Option B : cours du 5 janvier 2021.

Rappel : on a étudié les équations de transport :

$$(\mathcal{E}) \begin{cases} \partial_t u + a(t,x)\,\partial_x u = 0, & t \in \mathbb{R},\ x \in \mathbb{R}, \\ u(0,x) = u^0(x), & x \in \mathbb{R}. \end{cases}$$

Si u^0 est de classe C^1 et si a est continue, et lipschitzienne par rapport à sa seconde variable (x), le pb de Cauchy admet une unique solution : $u(t,x) = u^0\big(X(0,t,x)\big)$ où les courbes X sont les solutions de

$$\begin{cases} \partial_s X(s,t,x) = a\big(s, X(s,t,x)\big), & s \in \mathbb{R} \\ X(t,t,x) = x \end{cases}$$

Dans le cas où $a(t,x) = a \in \mathbb{R}$, on $u(t,x) = u^0(x - at)$.

Aujourd'hui nous allons calculer des solutions approchées de $(\mathcal{E})$.

Dans le cas $a(t,x) = a$, on n'en a pas besoin (car on connaît la solution exacte), mais commencer dans ce cas simple est un bon moyen de se faire la main.

L'idée pour approcher u est de calculer des valeurs approchées de u en certains points (de $\mathbb{R}_t^+ \times \mathbb{R}_x$).

Soit $\Delta x > 0$ et soit $\Delta t > 0$. Posons $t^n = n\Delta t$ pour $n \in \mathbb{N}$ et $x_j = j\Delta x$ pour $j \in \mathbb{Z}$.

Si u est une solution régulière (C^1 suffit), on a

$$(1) \quad \frac{u(t^{n+1}, x_j) - u(t^n, x_j)}{\Delta t} = \partial_t u(t^n, x_j) + o(\Delta t)$$

$$(2) \quad \frac{u(t^n, x_j) - u(t^{n-1}, x_j)}{\Delta t} = \partial_t u(t^n, x_j) + o(\Delta t)$$

(et on peut écrire une infinité d'approx. du même type)

et

$$(1\text{ bis}) \quad \frac{u(t^n, x_j) - u(t^n, x_{j-1})}{\Delta x} = \partial_x u(t^n, x_j) + o(\Delta x)$$

$$(2\text{ bis}) \quad \frac{u(t^n, x_{j+1}) - u(t^n, x_j)}{\Delta x} = \partial_x u(t^n, x_j) + o(\Delta x)$$

$(\dots)$

Donc l'EDP $\partial_t u + a \partial_x u = 0 \quad (a \in \mathbb{R})$

permet de savoir que (par exemple)

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

$$= \partial_t u(t^n, x_j) + a \partial_x u(t^n, x_j) + o(\Delta t) + o(\Delta x) = o(\Delta t) + o(\Delta x)$$

(j'ai pris les formules (1) et (1bis)).

L'idée des schémas numériques est de calculer des valeurs approchées u_j^n de $u(t^n, x_j)$ $(n \in \mathbb{N}, j \in \mathbb{Z})$ satisfaisant à

$$(U) \quad \frac{u_j^{n+1} - u_j^n}{\Delta t} + a \frac{u_j^n - u_{j-1}^n}{\Delta x} = 0, \quad n \in \mathbb{N}, j \in \mathbb{Z},$$

en espérant que u_j^n sera proche de $u(t^n, x_j)$. Il y a bien sûr une infinité de schémas numériques (naturels) possibles.

(U) se réécrit $u_j^{n+1} = u_j^n - \frac{a\Delta t}{\Delta x}(u_j^n - u_{j-1}^n)$, $n \in \mathbb{N}$, $j \in \mathbb{Z}$.

Donc si on pose $u_j^0 = u^0(x_j)$ (c'est très naturel !),

on est capables de calculer tous les u_j^n pour $n \in \mathbb{N}$ et $j \in \mathbb{Z}$.

