TP8_Transport

February 1, 2024

Table des matières

1 TP8 Équations de transport

[1]:
```
from math import *
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Question 1

Programmer les schémas aux différences finies décentrées à gauche et à droite, pour approcher la solution du problème

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) + a(t,x)\dfrac{\partial u}{\partial x}(t,x) = 0 \quad x \in \mathbb{R}, t > 0, \\ u(0,x) = u_0(x) \quad x \in \mathbb{R}, \end{cases} \tag{1}$$

lorsque a est constante.

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

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

On travaille ici sur le domaine borné $\mathbb{T} = [0,1]$ pour la variable d'espace, avec des conditions de bord périodique:

$$u(0,t) = u(1,t), \quad t \geq 0.$$

On se place alors dans le cas d'une donnée initiale u_0 supposée 1-périodique (de sorte que la solution est elle-même 1-périodique en espace à chaque instant, du moment que la fonction a l'est également). La subdivision x_j considérée est alors $\{x_j = j\Delta x,\ 0 \leq j \leq N\}$ mais les points $x_0 = 0$ et $x_N = 1$ sont identifiés par périodicité, les valeurs de la solution exacte coïncident en ces points. Le vecteur des inconnues à l'instant t^n est noté $(u_j^n)_{0\leq j\leq J}$ sachant que la condition de périodicité traduit le fait que la notation u_0^n désigne aussi bien la valeur u_J^n et le schéma (DFg) correspond précisément à l'itération suivante, sur $n \geq 0$:

$$u_j^{n+1} = u_j^n - a\frac{\Delta t}{\Delta x}\left(u_j^n - u_{j-1}^n\right), \quad 1 \leq j \leq J,$$
$$u_1^{n+1} = u_1^n - a\frac{\Delta t}{\Delta x}\left(u_1^n - u_J^n\right).$$

[2]:
```
def DFD (a,T,u0,J,dt):
    # différences finies à droite ;
    # J = nb de sous-intervalles de la subdivision en x, T = temps maximal
    # a = vitesse du transport (positive), u0 = donnee initiale (J valeurs)
    dx = 1/J
    N = floor(T/dt) # observer que l'instant de calcul
                    # final ne sera pas toujours exactement T
    v = np.zeros_like(u0) # facilitera les mides à jours
    U = np.zeros((N,J))   # stockage final
    U[0,:] = u0
    u = u0
    for n in range (1,N):
        v[0:-1] = u[1:]
        v[-1] = u[0]
        u = u - a*dt/dx* (v-u)
        U[n,:] = u
    return U
```

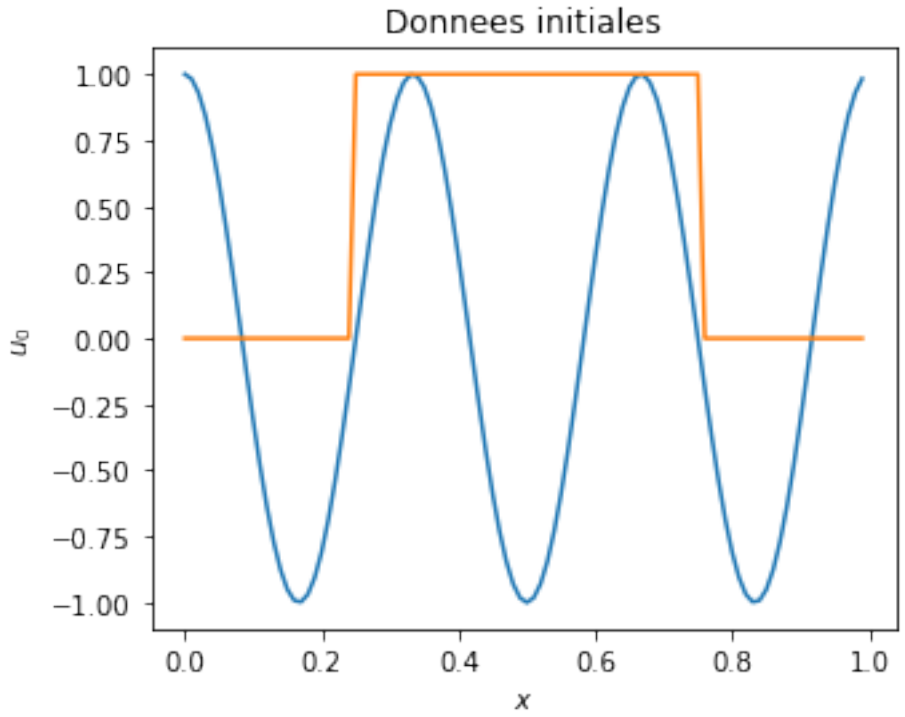
[3]:
```
def DFG (a,T,u0,J,dt):
    # différences finies à gauche ;
    # J = nb de sous-intervalles de la subdivision en x, T = temps maximal
    # a = vitesse du transport (positive), u0 = donnee initiale (J valeurs)
    dx = 1/J
    N = floor(T/dt) # observer que l'instant de calcul
                    # final ne sera pas toujours exactement T
    v = np.zeros_like(u0) # facilitera les mides à jours
    U = np.zeros((N,J))   # stockage final
    U[0,:] = u0
    u = u0
    for n in range (1,N):
        v[0] = U[n-1,-1]
        v[1:] = U[n-1,:-1]
```

