+1}^n + u_{j-1}^n}{2}}{\Delta t} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} = 0 \qquad \textit{i.e.} \qquad u_j^{n+1} = \frac{1-\alpha}{2}u_{j+1}^n + \frac{1+\alpha}{2}u_{j-1}^n$$

❻ le schéma de Lax-Wendroff

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}-\frac{c^2\Delta t}{2}\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{(\Delta x)^2}=0$$

i.e.

$$u_j^{n+1}=u_j^n-\alpha\frac{u_{j+1}^n-u_{j-1}^n}{2}+\alpha^2\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{2}$$

❼ le schéma de Beam-Warmimg (pour $c>0$)

$$u_j^{n+1}=u_j^n-\alpha\Big(g(u_{j-1}^n,u_j^n)-g(u_{j-2}^n,u_{j-1}^n)\Big)\quad\text{avec } g(A,B)=\frac{\alpha-1}{2}A+\frac{3-\alpha}{2}B$$

i.e.

$$u_j^{n+1}=u_j^n-\alpha\left((u_j^n-u_{j-1}^n)+\frac{(1-\alpha)}{2}(u_j^n-2u_{j-1}^n+u_{j-2}^n)\right)$$

ou encore

$$u_j^{n+1}=\frac{\alpha(\alpha-1)}{2}u_{j-2}^n+\alpha(2-\alpha)u_{j-1}^n+\frac{(\alpha-1)(\alpha-2)}{2}u_j^n$$

❽ le schéma de Fromm (pour $c>0$)

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{g(u_{j-1}^n,u_j^n,u_{j+1}^n)-g(u_{j-2}^n,u_{j-1}^n,u_j^n)}{\Delta x}=0\quad\text{avec } g(A,B,C)=\frac{\alpha-1}{4}A+B+\frac{1-\alpha}{4}C$$

i.e.

$$u_j^{n+1}=\frac{\alpha(\alpha-1)}{4}u_{j-2}^n+\frac{\alpha(5-\alpha)}{4}u_{j-1}^n+\frac{(1-\alpha)(\alpha+4)}{4}u_j^n+\frac{\alpha(\alpha-1)}{4}u_{j+1}^n$$

❾ le schéma anti-diffusif de Després-Lagoutière (pour $c>0$)

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{g(u_{j-1}^n,u_j^n,u_{j+1}^n)-g(u_{j-2}^n,u_{j-1}^n,u_j^n)}{\Delta x}=0$$

i.e.

$$u_j^{n+1}=u_j^n-\alpha(g(u_{j-1}^n,u_j^n,u_{j+1}^n)-g(u_{j-2}^n,u_{j-1}^n,u_j^n))$$

avec

$$g(L,C,R)=\begin{cases}A, & \text{si } R\le A,\\ B, & \text{si } R\ge B,\\ R, & \text{sinon,}\end{cases}$$

où

$$A=\max(L,C)+\frac{C-\max(L,C)}{\alpha},\qquad B=\min(L,C)+\frac{C-\min(L,C)}{\alpha}.$$

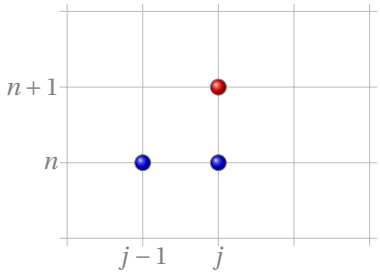
3.1. Exemples d'études de stabilité et de consistance

Exercice 3.1 Dessiner le stencil et étudier la stabilité L^2 et l'ordre de consistance du schéma ❶ (décentré à gauche) :

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_j^n-u_{j-1}^n}{\Delta x}=0\qquad \textit{i.e.}\qquad u_j^{n+1}=u_j^n-\alpha(u_j^n-u_{j-1}^n)$$

Correction

Stencil



Stabilité L^2. On utilise l'analyse de Fourier : pour $k \in \mathbb{Z}$, le coefficient de Fourier $\hat{u}^n(k)$ de la solution du schéma vérifie

$$\hat{u}^{n+1}(k) = \left[1 - \alpha + \alpha e^{-i2\pi k\Delta x}\right] \hat{u}^n(k).$$

En notant $\xi \equiv 2\pi k\Delta x$, on a

$$\begin{aligned}\hat{u}^{n+1}(k) = \left[1 - \alpha + \alpha e^{-i\xi}\right] \hat{u}^n(k) = \\ = \left[1 - \alpha + \alpha\big(\cos(-\xi) + i\sin(-\xi)\big)\right] \hat{u}^n(k) = \\ = \left[1 - \alpha + \alpha\cos(\xi) - i\alpha\sin(\xi)\right] \hat{u}^n(k).\end{aligned}$$

Après simplification on obtient

$$|\hat{u}^{n+1}(k)|^2 = |A(k)|^2 |\hat{u}^n(k)|^2$$

avec

$$\begin{aligned}|A(k)|^2 \equiv [1 - \alpha + \alpha\cos(\xi)]^2 + \alpha^2\sin^2\xi = \\ = 1 + 2\alpha(\alpha - 1)(1 - \cos(\xi)).\end{aligned}$$

On a

$$|A(k)| \le 1 \ \forall k \in \mathbb{Z} \iff 2\alpha(\alpha - 1)(1 - \cos(\xi)) \ge 0 \ \forall \xi \in \mathbb{R} \iff \alpha(\alpha - 1) \ge 0.$$

Pour $0 \le \alpha \le 1$ on a $|A(k)| \le 1$ pour toute fréquence $k \in \mathbb{Z}$, ce qui prouve que le schéma est stable en norme L^2 sous la condition CFL $0 \le \alpha \le 1$.

Ordre de consistance. On remplace u_i^m par $u(x_i, t^m)$ où u est une fonction régulière, $i = j-1, j$ et $m = n, n+1$. On définit l'erreur de troncature par

$$\tau_j^n \equiv \frac{u(x_j, t^{n+1}) - u(x_j, t^n)}{\Delta t} + c\frac{u(x_j, t^n) - u(x_{j-1}, t^n)}{\Delta x}.$$

On fait un développement de Taylor en x autour du point x_j et en t autour du point t^n et, comme u est solution de l'équation $\partial_t u = -c\partial_x u$, on a

$$\begin{aligned}u(x_{j-1}, t^n) &= u(x_j, t^n) - \Delta x\frac{\partial u}{\partial x}(x_j, t^n) + \frac{(\Delta x)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j, t^n) + O((\Delta x)^3), \\ u(x_j, t^{n+1}) &= u(x_j, t^n) + \Delta t\frac{\partial u}{\partial t}(x_j, t^n) + \frac{(\Delta t)^2}{2}\frac{\partial^2 u}{\partial t^2}(x_j, t^n) + O((\Delta t)^3) \\ &= u(x_j, t^n) + \Delta t\left(-c\frac{\partial u}{\partial x}(x_j, t^n)\right) + \frac{(\Delta t)^2}{2}\frac{\partial}{\partial t}\left(-c\frac{\partial u}{\partial x}(x_j, t^n)\right) + O((\Delta t)^3) \\ &= u(x_j, t^n) - c\Delta t\frac{\partial u}{\partial x}(x_j, t^n) - c\frac{(\Delta t)^2}{2}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial t}(x_j, t^n)\right) + O((\Delta t)^3) \\ &= u(x_j, t^n) - c\Delta t\frac{\partial u}{\partial x}(x_j, t^n) + c^2\frac{(\Delta t)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j, t^n) + O((\Delta t)^3).\end{aligned}$$