Remarquez qu'avec les formules (2) et (1bis) le schéma serait

$$\begin{cases} \frac{u_j^n - u_j^{n-1}}{\Delta t} + a \frac{u_j^n - u_{j-1}^n}{\Delta x} = 0, & n \geq 1, j \in \mathbb{Z}, \\ u_j^0 = u^0(x_j), & j \in \mathbb{Z}. \end{cases}$$

Il n'est pas du tout évident a priori que cette formule admet une solution !

$\frac{u_j^1 - u_j^0}{\Delta t} + a \frac{u_j^1 - u_{j-1}^1}{\Delta x} = 0$ n'est pas une équation triviale.

(Il s'agit de résoudre un système linéaire du type $AU^1 = U^0$

où $A = \begin{pmatrix} \ddots & & 0 \\ & 1 + \frac{a\Delta t}{\Delta x} & \\ & -\frac{a\Delta t}{\Delta x} \quad 1 + \frac{a\Delta t}{\Delta x} & \\ & -\frac{a\Delta t}{\Delta x} & \\ 0 & & \ddots \end{pmatrix}$: A est-il un opérateur inversible ?

On pourrait montrer que c'est le cas).

(Fin de la remarque).

Nous allons démontrer que (U) est un schéma convergeant ($(u_j^n) \xrightarrow[\Delta t \to 0]{\Delta x \to 0} (u(t^n, x_j))$ en un certain sens et sous certaines conditions).

1) Consistance du schéma

$$(U)\quad \begin{cases} \frac{u_j^{n+1} - u_j^n}{\Delta t} + a \frac{u_j^n - u_{j-1}^n}{\Delta x} = 0, & n \in \mathbb{N}, j \in \mathbb{Z} \\ u_j^0 = u^0(x_j), & j \in \mathbb{Z} \end{cases}$$

Déf On appelle erreur de consistance $(\varepsilon(t^n, x_j))$ du schéma (U) le réel

$$\varepsilon_j^n = \frac{u(t^{n+1}, x_j) - u(t^n, x_j)}{\Delta t} + a \frac{u(t^n, x_j) - u(t^n, x_{j-1})}{\Delta x}.$$

Attention : $\varepsilon_j^n \neq 0$ ($u(t^n, x_j) \neq u_j^n$!).

D'autre part ε_j^n dépend de u, la solution qu'on tâche d'approcher (il faudrait écrire $\varepsilon_j^n(u)$ et non ε_j^n).

Lemme Si $u \in C_b^2(\mathbb{R}_+, \mathbb{R})$, il existe $C \in \mathbb{R}$ t.q.

$$|\varepsilon_j^n| \leq C(\Delta t + \Delta x)$$

Déf $C_b^2(\mathbb{R}_+, \mathbb{R})$ est l'ensemble des fonctions de $C^2(\mathbb{R}_+ \times \mathbb{R})$ à dérivées secondes dans $L^\infty(\mathbb{R}_+ \times \mathbb{R})$.

Démo On a, puisque $u \in C^2(\mathbb{R}_+ \times \mathbb{R})$,

$$u(t^{n+1}, x_j) = u(t^n, x_j) + \Delta t\, \partial_t u(t^n, x_j) + \frac{\Delta t^2}{2} \partial_{t,t}^2 u(\overline{t^n}, x_j)$$

pour un certain $\overline{t^n} \in [t^n, t^{n+1}]$.

Donc $\frac{u(t^{n+1}, x_j) - u(t^n, x_j)}{\Delta t} = \partial_t u(t^n, x_j) + \frac{\Delta t}{2} \partial_{t,t}^2 u(\overline{t^n}, x_j)$.

