Équations aux dérivées partielles – séance TP

Le but de cette séance est d'implementer numériquement à l'aide du logiciel `Octave` ou `Matlab` quelques schémas numeriques pour l'équation d'advection, de tester leurs propriétés de stabilité et représenter graphiquement l'évolution de la solution au cours des itérations en temps.

Introduction. On considère le problème modèle

$$\begin{cases} \dfrac{\partial u}{\partial t}(x,t) + V\dfrac{\partial u}{\partial x}(x,t) = 0, x \in \mathbb{R}, x \in [0,L], t \geq 0, \\ u(x,0) = u_0(x),\ x \in [0,L], \\ u(0,t) = u(L,t),\ t \geq 0. \end{cases}$$

Pour simplifier on a supposée que la donnée initale est périodique de période L, si bien que nous pouvons nous restreindre à l'intervalle $[0,L]$. On vérifie aisément que la solution exacte est donnée par $u(x,t) = u_0(x - Vt)$.

On cherche à approcher numériquement la solution par le schéma *décentré amont* suivant:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + V\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0.$$

où $u_i^n \cong u(x_i, t^n)$, $1 \leq i \leq N$ et $\Delta x = L/(n-1)$, $x_i = (i-1)\Delta x$ et $t^n = n\Delta t$. En réarrangeant les termes on obtient:

$$u_i^{n+1} = u_i^n - \sigma(u_i^n - u_{i-1}^n)$$

où $\sigma = \dfrac{V\Delta t}{\Delta x}$ est connu sous le nom de *nombre de Courant.*
Pour approcher la condition initiale et la condition limite on écrit:

$$u_i^0 = u_0(x_i),\ 1 \leq i \leq N,\ u_1^n = u_N^n,\ n \geq 0.$$

Premier script Matlab ou Octave On écrira un script `test1.m` qui comprendra 4 parties:

1. Initialisation des paramètres physiques: vitesse de propagation, longueur du domaine, temps maximum et condition initiale. La condition initiale sera une fonction en créneau:

$$u_0(x) = \begin{cases} 1,\ 1 < x < 1.5, \\ 0, \text{ sinon.} \end{cases}$$

2. Initialisation des paramètres numériques. On prendra $N = 201$ et on déduira Δx, puis on choisira σ et on déduira Δt.

3. Boucle en temps: initaliser u_i^0, $i = 1, ..., N$, puis évaluer u_i^{n+1}, $i = 1, ..., N$ en fonction de u_i^n, $i = 1, ..., N$ pour $t^n < T$.

4. Visualisation graphique des résultats: afficher sur la même fenêtre la solution exacte et la solution approchée.

Indications. Initialiser le script par la commande `clear all, close all, clc` qui nettoie l'espace mémoire occupé avant le lancement du script, ferme toutes les fenêtres graphiques et positionne le curseur tout en haut de l'espace d'execution.

1. Paramètres physiques:

   ```
   V = 0.1;   % vitesse d'advection
   Tmax = 10; %  temps maximum
   L = 5;     % longueur du domaine
   ```

   La condition initiale sera définie à l'aide d'une function `inline` (sur une seule ligne)

   ```
   condinit = @(x) (x>1.) & (x<1.5);
   ```

2. Paramètres numériques:

   ```
   N = 201;      % nb de points de discretisation en espace
   dx = L/(N-1); % pas d'espace
   s = 0.8;      % nombre de Courant
   dt = s*dx/V;  % pas de temps respectant la condition V*dt/dx = nombre de CFL
   ```

   Initialisation du maillage et de la donnée initiale:

   ```
   x = linspace(0,L,N)';
   u = double(condinit(x)); % calcul de la solution initale + conversion booleen reel
   ```

3. Boucle en temps: à chaque pas de temps on met à jour la solution de façon vectorielle en tenant compte de la condition de périodicité:

   ```
   tps = dt:dt:Tmax;
   for t = tps
      uold = u;
      u(2:N) = uold(2:N)-s*(uold(2:N)-uold(1:N-1));
      u(1)=u(N); % condition de périodicité
   end
   ```

4. Sortie graphique: on pourra par exemple utiliser les commandes suivantes:

   ```
   uexact = condinit(x-V*t); % solution exacte au pas de temps final
   plot(x,u,'rx-',x,uexact,'bo-'), grid on
   title('Comparaison entre la solution exacte et approchee')
   legend('Sol. approchee','Sol. exacte')
   xlabel('x')
   ylabel('Solution')
   ```

Afin de mesurer le temps d'exécution du programme on insérera la commande `tic` avant le démarrage de la boucle en temps et `toc` en dernière ligne du script.

Question: Augmenter progressivement le paramètre σ et observer le résultat. Quelle est la valeur critique? Constater aussi en augmentant progressivement Tmax que la solution numérique s'aplatit au fil des itérations en temps. Ceci est le phénomène de diffusion numérique.

Deuxième script On souhaite également pouvoir considérer la condition initiale suivante (contrairement a la condition initiale précédente, il s'agit d'une fonction beaucoup plus régulière):

$$u_0(x) = \sin(8\pi x/L)$$

et avoir le choix entre 2 schémas numériques: le schéma *décentré amont* et le schéma de *Lax-Wendroff*:

$$u_i^{n+1} = u_i^n - \frac{\sigma}{2}(u_{i+1}^n - u_{i-1}^n) + \frac{\sigma^2}{2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

Écrire un nouveau script qu'on appelera `test2.m` et qui offre ces 2 options.