Par conséquent l'erreur de troncature se réécrit

$$\begin{aligned}\tau_j^n &\equiv \frac{u(x_j, t^{n+1}) - u(x_j, t^n)}{\Delta t} + c\frac{u(x_j, t^n) - u(x_{j-1}, t^n)}{\Delta x} \\ &= \left(c^2\frac{\Delta t}{2} - c\frac{\Delta x}{2}\right)\frac{\partial^2 u}{\partial x^2}(x_j, t^n) + O((\Delta x)^2 + (\Delta t)^2)\end{aligned}$$

$$= O((\Delta x) + (\Delta t)).$$

Le schéma est donc d'ordre 1 en temps et en espace.

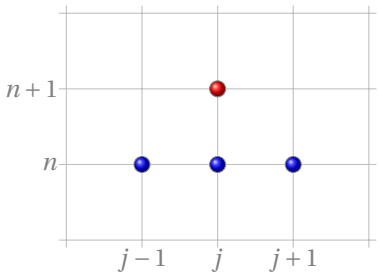
On aurait pu s'arrêter un ordre avant dans les développements de Taylor, mais vous voyez bien que même si on va plus loin on obtient le bon résultat ! Cependant, dans la consigne de l'exercice je vous ai suggéré l'ordre pour éviter de faire des calculs inutiles. Dans les exemples qui suivent on va s'arrêter au minimum nécessaire.

Exercice 3.2 Dessiner le stencil et étudier la stabilité L^2 et l'ordre de consistance du schéma ❸ (centré) :

Correction

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}=0 \qquad i.e. \qquad u_j^{n+1}=u_j^n-\frac{\alpha}{2}(u_{j+1}^n-u_{j-1}^n)$$

Stencil



Stabilité L^2. On utilise l'analyse de Fourier : pour $k \in \mathbb{Z}$, le coefficient de Fourier $\hat{u}^n(k)$ de la solution du schéma vérifie

$$\hat{u}^{n+1}(k)=\left[1-\frac{\alpha}{2}\left(e^{i2\pi k\Delta x}-e^{-i2\pi k\Delta x}\right)\right]\hat{u}^n(k).$$

En notant $\xi \equiv 2\pi k\Delta x$, on a

$$\begin{aligned}\hat{u}^{n+1}(k)&=\left[1-\frac{\alpha}{2}\left(e^{i\xi}-e^{-i\xi}\right)\right]\hat{u}^n(k)=\\&=\left[1-\frac{\alpha}{2}\left(\cos(\xi)+i\sin(\xi)-\cos(-\xi)-i\sin(-xi)\right)\right]\hat{u}^n(k)=\\&=\left[1-i\alpha\sin(\xi)\right]\hat{u}^n(k).\end{aligned}$$

Après simplification on obtient

$$|\hat{u}^{n+1}(k)|^2=|A(k)|^2|\hat{u}^n(k)|^2$$

avec

$$|A(k)|^2\equiv 1+\alpha^2\sin^2(\xi).$$

On a

$$|A(k)|\le 1\ \forall k\in\mathbb{Z} \iff \alpha^2\sin^2(\xi)\le 0\ \forall\xi\in\mathbb{R} \iff \nexists\alpha\in\mathbb{R}.$$

Ce qui prouve que le schéma est inconditionnellement instable en norme L^2.

Ordre de consistance. On remplace u_i^m par $u(x_i,t^m)$ où u est une fonction régulière, $i=j-1,j,j+1$ et $m=n,n+1$. On définit l'erreur de troncature par

$$\tau_j^n\equiv\frac{u(x_j,t^{n+1})-u(x_j,t^n)}{\Delta t}+c\frac{u(x_{j+1},t^n)-u(x_{j-1},t^n)}{2\Delta x}.$$

On fait un développement de Taylor en x autour du point x_j et en t autour du point t^n et, comme u est solution de l'équation $\partial_t u=-c\partial_x u$, on a

$$\begin{aligned}u(x_{j-1},t^n)&=u(x_j,t^n)-\Delta x\frac{\partial u}{\partial x}(x_j,t^n)+\frac{(\Delta x)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j,t^n)+O((\Delta x)^3),\\u(x_{j+1},t^n)&=u(x_j,t^n)+\Delta x\frac{\partial u}{\partial x}(x_j,t^n)+\frac{(\Delta x)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j,t^n)+O((\Delta x)^3),\end{aligned}$$

$$\begin{aligned} u(x_j,t^{n+1}) &= u(x_j,t^n)+\Delta t\frac{\partial u}{\partial t}(x_j,t^n)+O((\Delta t)^2)\\ &= u(x_j,t^n)+\Delta t\left(-c\frac{\partial u}{\partial x}(x_j,t^n)\right)+O((\Delta t)^2)\\ &= u(x_j,t^n)-c\Delta t\frac{\partial u}{\partial x}(x_j,t^n)+O((\Delta t)^2). \end{aligned}$$

Par conséquent l'erreur de troncature se réécrit

$$\tau_j^n \equiv \frac{u(x_j,t^{n+1})-u(x_j,t^n)}{\Delta t}+c\frac{u(x_{j+1},t^n)-u(x_{j-1},t^n)}{2\Delta x}=O((\Delta x)^2+(\Delta t)).$$

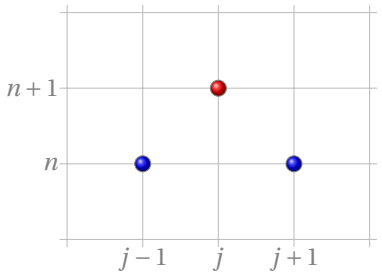
Le schéma est donc d'ordre 1 en temps et 2 en espace.