De même, on a

$$\frac{u(t^n, x_j) - u(t^n, x_{j-1})}{\Delta x} = \partial_x u(t^n, x_j) - \frac{\Delta x}{2} \partial_{x,x}^2 u(t^n, \overline{x_j})$$

pour un certain $\overline{x_j} \in [x_{j-1}, x_j]$

(en effet $u(t^n, x_{j-1}) = u(t^n, x_j) - \Delta x\, \partial_x u(t^n, x_j) + \frac{\Delta x^2}{2} \partial_{x,x}^2 u(t^n, \overline{x_j})$ $\Leftarrow$)

(Remarque : ici on quantifié les formules (1) et (1bis)).

Donc :

$$\varepsilon_j^n = \underbrace{\partial_t u(t^n, x_j) + a\, \partial_x u(t^n, x_j)}_{0 \text{ car } u \text{ est solution de } \partial_t u(t,x) + a\, \partial_x u(t,x) = 0 \ \forall t,x.} + \frac{\Delta t}{2} \partial_{t,t}^2 u(\overline{t^n}, x_j) - a \frac{\Delta x}{2} \partial_{x,x}^2 u(t^n, \overline{x_j})$$

(Remarque : il est important d'avoir fait les développements limités aux bons points : il faut avoir $\partial_t u$ et $\partial_x u$ en le même point).

Finalement $|\varepsilon_j^n| = \left| \frac{\Delta t}{2} \partial^2_{t,t} u(\overline{t}^n, x_j) - \frac{\Delta x}{2} \partial^2_{x,x} u(t^n, \overline{x}_j) \right|$

$\leq \frac{\Delta t}{2} \left| \partial^2_{t,t} u(\overline{t}^n, x_j) \right| + \frac{\Delta x}{2} \left| \partial^2_{x,x} u(t^n, \overline{x}_j) \right|$.

Comme $u \in C^2_b(\mathbb{R}_+, \mathbb{R})$, $\exists \tilde{C} \in \mathbb{R}$ / $\left| \partial^2_{t,t} u(t,x) \right| \leq \tilde{C}$ $\forall t,x$

et $\left| \partial^2_{x,x} u(t,x) \right| \leq \tilde{C}$ $\forall t,x$.

Donc $|\varepsilon_j^n| \leq \tilde{C} \left(\frac{\Delta t}{2} + \frac{\Delta x}{2} \right)$: CQFD avec $C = \frac{\tilde{C}}{2}$.

Ceci ne montre pas la convergence du schéma !

Ce n'est qu'une étape qu'on utilisera plus loin.

2) Stabilité et convergence

Pour montrer la convergence du schéma, on va majorer

$|e_j^n|$ où $e_j^n = u(t^n, x_j) - u_j^n$, $n \in \mathbb{N}$, $j \in \mathbb{Z}$.

Pour majorer e_j^n on a envie de trouver une formule d'évolution de cette erreur (évolution en temps, en n) un peu comme u_j^{n+1} est fonction des u_k^n ...

$$e_j^{n+1} = u(t^{n+1}, x_j) - u_j^{n+1} = u(t^{n+1}, x_j) - \left[u_j^n - \frac{a\Delta t}{\Delta x} (u_j^n - u_{j-1}^n) \right]$$

$$= u(t^n, x_j) - \left[u_j^n - \frac{a\Delta t}{\Delta x} (u_j^n - u_{j-1}^n) \right] + u(t^{n+1}, x_j) - u(t^n, x_j)$$

$$= u(t^n, x_j) - \left[u_j^n - \frac{a\Delta t}{\Delta x} (u_j^n - u_{j-1}^n) \right] + \Delta t \, \varepsilon_j^n - \frac{a\Delta t}{\Delta x} \left(u(t^n, x_j) - u(t^n, x_{j-1}) \right)$$

(cf. la définition de ε_j^n ← c'est pour ça qu'on a défini ε_j^n ainsi !)

$$= e_j^n - \frac{a\Delta t}{\Delta x} \left(e_j^n - e_{j-1}^n \right) + \Delta t \, \varepsilon_j^n$$

Remarquer que l'erreur suit une éq. d'évolution discrète qui la même que la solution approchée (discrète) à un terme source près qui est $\Delta t \, \varepsilon_j^n$.