`Indication`: Dans ce nouveau script au tout début, à l'intérieur de la boucle en temps on va fixer la condition à un des bords (condition entrante)

```
uexact = condinit(x-V*t);
u(N) = uexact(N);
```

Question: Repartir de $\sigma = 0.8$ et essayer 4 possibilités du couple conditions initiale/schéma numérique.

- Quelles conditions tirez-vous?
- Augmenter progressivement σ pour le schéma de Lax-Wendroff et observer.
- Que peut-on dire de la diffusion numérique observée précédemment dans le cas du schéma décentré?
- Est-ce que le schéma de Lax-Wendroff est diffusif?

Troisième script: animation graphique. L'objectif de cette section est de pouvoir visualiser la solution numérique au cours de itérations au lieu de visualiser que la solution finale. On réalisera un nouveau script appelé `test3.m`

Indications Au sein de la boucle en temps, la fenêtre graphique est rafraichie de la façon suivante:

```
uexact =...;
plot(...), grid on
legend(...)
title(...)
pause
```

La commande `pause` arrête l'exécution du programme jusqu'à ce qu'on utilise de nouveau le clavier. Il y a aussi l'option `pause(n)` ou n est le laps de temps (en secondes) pendant lequel le graphique va rester a l'écran.

Question: Reprendre les essais précédents puis augmenter progressivement le paramètre σ.

- Prendre $\sigma = 1.2, 1.5, 1.8$ pour le schéma de Lax-Wendroff). Quelles conclusions en tirer?
- Est-ce que le schéma de Lax-Wendroff se comporte différemment en fonction de la régularité de la condition initiale?

Quatrième script: un schéma implicite. Nous allons modifier le schéma de Lax-Wendroff en "implicitant" le terme de diffusion de la façon suivante:

$$u_i^{n+1} = u_i^n - \frac{\sigma}{2}(u_{i+1}^n - u_{i-1}^n) + \frac{\sigma^2}{2}(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1})$$

En introduisant $W^n = (u_i^n - \frac{\sigma}{2}(u_{i+1}^n - u_{i-1}^n))_{0\leq i\leq N-1}$ et $U^{n+1} = (u_i^{n+1})_{0\leq i\leq N-1}$ ainsi que la matrice A d'ordre $N-1$ donnée par:

$$A = \begin{pmatrix} 1+\sigma^2 & -\sigma^2/2 & 0 & \dots & 0 & -\sigma^2/2 \\ -\sigma^2/2 & 1+\sigma^2 & -\sigma^2/2 & 0 & \dots & 0 \\ 0 & & & & & \vdots \\ \vdots & & & & & 0 \\ 0 & \dots & 0 & -\sigma^2/2 & 1+\sigma^2 & -\sigma^2/2 \\ -\sigma^2/2 & 0 & \dots & 0 & -\sigma^2/2 & 1+\sigma^2 \end{pmatrix}$$

le schéma s'écrit comme $AU^{n+1} = W^n$. On notera que la condition de périodicité est utilisée directement dans le schéma numérique afin d'éliminer u_i^N.
On constate que le schéma est *implicite*: l'évaluation de U^{n+1} à partir de U^n nécessite l'inversion d'un système linéaire, le coût de l'itération sera donc plus élévé que dans le cas d'un schéma explicite. Cependant le schéma n'est pas limité par la valeur du nombre de Courant σ ce qui permet l'utilisation des pas de temps plus grands et donc réaliser moins d'itérations en temps.

On va créer un nouveau script appelé `test4.sce` à partir du `test3.sce` dont on enlevera les parties relatives au schéma numérique.

Indications On va procédér de la façon suivante

1. Initialisation de la matrice A:

```
A = diag((1+s^2)*ones(N,1))-diag(s^2/2*ones(N-1,1),-1)-...
    diag(s^2/2*ones(N-1,1),1);
A(1,N) = -s^2/2; % on modifie la premiere ligne et la derniere
A(N,1) = -s^2/2; % afin de prendre en compte la periodicite
```

2. Deux options sont possibles pour la boucle en temps: résoudre le système linéaire à chaque pas de temps:

```
u = A\Wn;                          % a faire a chaque iteration en temps
```

   ou inverser (factoriser) la matrice A une fois pour toutes avant les itérations en temps puis utiliser son inverse (ou factorisation):

```
B = inv(A);//ou [L,U,P]= lu(A); // inversion ou factorisation LU de A
tps = dt:dt:Tmax;
for t = tps
...
u =B*Wn                            % ou u = U\(L\(P*Wn));
```

   Il est plutôt conseillé d'utiliser la factorisation LU au lieu de l'inverse de la matrice.

Question: Comparer les 3 différentes options pour l'inversion du système linéaire en calculant le temps d'execution et en agrandissant progréssivement la taille du système à résoudre. Réaliser des expériences numériques en faisant varier le nombre de Courant. Discuter les performances relatives des schémas implicite et explicite en terme de temps de calcul (pour mesurer le temps d'un bloc à l'intérieur d'un script on peut utiliser les commandes `tic` et `toc`)