Exercice 3.3 Dessiner le stencil et étudier la stabilité L^2 et l'ordre de consistance du schéma ❺ (Lax-Friedrichs) :

Correction

$$\frac{u_j^{n+1}-\frac{u_{j+1}^n+u_{j-1}^n}{2}}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}=0 \qquad i.e. \qquad u_j^{n+1}=\frac{1-\alpha}{2}u_{j+1}^n+\frac{1+\alpha}{2}u_{j-1}^n$$

Stencil



Stabilité L^2**.** On utilise l'analyse de Fourier : pour $k\in\mathbb{Z}$, le coefficient de Fourier $\hat{u}^n(k)$ de la solution du schéma vérifie

$$\hat{u}^{n+1}(k)=\left[\frac{1-\alpha}{2}e^{i2\pi k\Delta x}-\frac{1+\alpha}{2}e^{-i2\pi k\Delta x}\right]\hat{u}^n(k).$$

En notant $\xi\equiv 2\pi k\Delta x$, on a

$$\hat{u}^{n+1}(k)=\left[\frac{1-\alpha}{2}e^{i\xi}+\frac{1+\alpha}{2}e^{-i\xi}\right]\hat{u}^n(k)=[\cos(\xi)-i\alpha\sin(\xi)]\,\hat{u}^n(k).$$

Après simplification on obtient

$$|\hat{u}^{n+1}(k)|^2=|A(k)|^2|\hat{u}^n(k)|^2$$

avec

$$|A(k)|^2\equiv\cos^2(\xi)+\alpha^2\sin^2(\xi).$$

On a

$$|A(k)|\le 1\ \forall k\in\mathbb{Z}\iff \cos^2(\xi)+\alpha^2\sin^2(\xi)\ge 1\ \forall\xi\in\mathbb{R}\iff \alpha^2\le 1.$$

Ce qui prouve que le schéma est stable en norme L^2 sous la condition CFL $|\alpha|\le 1$.

Ordre de consistance. On remplace u_i^m par $u(x_i,t^m)$ où u est une fonction régulière, $i=j-1,j,j+1$ et $m=n,n+1$. On définit l'erreur de troncature par

$$\tau_j^n\equiv\frac{u(x_j,t^{n+1})-\frac{u(x_{j+1},t^n)+u(x_{j-1},t^n)}{2}}{\Delta t}+c\frac{u(x_{j+1},t^n)-u(x_{j-1},t^n)}{2\Delta x}.$$

On fait un développement de Taylor en x autour du point x_j et en t autour du point t^n et, comme u est solution de l'équation $\partial_t u=-c\partial_x u$, on a

$$u(x_{j-1},t^n)=u(x_j,t^n)-\Delta x\frac{\partial u}{\partial x}(x_j,t^n)+O((\Delta x)^2),$$

$$
\begin{aligned}
u(x_{j+1},t^n) &= u(x_j,t^n) + \Delta x \frac{\partial u}{\partial x}(x_j,t^n) + O((\Delta x)^2),\\
u(x_j,t^{n+1}) &= u(x_j,t^n) + \Delta t \frac{\partial u}{\partial t}(x_j,t^n) + O((\Delta t)^2)\\
&= u(x_j,t^n) + \Delta t\left(-c\frac{\partial u}{\partial x}(x_j,t^n)\right) + O((\Delta t)^2)\\
&= u(x_j,t^n) - c\Delta t \frac{\partial u}{\partial x}(x_j,t^n) + O((\Delta t)^2).
\end{aligned}
$$

Par conséquent l'erreur de troncature se réécrit

$$
\tau_j^n \equiv \frac{u(x_j,t^{n+1}) - u(x_j,t^n)}{\Delta t} + c\frac{u(x_{j+1},t^n) - u(x_{j-1},t^n)}{2\Delta x} = O\left((\Delta x) + (\Delta t) + \frac{(\Delta x)^2}{\Delta t}\right).
$$

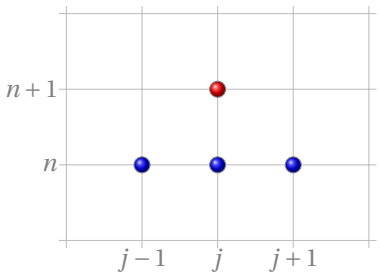
Étant donné que, sous la condition CFL calculée précédemment, $O\left(\frac{(\Delta x)^2}{\Delta t}\right) = O(\Delta x)$, le schéma est donc d'ordre 1 en temps et en espace.

Exercice 3.4 Dessiner le stencil et étudier la stabilité L^2 et l'ordre de consistance du schéma ❻ (Lax-Wendroff).

Correction
Pour $j = 0,\dots,N-1$ et $n \in \mathbb{N}$, le schéma de Lax-Wendroff s'écrit

$$
\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} - c^2\Delta t\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{2(\Delta x)^2} = 0 \qquad i.e. \qquad u_j^{n+1} = u_j^n - \alpha\frac{u_{j+1}^n - u_{j-1}^n}{2} + \alpha^2\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{2}
$$

Stencil



Stabilité L^2**.** On utilise l'analyse de Fourier : pour $k \in \mathbb{Z}$, le coefficient de Fourier $\hat{u}^n(k)$ de la solution du schéma vérifie

$$
\hat{u}^{n+1}(k) = \left[\frac{-\alpha+\alpha^2}{2}e^{i2\pi k\Delta x} + 1 - 2\alpha^2 + \frac{\alpha+\alpha^2}{2}e^{-i2\pi k\Delta x}\right]\hat{u}^n(k).
$$

En notant $\xi \equiv 2\pi k\Delta x$, on a

$$
\hat{u}^{n+1}(k) = \left[\frac{-\alpha+\alpha^2}{2}e^{i\xi} + 1 - \alpha^2 + \frac{\alpha+\alpha^2}{2}e^{-i\xi}\right]\hat{u}^n(k) = \left[1 - \alpha^2(1-\cos(\xi)) - i\alpha\sin(\xi)\right]\hat{u}^n(k).
$$

Après simplification on obtient

$$
|\hat{u}^{n+1}(k)|^2 = |A(k)|^2|\hat{u}^n(k)|^2
$$

avec

$$
|A(k)|^2 \equiv \alpha^2(\alpha^2-1)(\cos(\xi)-1)^2 + 1.
$$

On a

$$
|A(k)| \le 1 \ \forall k \in \mathbb{Z} \iff \alpha^2 \le 1.
$$

Ce qui prouve que le schéma est stable en norme L^2 sous la condition CFL $|\alpha| \le 1$.