On a vu que ε_j^n était petit (si u est régulier) : maintenant il s'agit de montrer que ces petits termes sources ε_j^n ne sont pas (trop) amplifiés par le schéma (on parle alors de stabilité du schéma).

Remarquer que $e_j^0 = u(0, x_j) - u_j^0 = u^0(x_j) - u^0(x_j) = 0$.

$$|e_j^{n+1}| = \left| e_j^n - \frac{a\Delta t}{\Delta x} (e_j^n - e_{j-1}^n) + \Delta t \, \varepsilon_j^n \right|$$

$$\leq |e_j^n| \left| 1 - \frac{a\Delta t}{\Delta x} \right| + \left| \frac{a\Delta t}{\Delta x} \right| |e_{j-1}^n| + \Delta t |\varepsilon_j^n|$$

Supposons que $a \geq 0$: alors $\left|\frac{a\Delta t}{\Delta x}\right| = \frac{a\Delta t}{\Delta x}$.

Supposons que $1 - \frac{a\Delta t}{\Delta x} \geq 0$: alors $\left|1 - \frac{a\Delta t}{\Delta x}\right| = 1 - \frac{a\Delta t}{\Delta x}$.

Sous ces deux hypothèses,

$$|e_j^{n+1}| \leq |e_j^n|\left(1 - \frac{a\Delta t}{\Delta x}\right) + |e_{j-1}^n| \frac{a\Delta t}{\Delta x} + \Delta t |\epsilon_j^n|$$

$$\leq \sup_{k\in\mathbb{Z}} |e_k^n| \left(1 - \frac{a\Delta t}{\Delta x} + \frac{a\Delta t}{\Delta x}\right) + \Delta t |\epsilon_j^n| :$$

$$|e_j^{n+1}| \leq \sup_{k\in\mathbb{Z}} |e_k^n| + \Delta t |\epsilon_j^n|, \text{ donc}$$

$$\sup_k |e_k^{n+1}| \leq \sup_k |e_k^n| + C\Delta t (\Delta t + \Delta x)$$

(si $u \in C_b^2(\mathbb{R}_+, \mathbb{R})$, cf. lemme)

$$\leq \sup_k (e_k^{n-1}) + 2C\Delta t(\Delta t + \Delta x)$$

$$\leq \ldots \leq \sup_k (e_k^0) + (n+1) C\Delta t (\Delta t + \Delta x).$$

Or $e_k^0 = 0$ $\forall k$. Ainsi

$$\sup_k |e_k^{n+1}| \leq C t^{n+1} (\Delta t + \Delta x).$$

Th Supposons $u \in C_b^2(\mathbb{R}_+, \mathbb{R})$. Supposons $a \geq 0$.
Supposons que $\frac{a\Delta t}{\Delta x} \leq 1$.
Alors, il existe $C \in \mathbb{R}$ t.q.

$$\sup_k |e_k^n| \leq C t^n (\Delta t + \Delta x) \qquad \forall n \in \mathbb{N}.$$

(On vient de le démontrer !).

Remarques . le résultat s'écrit aussi $\|e^n\|_\infty \leq C t^n (\Delta t + \Delta x)$
où $e^n = (e_k^n)_{k\in\mathbb{Z}}$.

. le résultat exprime qu'en temps fixé (fini)
l'erreur est majorée par une constante fois $(\Delta t + \Delta x)$:
on dit que le schéma est convergent à l'ordre 1 en Δt
et 1 en Δx ($\Delta t^1 + \Delta x^1$)

. L'hypothèse $a \geq 0$ est absolument nécessaire. Pour $a < 0$
il faudrait utiliser les formules (1) et (2bis) pour finalement
obtenir le schéma $\frac{u_j^{n+1} - u_j^n}{\Delta t} + a \frac{u_{j+1}^n - u_j^n}{\Delta x} = 0$ (exercice :
montrer que ce schéma est convergent si $a \leq 0$!).