Ordre de consistance. On remplace u_i^m par $u(x_i,t^m)$ où u est une fonction régulière, $i = j-1, j, j+1$ et $m = n, n+1$. On définit l'erreur de troncature par

$$
\tau_j^n \equiv \frac{u(x_j,t^{n+1}) - u(x_j,t^n)}{\Delta t} + c\frac{u(x_{j+1},t^n) - u(x_{j-1},t^n)}{2\Delta x} - c^2\Delta t\frac{u(x_{j+1},t^n) - 2u(x_j,t^n) + u(x_{j-1},t^n)}{2(\Delta x)^2} = 0.
$$

On fait un développement de Taylor en x autour du point x_j et en t autour du point t^n et, comme u est solution de l'équation $\partial_t u = -c\partial_x u$, on a

$$\begin{aligned}
u(x_{j-1}, t^n) &= u(x_j, t^n) - \Delta x \frac{\partial u}{\partial x}(x_j, t^n) + \frac{(\Delta x)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j, t^n) + O((\Delta x)^3),\\
u(x_{j+1}, t^n) &= u(x_j, t^n) + \Delta x \frac{\partial u}{\partial x}(x_j, t^n) + \frac{(\Delta x)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j, t^n) + O((\Delta x)^3),\\
u(x_j, t^{n+1}) &= u(x_j, t^n) + \Delta t \frac{\partial u}{\partial t}(x_j, t^n) + \frac{(\Delta t)^2}{2}\frac{\partial^2 u}{\partial t^2}(x_j, t^n) + O((\Delta t)^3)\\
&= u(x_j, t^n) + \Delta t\left(-c\frac{\partial u}{\partial x}(x_j, t^n)\right) + \frac{(\Delta t)^2}{2}\left(c^2\frac{\partial^2 u}{\partial x^2}(x_j, t^n)\right) + O((\Delta t)^3)\\
&= u(x_j, t^n) - c\Delta t\frac{\partial u}{\partial x}(x_j, t^n) + c^2\frac{(\Delta t)^2}{2}\frac{\partial^2 u}{\partial x^2}(x_j, t^n) + O((\Delta t)^3).
\end{aligned}$$

Par conséquent l'erreur de troncature se réécrit

$$\tau_j^n \equiv \frac{u(x_j, t^{n+1}) - u(x_j, t^n)}{\Delta t} + c\frac{u(x_{j+1}, t^n) - u(x_{j-1}, t^n)}{2\Delta x} - c^2\Delta t\frac{u(x_{j+1}, t^n) - 2u(x_j, t^n) + u(x_{j-1}, t^n)}{2(\Delta x)^2} = O((\Delta t)^2) + O((\Delta x)^2).$$

car, sous la condition CFL calculée précédemment, $O\left(c\frac{\Delta t}{\Delta x}\right) = O(1)$, le schéma est donc d'ordre 2 en temps et en espace.

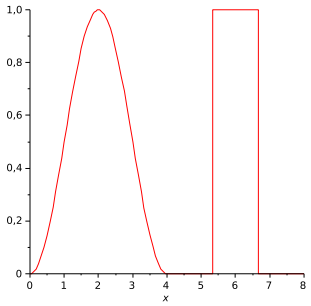
Chapitre 4.

TP

Soit la largeur du domaine $L = 8$ et le temps final $T = 24$.

On considère la condition initiale

$$g(x) = \begin{cases} \frac{1}{2} + \frac{1}{2}\sin\left(\frac{4\pi}{L}x - \frac{\pi}{2}\right) & \text{si } x \in \left]0; \frac{L}{2}\right[, \\ 0 & \text{si } x \in \left]\frac{L}{2}; \frac{2L}{3}\right[\cup \left]\frac{5L}{6}; L\right[, \\ 1 & \text{sinon.} \end{cases}$$



① Compléter le code fortran avec les schémas qui manquent (lorsque le schéma a été donné pour $c > 0$, réfléchir à comment il faut le modifier pour $c < 0$ et compléter le code).

② Pour $c = 1$ et une grille de $Nx = 50$ mailles, comparer les schémas ❶-❾ : d'abord on prendra la constante de Courant-Friedrichs-Levy égale à `cfl`= 0.99, puis `cfl`= 0.1 et enfin `cfl`= 1.1. Que remarque-t-on à propos de la stabilité ? Que remarque-t-on à propos de la diffusion ? Et de la dispersion ? Répéter les calculs sur une grille de $Nx = 500$ mailles. Répéter les calculs pour $c = -1$.

Annexe A.

Code Fortran 90

Langage : Fortran 90
Éditeur : Programmer's Notepad (sous Windows, coloration syntaxique de Fortran 90)
Système d'exploitation : Linux (via Cygwin)
Visualisation : Gnuplot

A.1. Instructions

Copier dans un sous-dossier du dossier `C:/cygwin` le fichier `transport.f90` et le dossier `data` où seront sauvegardé les sorties.

Lancer `Cygwin` depuis le menu : `Start` → `Cygwin-X` → `XWin-Serveur`

Pour compiler le programme `transport.f90` on utilisera la commande

```
gfortran transport.f90 -o transport.o
```

L'exécutable ainsi crée s'appelle `transport.o` qu'on lancera en tapant

```
./transport.o
```

Les deux instruction peuvent être exécutés l'une à la suite de l'autre en utilisant && :

```
gfortran transport.f90 -o transport.o && ./transport.o
```

Les résultats sont sauvegardés dans les fichiers `transportN.dat` où `N`= 0...`Smax` est le numéro de la sauvegarde. Ces fichier se trouvent dans le dossier `data`. Un fichier `transportN.dat` comporte `NX` lignes et 11 colonnes :

x	décentré gauche	décentré droite	centré	upwind	lax-friedrichs	lax-wendroff	beam-warmimg	fromm	antidiffusif

Pour comparer la solution approchée du schéma XXX avec la solution exacte au cours du temps, on tapera

```
cd data
gnuplot plot_XXX.gnu
```

Pour comparer les solutions approchées des schémas avec la solution exacte à l'instant final et sauvegarder l'image dans le fichier `comparaison_finale.png` on tapera

```
gnuplot plot_comaparaison_finale.gnu
```