• L'hypothèse $a\Delta t \leq \Delta x$ est absolument nécessaire.
Elle s'appelle condition de Courant-Friedrichs-Lewy.

• Revenons sur le signe de a ...
* Si a est positif, $u_j^{n+1} = u_j^n - \frac{a\Delta t}{\Delta x}(u_j^n - u_{j-1}^n)$
est convergent (ce qu'on a démontré)
* si est négatif, $u_j^{n+1} = u_j^n - \frac{a\Delta t}{\Delta x}(u_{j+1}^n - u_j^n)$
est convergent (exercice ci-dessus).

Dans les 2 cas la convergence du schéma est conséquence de la stabilité qui découle directement du fait que u_j^{n+1} est une combinaison convexe des u_k^n :

Ces combinaisons convexes ont été essentielles dans les majorations de $|e_j^{n+1}|$: d'ailleurs l'éq. d'évolution de e_j^n est presque identique à celle de u_j^n

• $a>0$: $u_j^{n+1} = u_j^n\left(1 - \frac{a\Delta t}{\Delta x}\right) + u_{j-1}^n \frac{a\Delta t}{\Delta x}$
≥ 0 pour la condition de CFL ; ≥ 0
somme de ces 2 coeff $= 1$

• $a<0$: $u_j^{n+1} = u_j^n\left(1 + \frac{a\Delta t}{\Delta x}\right) + u_{j+1}^n\left(-\frac{a\Delta t}{\Delta x}\right)$
≥ 0 sous la condition de CFL $-\frac{a\Delta t}{\Delta x} \leq 1$; ≥ 0

Le schéma $u_j^{n+1} = u_j^n - \frac{a\Delta t}{\Delta x}\begin{cases} u_j^n - u_{j-1}^n & \text{si } a \geqslant 0 \\ u_{j+1}^n - u_j^n & \text{si } a \leqslant 0 \end{cases}$

est appelé SCHÉMA UPWIND (décentré amont) → en remontant le courant.
car : si $a>0$ il est décalé vers la gauche
si $a<0$ ——— droite
} : décalé vers l'amont dans les 2 cas.

Avec le schéma décentré à gauche $\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_j^n - u_{j-1}^n}{\Delta x} = 0$,
u_j^{n+1} est fonction de u_j^n et u_{j-1}^n : u_j^{n+1} dépend de valeurs u_k^n
pour $k \leqslant j$, donc de valeurs de la solution approchée à l'étape en temps n
à gauche du point x_j : c'est comme pour la solution exacte
si $a>0$ ($u(t^{n+1}, x_j) = u(t^n, \underbrace{x_j - a\Delta t}_{\text{gauche de } x_j \text{ si } a>0})$ (formule des caractéristiques))

En particulier si $a<0$ u_j^{n+1} dépend des u_k^n pour $k \leq j$ (schéma décentré à gauche)

mais $u(t^{n+1}, x_j)$ dépend d'une valeur $u(t^n, \cdot)$ à droite de x_j.

Si $a<0$ il paraît donc impossible intuitivement que u_j^{n+1} soit proche $u(t^{n+1}, x_j)$. D'ailleurs la démo de convergence ne marche pas (pas de combinaison convexe).

Application numérique, mise en œuvre.