Pour revenir au dossier précédent taper

```
cd ..
```

A.2. Code

```
!
! Resolution du
! probleme de Cauchy
! d_t u(x,t) + c d_x u(x,t) = 0
! u(x,t=0)=g(x)
! pour t>0 et x in [0;L] avec conditions au bord periodiques
!
! Creer une directory data dans laquelle seront sauvegarde les sorties
!
! --- Pour compiler && executer
! gfortran transport.f90 -o transport.o && ./transport.o
!
! --- Pour voir les ``films'' avec gnuplot
! cd data
! gnuplot plot_centree.gnu
! gnuplot plot_decentreegauche.gnu
! gnuplot plot_decentreedroite.gnu
! gnuplot plot_upwind.gnu
! gnuplot plot_laxfriedrichs.gnu
! gnuplot plot_laxwendroff.gnu
! gnuplot plot_beamwarming.gnu
! gnuplot plot_fromm.gnu
! gnuplot plot_antidiffusif.gnu
!


program transport

        implicit none
        double precision, parameter :: pi=2.0*acos(0.0)

        integer,                        parameter :: Nx   = 500         ! nombre de points
        double precision,       parameter :: L    = 8.0             ! domaine
        double precision,       parameter :: c    = 1.              ! vitesse du transport
        double precision,       parameter :: Tmax = 24.0                ! borne en temps
        integer,                        parameter :: Smax = 40                  ! nombre de sauvegardes
        double precision,       parameter :: cfl  = 0.99            ! coefficient CFL

        integer                                                 :: Nt, j, sauv
        double precision                                :: dx, dt, t
        double precision, dimension(Nx) :: x
        double precision, dimension(Nx) :: exacte
        double precision, dimension(Nx) :: centree
        double precision, dimension(Nx) :: decentreegauche
        double precision, dimension(Nx) :: decentreedroite
        double precision, dimension(Nx) :: upwind
        double precision, dimension(Nx) :: laxfriedrichs
        double precision, dimension(Nx) :: laxwendroff
        double precision, dimension(Nx) :: beamwarming
        double precision, dimension(Nx) :: fromm
        double precision, dimension(Nx) :: antidiffusif

        Nt = 0                                  ! numero de la sauvegarde
        dx = L/Nx                               ! pas d'espace
        dt = cfl*abs(c)*dx              ! pas de temps
        t  = 0.0                                ! instant

        x                               = (/(j*dx, j=1,Nx)/)    ! position
        exacte                  = (/(0., j=1,Nx)/)              ! solution exacte
        centree                 = (/(0., j=1,Nx)/)              ! solution avec schema centre
        decentreegauche = (/(0., j=1,Nx)/)              ! solution avec schema decentre a gauche
        decentreedroite = (/(0., j=1,Nx)/)              ! solution avec schema decentre a droite
        upwind                  = (/(0., j=1,Nx)/)              ! solution avec schema upwind
        laxfriedrichs   = (/(0., j=1,Nx)/)              ! solution avec schema de Lax-Friedrichs
        laxwendroff     = (/(0., j=1,Nx)/)              ! solution avec schema de Lax-Wendroff
        beamwarming     = (/(0., j=1,Nx)/)              ! solution avec schema de Beam-Warming
        fromm                   = (/(0., j=1,Nx)/)              ! solution avec schema de Fromm
        antidiffusif    = (/(0., j=1,Nx)/)              ! solution avec schema AntiDiffusif

        ! Initialisations
        call Calcul_Exacte(x,t,exacte)
        centree                 = exacte
        decentreegauche = exacte
        decentreedroite = exacte
        upwind                  = exacte
        laxfriedrichs   = exacte
        laxwendroff             = exacte
        beamwarming             = exacte
        fromm                   = exacte
        antidiffusif    = exacte

    ! Sauvegarde de la CI
    sauv = 0
```

```
    call sauvegarde(sauv,x,exacte,centree, decentreegauche,decentreedroite,upwind,laxfriedrichs, laxwendroff
        ➥, beamwarming,&
                    fromm, antidiffusif)


! ********************************************************************
! MARCHE EN TEMPS
! ********************************************************************
        do while (t<Tmax)

                Nt = Nt+1
                t  = t+dt

        ! calcul de la solution exacte
                call Calcul_Exacte(x,t,exacte)
                ! calcul des solutions approchees
                call Calcul_Centree(centree)
                call Calcul_Decentree_Gauche(decentreegauche)
                call Calcul_Decentree_Droite(decentreedroite)
                call Calcul_Upwind(upwind)
                call Calcul_Lax_Friedrichs(laxfriedrichs)
                call Calcul_Lax_Wendroff(laxwendroff)
                call Calcul_Beam_Warming(beamwarming)
                call Calcul_Fromm(fromm)
                call Calcul_Anti_Diffusif(antidiffusif)
                ! sauvegarde dans un fichier de toutes les solutions
                if ( floor(t*Smax/Tmax)>floor((t-dt)*Smax/Tmax) ) then
                        sauv=sauv+1
                        call sauvegarde(sauv,x,exacte,centree, decentreegauche,decentreedroite,upwind,
                            ➥laxfriedrichs, laxwendroff, beamwarming,&
                                                fromm, antidiffusif)
        end if
        end do

        call scriptgnuplot(sauv)



! ********************************************************************
! SOUS-PROGRAMMES
! ********************************************************************
        contains


        subroutine sauvegarde(sauv,x,exacte,centree, decentreegauche,decentreedroite,&
                              upwind,laxfriedrichs, laxwendroff, beamwarming, fromm, antidiffusif)
            integer                                         :: j
            integer,                                                intent(in)     :: sauv
            double precision, dimension(Nx), intent(in) :: x, exacte,centree, decentreegauche,
                ➥decentreedroite,upwind,&
                                                               laxfriedrichs, laxwendroff, beamwarming, fromm,
                                                                   ➥antidiffusif
            character(len=30)                                           :: file_name ! nome du fichier a
                ➥sauvegarder

            write(file_name,'("./data/transport",i3.3,".dat")')  sauv
            open(unit=11,file=file_name)
            do j=1,Nx
                write(11,*) x(j), &
                        exacte(j), &
                        centree(j), &
                        decentreegauche(j), &
                        decentreedroite(j), &
                        upwind(j), &
                        laxfriedrichs(j), &
                        laxwendroff(j), &
                        beamwarming(j), &
                        fromm(j), &
                        antidiffusif(j)
            end do
            close(11)