Schéma proposé par Thomas Lepoutre :

Ouvrir — cod... ~/Ens... — Enregistrer

```
1  import numpy as np
2  import numpy.linalg as npl
3  import matplotlib.pyplot as plt
4
5  # Variations sur les schémas,
6  J=100;
7  dx=1/J;
8  X=np.arange(0,J)*dx
9
10 T=1/10; #horizon temporel
11
12 #donnée initiale
13 U0=np.sin(np.pi*X);
14 V=1;
15 t=0;
16 U=U0;
17 Ugauche=np.zeros(J);
18 CFL=0.5;# évidemment modifiable.
19 dt=CFL*dx
20 while t<T:
21     Ugauche[0]=U[-1]; #U j-1
22     Ugauche[1:]=U[0:J-1]; # Attention aux indices ici!!! 1:J = 1 ... J-1;
23     U = U - V*dt/dx*(U-Ugauche);
24     t = t + dt;
25 plt.figure()
26 plt.plot(X,U)
27 plt.show()
```

Python — Largeur des tabulations : 8 — Lig 27, Col 11 — INS

importation de bibliothèques → pas nécessaire ici

nombre de points en espace

Δx

vecteurs des points du maillage en espace

temps final du calcul

$\sin(\pi x)$

vitesse : a dans le cours

temps de départ

coefficient $\frac{a\Delta t}{\Delta x}$ (doit être inférieur à 1)

Δt : en réalité $\Delta t = \frac{a\Delta t}{\Delta x} \times \frac{\Delta x}{a}$, donc il faudrait dt = CFL*dx/a

En python U[-1] = U[J-1] : ceci simule des conditions périodiques

À chaque étape en temps on écrase la solution à l'étape précédente.

→ On peut pas simuler l'équation $\partial_t u + a\partial_x u = 0$ sur tout $\mathbb{R}_+ \times \mathbb{R}$.

On ne considère que l'intervalle $[0,T]$ en temps, et pour l'espace on se restreint à l'intervalle $[0,1]$ qu'on munit de conditions périodiques

(finalement ceci revient à étudier $\partial_t u + a\partial_x u = 0$, $t \in [0,T]$, $x \in \mathbb{T}$)

→ Pourrait être remplacé par une boucle sur l'indice d'espace

```
for j in range(J):
    U[j] = U[j] - V*dt/dx*(U[j] - U[j-1])
```

← c'est une erreur classique

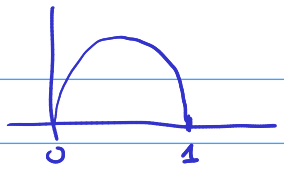
Attention pas ça ! ↑ car ceci correspond au schéma

$$u_j^{n+1} = u_j^n - \frac{a\Delta t}{\Delta x}\left(u_j^n - u_{j-1}^{n+1}\right)$$

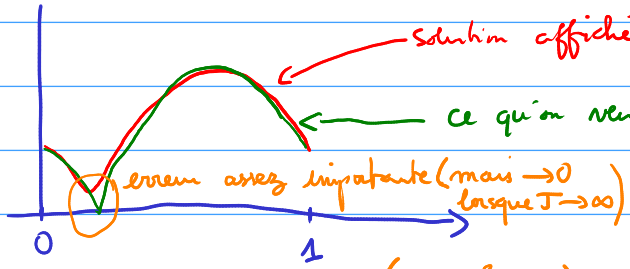
À remplacer par →

```
Ubis[j] = U[j] - V*dt/dx*(U[j] - U[j-1])   (fin de la boucle)
U = Ubis
```

En faisant tourner ce code avec la donnée $\sin(\pi x)$



on voit à l'écran (pour le tps final $\frac{1}{10}$)



Remarque : $\sin(\pi x)$ périodisé $\notin C^2_b(\mathbb{R})$ donc l'analyse qu'on a faite tombe. En fait on pourrait montrer quand même la convergence.