            print "(A,I3,A)", "␣=====␣Sauvegarde␣num.", sauv, "␣====="
        end subroutine


                subroutine Calcul_Exacte(x,t,exacte)
                        double precision, dimension(Nx), intent(in)     :: x
                        double precision,                                intent(in)     :: t
                        double precision, dimension(Nx), intent(out)    :: exacte
                        double precision, dimension(Nx)                                 :: xi
                        xi = x-c*t - floor((x-c*t)/L)*L
                        where (xi<L*0.5)
                                exacte = (1.+sin(4.*pi*xi/L-0.5*pi))*0.5
                        elsewhere (xi<L*0.66 .or. xi>L*0.833)
                                exacte = 0.0
```

```
                elsewhere
                        exacte = 1.0
                end where
        end subroutine Calcul_Exacte


        subroutine Calcul_Centree(centree)
                double precision, dimension(Nx), intent(inout)  :: centree
                integer
                    ➥        :: j
                double precision, dimension(Nx)                                 :: centree_P,
                    ➥centree_M
                double precision                                                ::
                    ➥alpha
                alpha = c*dt/dx
    ! conditions periodiques
                centree_P(1:Nx-1) = centree(2:Nx)
                centree_P(Nx)     = centree(1)
                centree_M(2:Nx)   = centree(1:Nx-1)
                centree_M(1)      = centree(Nx)
    ! schema
                centree = centree-alpha*0.5*(centree_P-centree_M)
        end subroutine Calcul_Centree


        subroutine Calcul_Decentree_Gauche(decentreegauche)
                double precision, dimension(Nx), intent(inout)  :: decentreegauche
                integer
                    ➥        :: j
                double precision, dimension(Nx)                                 :: decentreegauche_P
                    ➥, decentreegauche_M
                double precision                                                ::
                    ➥alpha
                alpha = c*dt/dx
    ! conditions periodiques
                decentreegauche_M(2:Nx)   = decentreegauche(1:Nx-1)
                decentreegauche_M(1)      = decentreegauche(Nx)
    ! schema
                decentreegauche = decentreegauche-alpha*(decentreegauche-decentreegauche_M)
        end subroutine Calcul_Decentree_Gauche


        subroutine Calcul_Decentree_Droite(decentreedroite)
                double precision, dimension(Nx), intent(inout)  :: decentreedroite
                integer
                    ➥        :: j
                double precision, dimension(Nx)                                 :: decentreedroite_P
                    ➥, decentreedroite_M
                double precision                                                ::
                    ➥alpha
                alpha = c*dt/dx
    ! conditions periodiques
                decentreedroite_P(1:Nx-1) = decentreedroite(2:Nx)
                decentreedroite_P(Nx)     = decentreedroite(1)
    ! schema
                decentreedroite = decentreedroite-alpha*(decentreedroite_P-decentreedroite)
        end subroutine Calcul_Decentree_Droite


        subroutine Calcul_Upwind(upwind)
                double precision, dimension(Nx), intent(inout)  :: upwind
                double precision, dimension(Nx)                                 :: upwind_P,
                    ➥upwind_M
    ! conditions periodiques
                upwind_P(1:Nx-1) = upwind(2:Nx)
                upwind_P(Nx)     = upwind(1)
                upwind_M(2:Nx)   = upwind(1:Nx-1)
                upwind_M(1)      = upwind(Nx)
    ! schema
                upwind = upwind-(dt/dx)*(c+abs(c))*0.5*(upwind-upwind_M)-(dt/dx)*(c-abs(c))*0.5*(
                    ➥upwind_P-upwind)
        end subroutine Calcul_Upwind


        subroutine Calcul_Lax_Friedrichs(laxfriedrichs)
                double precision, dimension(Nx), intent(inout)  :: laxfriedrichs
                double precision, dimension(Nx)                                 :: laxfriedrichs_P,
                    ➥laxfriedrichs_M
                double precision                                                ::
                    ➥alpha
                alpha = c*dt/dx
    ! conditions periodiques
                laxfriedrichs_P(1:Nx-1) = laxfriedrichs(2:Nx)
                laxfriedrichs_P(Nx)     = laxfriedrichs(1)
                laxfriedrichs_M(2:Nx)   = laxfriedrichs(1:Nx-1)
                laxfriedrichs_M(1)      = laxfriedrichs(Nx)
```

```
! schema
            laxfriedrichs = laxfriedrichs_P + laxfriedrichs_M - alpha*(laxfriedrichs_P-
                ➥laxfriedrichs_M)
            laxfriedrichs = 0.5*laxfriedrichs
    end subroutine Calcul_Lax_Friedrichs


    subroutine Calcul_Lax_Wendroff(laxwendroff)
            double precision, dimension(Nx), intent(inout)  :: laxwendroff
            double precision, dimension(Nx)                                  :: laxwendroff_P,
                ➥laxwendroff_M
            double precision                                                 ::
                ➥alpha
            alpha = c*dt/dx
! conditions periodiques
            laxwendroff_P(1:Nx-1) = laxwendroff(2:Nx)
            laxwendroff_P(Nx)     = laxwendroff(1)
            laxwendroff_M(2:Nx)   = laxwendroff(1:Nx-1)
            laxwendroff_M(1)      = laxwendroff(Nx)
! schema
            laxwendroff = laxwendroff &
                                - 0.5*alpha*(laxwendroff_P - laxwendroff_M) &
                                + 0.5*alpha**2*(laxwendroff_P-2*laxwendroff+laxwendroff_M)
    end subroutine Calcul_Lax_Wendroff


    subroutine Calcul_Beam_Warming(beamwarming)
            double precision, dimension(Nx), intent(inout)  :: beamwarming
            double precision, dimension(Nx)                                  :: beamwarming_P,
                ➥beamwarming_PP, beamwarming_M, beamwarming_MM, g_P, g_M
            double precision                                                 ::
                ➥alpha
            alpha = c*dt/dx
! conditions periodiques
            beamwarming_P(1:Nx-1)  = beamwarming(2:Nx)
            beamwarming_P(Nx)      = beamwarming(1)
            beamwarming_PP(1:Nx-1) = beamwarming_P(2:Nx)
            beamwarming_PP(Nx)     = beamwarming_P(1)
            beamwarming_M(2:Nx)    = beamwarming(1:Nx-1)
            beamwarming_M(1)       = beamwarming(Nx)
            beamwarming_MM(2:Nx)   = beamwarming_M(1:Nx-1)
            beamwarming_MM(1)      = beamwarming_M(Nx)
! schema
if (c>0) then
                    beamwarming=beamwarming-dt/dx*(beamwarming-beamwarming_M+0.5*(1.-alpha)*(
                        ➥beamwarming-2.*beamwarming_M+beamwarming_MM))
            else
                    ! a implementer
                    beamwarming = 0.0*beamwarming
            end if
    end subroutine Calcul_Beam_Warming


    subroutine Calcul_Fromm(fromm)
            double precision, dimension(Nx), intent(inout)  :: fromm
            double precision, dimension(Nx)                                  :: fromm_P, fromm_PP
                ➥, fromm_M, fromm_MM
            double precision                                                 ::
                ➥alpha
            alpha = c*dt/dx
! conditions periodiques
            fromm_P(1:Nx-1)  = fromm(2:Nx)
            fromm_P(Nx)      = fromm(1)
            fromm_PP(1:Nx-1) = fromm_P(2:Nx)
            fromm_PP(Nx)     = fromm_P(1)
            fromm_M(2:Nx)    = fromm(1:Nx-1)
            fromm_M(1)       = fromm(Nx)
            fromm_MM(2:Nx)   = fromm_M(1:Nx-1)
            fromm_MM(1)      = fromm_M(Nx)
! schema
if (c>0) then
                    fromm = alpha*(alpha-1.)*0.25*fromm_MM &
                              + alpha*(5.-alpha)*0.25*fromm_M &
                              + (1.-alpha)*(alpha+4.)*0.25*fromm &
                              + alpha*(alpha-1.)*0.25*fromm_P
            else
                    ! a implementer
                    fromm = 0.0*fromm
            end if
    end subroutine Calcul_Fromm


    subroutine Calcul_Anti_Diffusif(antidiffusif)
            double precision, dimension(Nx), intent(inout)  :: antidiffusif
            double precision, dimension(Nx)                                  :: antidiffusif_P,
                ➥antidiffusif_PP, antidiffusif_M, antidiffusif_MM
```

```
                double precision, dimension(Nx)                                 :: A_P, A_M, B_P,
                    ➥B_M, g_P, g_M
                double precision                                                ::
                    ➥alpha
                alpha = c*dt/dx
        ! conditions periodiques
                antidiffusif_P(1:Nx-1)  = antidiffusif(2:Nx)
                antidiffusif_P(Nx)      = antidiffusif(1)
                !antidiffusif_PP(1:Nx-1) = antidiffusif_P(2:Nx)
                !antidiffusif_PP(Nx)     = antidiffusif_P(1)
                antidiffusif_M(2:Nx)    = antidiffusif(1:Nx-1)
                antidiffusif_M(1)       = antidiffusif(Nx)
                antidiffusif_MM(2:Nx)   = antidiffusif_M(1:Nx-1)
                antidiffusif_MM(1)      = antidiffusif_M(Nx)
        ! schema
        if (c>0) then
                        A_P = max(antidiffusif_M ,antidiffusif  )+(antidiffusif  -max(antidiffusif_M
                            ➥ ,antidiffusif  ))/alpha
                        A_M = max(antidiffusif_MM,antidiffusif_M)+(antidiffusif_M-max(
                            ➥antidiffusif_MM,antidiffusif_M))/alpha
                        B_P = min(antidiffusif_M ,antidiffusif  )+(antidiffusif  -min(antidiffusif_M
                            ➥ ,antidiffusif  ))/alpha
                        B_M = min(antidiffusif_MM,antidiffusif_M)+(antidiffusif_M-min(
                            ➥antidiffusif_MM,antidiffusif_M))/alpha
                        where(antidiffusif_P<=A_P)
                                g_P=A_P
                        elsewhere(antidiffusif_P>B_P)
                                g_P=B_P
                        elsewhere
                                g_P=antidiffusif_P
                        end where
                        where(antidiffusif<=A_M)
                                g_M=A_M
                        elsewhere(antidiffusif>B_M)
                                g_M=B_M
                        elsewhere
                                g_M=antidiffusif
                        end where
                        antidiffusif = antidiffusif - alpha*(g_P-g_M)
                else
                        ! a implementer
                        antidiffusif = 0.0*antidiffusif
                end if
            end subroutine Calcul_Anti_Diffusif


        !*********************************************************************
        ! Ecriture des fichier "plot_XXX.gnu" de commandes gnuplot
        subroutine scriptgnuplot(sauv)
                integer, intent(in) :: sauv
                integer                         :: i, j, h
                character(len=30)       :: file_name

                open(unit=13,file="./data/plot_centree.gnu")
                open(unit=14,file="./data/plot_decentreegauche.gnu")
                open(unit=15,file="./data/plot_decentreedroite.gnu")
                open(unit=16,file="./data/plot_upwind.gnu")
                open(unit=17,file="./data/plot_laxfriedrichs.gnu")
                open(unit=18,file="./data/plot_laxwendroff.gnu")
                open(unit=19,file="./data/plot_beamwarming.gnu")
                open(unit=20,file="./data/plot_fromm.gnu")
                open(unit=21,file="./data/plot_antidiffusif.gnu")

                do i=13,21
                        write(i,'("set␣yrange[-0.25:1.25];")')
                        h=i-10
                        write(i,'(a58,i2,a6)') "plot␣'transport000.dat'␣u␣1:2␣w␣l,␣'transport000.dat'␣u␣1:",
                            ➥h,"␣w␣lp;"
                        write(i,'("pause␣-1;")')
                        do j=1,sauv
                                h=i-10
                                write(i,'(a15,i3.3,a27,i3.3,a10,i2,a6)') "plot␣'transport",j,".dat'␣u␣1:2␣w␣
                                    ➥l,␣'transport",j,".dat'␣u␣1:",h,"␣w␣lp;"
                                write(i,'("pause␣0.25;")')
                        end do
                        write(i,'("pause␣-1;")')
                        close(i)
                end do

                open(unit=30,file="./data/plot_comparaison_finale.gnu")
                        write(i,'("set␣yrange[-0.25:1.25];")')
                        write(30,'(a18,i3.3,a6)') "fichier='transport",sauv,".dat';"
                        write(30,*) "plot␣fichier␣u␣1:␣2␣w␣l␣ti␣'Exacte'\"
                        !write(30,*) "   , fichier u 1: 3 w lp ti 'Centre'\"
                        !write(30,*) "   , fichier u 1: 4 w lp ti 'Decentre a gauche'\"
                        !write(30,*) "   , fichier u 1: 5 w lp ti 'Decentre a droite'\"
```

```
                        write(30,*) "␣␣␣,␣fichier␣u␣1:␣6␣w␣lp␣ti␣'Upwind'\"
                        write(30,*) "␣␣␣,␣fichier␣u␣1:␣7␣w␣lp␣ti␣'Lax-Friedrichs'\"
                        write(30,*) "␣␣␣,␣fichier␣u␣1:␣8␣w␣lp␣ti␣'Lax-Wendroff'\"
                        write(30,*) "␣␣␣,␣fichier␣u␣1:␣9␣w␣lp␣ti␣'Beam-Warmimg'\"
                        write(30,*) "␣␣␣,␣fichier␣u␣1:10␣w␣lp␣ti␣'Fromm'\"
                        write(30,*) "␣␣␣,␣fichier␣u␣1:11␣w␣lp␣ti␣'Antidiffusif'\"
                        write(30,*) "␣␣␣;"
                        write(30,'("pause␣-1;")')
                        write(30,*) "set␣term␣png"
                        write(30,*) "set␣output␣'comparaison_finale.png'"
                        write(30,*) "rep"
                        close(30)
        end subroutine scriptgnuplot


end program transport
```