Il est peut-être plus logique de prendre comme donnée initiale $\sin(2\pi x)$

Code plus complet (et plus compliqué)

Ouvrir ▾ | tran... ~/Ens... | Enregistrer

```
# -*- coding: utf-8 -*-

import numpy as np # Bibliothèque de calcul.
import matplotlib.pyplot as plt # Bibliothèque pour la représentation graphique.

def u0(x): # Donnée de Cauchy.
#    return np.exp(-50*(x-0.5)*(x-0.5)) # Une gaussienne.
#    return 0.*x # 0... de la taille de x.
#    return ((x>0.3)*(x<0.7)).astype(float) # Fonction indicatrice de ]0.3, 0.7[.
    return np.sin(2.*np.pi*x)

L = 1. # Longueur du segment en espace (le segment sera [0, L]).
J = 100 # Nombre de points en espace (ou nombre de mailles).
T = 1. # Temps final pour le calcul.
C = 0.7 # Nombre de Courant (associé à la condition de Courant-Friedrichs-Lewy).
a = -1 # Vitesse.

dx = 1./J # Distance entre deux points du maillage !
X = dx*np.arange(J) + 0.5*dx # Vecteur des positions des points du maillage
# Ici j'ai décidé de prendre pour points du maillage les centres des J mailles
# de même longueur découpant le segment [0, L]. D'autres choix sont possibles !
u = u0(X) # On initialise u avec la donnée de Cauchy.
uG = np.zeros(J) # Vecteur des flux numériques à GAUCHE de chaque maille.
uD = np.zeros(J) # Vecteur des flux numériques à DROITE de chaque maille.
n = 0 # Numéro d'itération en temps, initialisé à 0.
t = 0. # Temps initialisé à 0.
dt = C*dx/abs(a) # Pas de temps. Si la vitesse dépendait du temps il faudrait
# recalculer un pas de temps autorisé (satisfaisant à la condition
# de Courant-Friedrichs-Lewy) à chaque itération, dans la boucle en temps.
# C'est la raison pour laquelle je préfère que la boucle en temps
# soit de type « while ».

ymin = min(u)
ymax = max(u) # Deux paramètres pour calibrer l'affichage de la solution.

figtps = plt.figure("Solution au cours du temps",figsize=(14,8)) # Affichage...
figsol, = plt.plot(X,u,'.-')
figsol.set_ydata(u)
plt.ylim(ymin-0.1, ymax+0.1)
plt.grid() # Cette instruction permet de dessiner une grille et d'avoir une
# meilleure appréhension des valeurs prises par la solution
# approchée : je vous conseille de l'utiliser systématiquement.
plt.show(block = False) # Permet d'afficher sans avoir
# à « tuer » la fenêtre pour continuer.
input() # Cette instruction permet de ne lancer le calcul que
# lorsque l'utilisateur appuie sur une touche.

while t<T: # Boucle en temps : ça commence !
    dt = min(dt,T-t) # Cette instruction permet d'assurer qu'on ne
    # dépassera pas le temps final T demandé.
    t = t + dt
    n = n + 1
    if a > 0:
        uG[1:] = u[:J-1] # Décentrement à gauche.
        uG[0] = u[J-1] # Condition périodique.
    else:
        uG = u # décentrement à droite.
    uD[0:J-1] = uG[1:] # Calcul des flux à DROITE (ce sont les mêmes,
    # décalés d'un indice, bien sûr !).
    uD[J-1] = uG[0] # Condition périodique.

    u = u - a*dt/dx*(uD - uG) # Évolution du vecteur des valeurs approchées
    # u : on décide d'écraser l'ancienne valeur pour ne pas utiliser trop de mémoire.

    ymin = min(u)
    ymax = max(u) # Deux paramètres pour calibrer l'affichage de la solution
    # (à chaque pas de temps, donc).

    figsol.set_ydata(u) # Affichage...
    plt.grid()
    plt.ylim(ymin-0.1, ymax+0.1)
    plt.grid()
    plt.pause(dt)

    print ("itération ", n)
    print("t = ", t) # On affiche le numéro d'itération et l'instant courant.

input()

figfin = plt.figure("Solution au temps T",figsize=(14,8)) # Affichage...
plt.plot(X,u,'.-')
plt.ylim(ymin-0.1, ymax+0.1)
plt.grid()
plt.show(block = False)
input()

############################
```

Python ▾ | Largeur des tabulations : 8 ▾ | Lig 50, Col 46 ▾ | INS