Annexe B.

Code Python

```
#
# Copyright G. Faccanoni, Juin 2010
#
# schemas numeriques sur l'equation de transport lineaire
# avec condition aux limites periodiques
# d_t u + a d_x u = 0
#


# packages
import numpy as np
from numpy import *
import matplotlib.pyplot as plt

def dessiner(t,x,u_lf,u_lw,u_up,u_bw,u_fr,u_ad):
        #
        plt.subplot(321)
        plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        plt.plot(x, u_lf, 'b-', label="Lax Friedrichs")
        plt.legend(loc='lower center')
        plt.axis([0.0,1.0,-1.5,1.5])
        plt.title('t='+str(t))
        #
        plt.subplot(322)
        plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        plt.plot(x, u_lw, 'g-', label="Lax Wendroff")
        plt.legend(loc='lower center')
        plt.axis([0.0,1.0,-1.5,1.5])
        plt.title('t='+str(t))
        #
        plt.subplot(323)
        plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        plt.plot(x, u_up, 'y-', label="Upwind")
        plt.legend(loc='lower center')
        plt.axis([0.0,1.0,-1.5,1.5])
        plt.title('t='+str(t))
        #
        plt.subplot(324)
        plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        plt.plot(x, u_bw, 'c-', label="Beam Warming")
        plt.legend(loc='lower center')
        plt.axis([0.0,1.0,-1.5,1.5])
        plt.title('t='+str(t))
        #
        plt.subplot(325)
        plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        plt.plot(x, u_ad, 'g-', label="Anti Diffusif")
        plt.legend(loc='lower center')
        plt.axis([0.0,1.0,-1.5,1.5])
        plt.title('t='+str(t))
        #
        plt.subplot(326)
        plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        plt.plot(x, u_fr, 'm-', label="Fromm")
        plt.legend(loc='lower center')
        plt.axis([0.0,1.0,-1.5,1.5])
        plt.title('t='+str(t))
        #
        plt.draw()
        #plt.plot(x, exacte, 'r-', label="Exacte", linewidth=2)
        #plt.plot(x, u_lf, 'b-', label="Lax Friedrichs")
        #plt.plot(x, u_lw, 'g-', label="Lax Wendroff")
        #plt.plot(x, u_up, 'y-', label="Upwind")
        #plt.plot(x, u_bw, 'c-', label="Beam Warming")
        #plt.plot(x, u_fr, 'm-', label="Fromm")
        #plt.legend()
        #plt.axis([0.0,1.0,-1.5,1.5])
        #plt.title('t='+str(n*dt))
```

```
        #plt.draw()



# macro pour le shift des vecteurs avec conditions periodiques
def shiftp(typ,u):
    m = u.size # nombre d'elements, les indices vont de 0 a m-1 ?
    us = copy(u)
    if typ == 1 :
        us[m-1] = u[0]
        us[0:m-1] = u[1:m]
        return us
    elif typ == 2 :
        us[0] = u[m-1]
        us[1:m] = u[0:m-1]
        return us
    else:
        print "typ non ok"


# Trois types de solutions exactes
def solexacte(x,n,dt):
# SINUS
    #return sin((x-a*n*dt)*2*math.pi)
# CRENAUX
    # sol=np.zeros(nx)
    # for i in range(nx):
        # xi=i*dx-a*n*dt
        # xint = floor(xi/lg)
        # xi = xi - xint*lg
        # if xi < lg/2:
                        # sol[i] = 0.
        # else:
                        # sol[i] = 1.
    # return sol
# DOUBLE CRENAUX, Toro page 210 etc Test2
    sol=np.zeros(nx)
    for i in range(nx):
        xi = (i*dx-a*n*dt) - floor((i*dx-a*n*dt)/lg)*lg
        if xi < lg*0.3:
            sol[i] = 0.
        elif xi < lg*0.7:
            sol[i] = 1.
        else:
            sol[i] = 0.
    return sol



# flux numeriques pour les schemas de
# Lax-Friedrichs,
# Lax-Wendroff,
# Upwind,
# Beam-Warming,
# Fromm

def g_lf(wL,wR):
    return (a*wL+a*wR)*0.5-(wR-wL)*(0.5*dx/dt)

def g_lw(wL,wR):
    return (a*wL+a*wR)*0.5-(0.5*dt/dx)*(a*wR-a*wL)*a

def g_up(wL,wR):
        if a>0:
                return a*wL
        else:
                return a*wR

def g_bw(wL,wR):
        c=a*dt/dx
        return (c-1.0)*0.5*a*wL + (3.0-c)*0.5*a*wR

def g_fr(wL,wC,wR):
        c=a*dt/dx
        return (c-1.0)*0.25*a*wL + wC + (1.0-c)*0.25*a*wR

def g_ad(wL,wC,wR):
        temp=zeros(nx)
        c=a*dt/dx
        for i in range(nx):
                b=max(wL[i],wC[i])+(wC[i]-max(wL[i],wC[i]))/c
                B=min(wL[i],wC[i])+(wC[i]-min(wL[i],wC[i]))/c
                if wR[i]<=b:
                        temp[i]=b
                elif wR[i]>=B:
```

```
                        temp[i]=B
                else:
                        temp[i]=wR[i]
        return a*temp

#-------------------------------------------------------------------------
# INITIALISATION
#-------------------------------------------------------------------------

lg = 1.                 # lg = longueur du domaine
nx = 100                # nx = nombre de mailles
dx = lg/nx              # dx = pas d'espace
a =  1.                 # a = vitesse de transport
nt = 1000               # nt = nombre de pas de temps effectues
cfl = dx/abs(a)
dt = 0.5*cfl                    # dt = pas de temps


n=0
x=np.linspace(0,1,nx)# x = points du maillage
exacte=solexacte(x,n,dt)
u_lf = exacte # u_lf = donnee initiale de LF
u_lw = exacte # u_lw = donnee initiale de LW
u_up = exacte # u_up = donnee initiale de upwind
u_bw = exacte # u_bw = donnee initiale de BW
u_fr = exacte # u_fr = donnee initiale de FR
u_ad = exacte # u_ad = donnee initiale de AD



plt.ion()
plt.figure(1)
dessiner(0,x,u_lf,u_lw,u_up,u_bw,u_fr,u_ad)
raw_input('CI affichee')




#-------------------------------------------------------------------------
# marche en temps
#-------------------------------------------------------------------------

for n in xrange(1,nt+2):

        exacte=solexacte(x,n,dt)

        # LF
        up_lf = shiftp(1,u_lf)
        um_lf = shiftp(2,u_lf)
        v_lf = u_lf - (dt/dx)*(g_lf(u_lf,up_lf)-g_lf(um_lf,u_lf)) # Lax Friedrichs
        u_lf = v_lf

        # LW
        up_lw = shiftp(1,u_lw)
        um_lw = shiftp(2,u_lw)
        v_lw = u_lw - (dt/dx)*(g_lw(u_lw,up_lw)-g_lw(um_lw,u_lw))  # Lax Wendroff
        u_lw = v_lw

        # UPWIND
        up_up = shiftp(1,u_up)
        um_up = shiftp(2,u_up)
        v_up = u_up - (dt/dx)*(g_up(u_up,up_up)-g_up(um_up,u_up))  # Upwind
        u_up = v_up

        # BW
        if a>0:
                um_bw = shiftp(2,u_bw)
                umm_bw = shiftp(2,um_bw)
                v_bw = u_bw - (dt/dx)*(g_bw(um_bw,u_bw)-g_bw(umm_bw,um_bw))  # Beam-Warming
                u_bw = v_bw
        else:
                print 'BW : cas non considere'

        # FR
        if a>0:
                up_fr = shiftp(1,u_fr)
                um_fr = shiftp(2,u_fr)
                umm_fr = shiftp(2,um_fr)
                v_fr = u_fr - (dt/dx)*(g_fr(um_fr,u_fr,up_fr)-g_fr(umm_fr,um_fr,u_fr))  # Fromm
                u_fr = v_fr
        else:
                print 'FR : cas non considere'

        # ANTI DIFFUSIF
        if a>0:
                up_ad = shiftp(1,u_ad)
```

```
                um_ad = shiftp(2,u_ad)
                umm_ad = shiftp(2,um_ad)
                v_ad = u_ad - (dt/dx)*(g_ad(um_ad,u_ad,up_ad)-g_ad(umm_ad,um_ad,u_ad))  # Antidiff
                u_ad = v_ad
        else:
                print 'AD : cas non considere'

        if ((n==nt+2)|(n%15==0)) :
                plt.clf()
                dessiner(n*dt,x,u_lf,u_lw,u_up,u_bw,u_fr,u_ad)

raw_input('Marche en temps finie')
# pylab.waitforbuttonpres
```