```
        U[n,:] = U[n-1,:] - a*dt/dx* (U[n-1,:]-v)
        v[1:] = u[0:-1]
        v[0] = u[-1]
        u = u - a*dt/dx* (u-v)
        U[n,:] = u
    return U
```

[5]:
```
# Donnée initiale continue
def u0cont(x):
    return np.cos(6*pi*x)

# Donnée initiale discontinue
def u0disc(x):
    return np.array((x<=3./4)*(x>=1./4),dtype=float)

J=100
a=1
x = np.linspace(0,1,J+1)[0:J]
plt.rcParams['figure.figsize'] = (5, 4)
plt.plot(x,u0cont(x),x,u0disc(x))
plt.title('Donnees initiales')
plt.xlabel('$x$')
plt.ylabel('$u_0$')
plt.show()
```

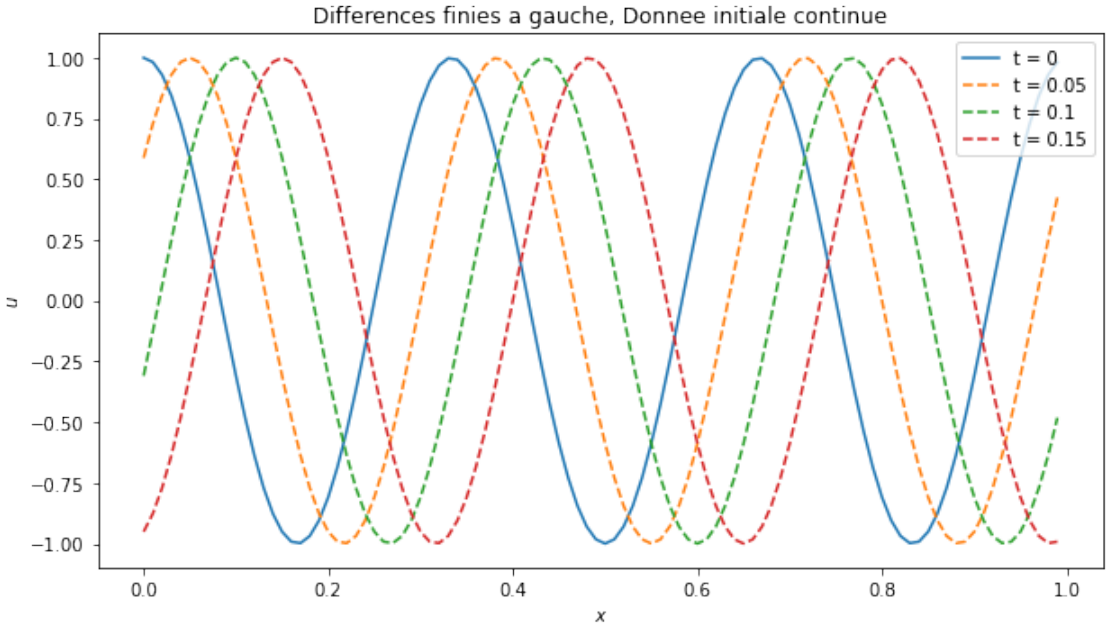


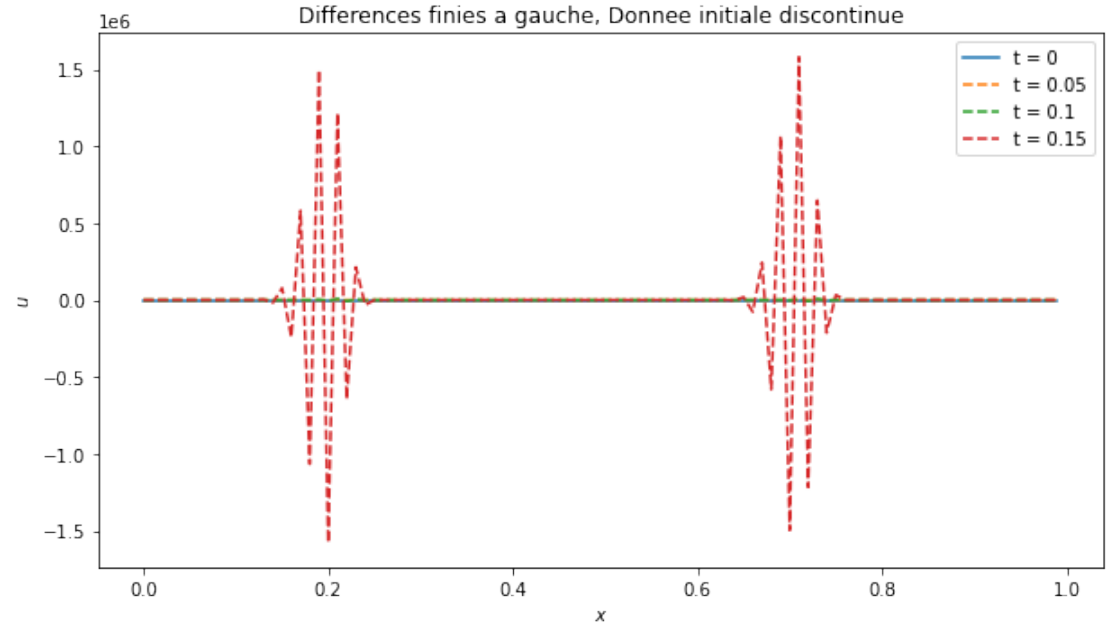
[6]:
```
J = 100
x = np.linspace(0,1,J+1)[0:J]
u0 = u0cont(x)
a = 1
T = 1
dt = 1/100
U1 = DFG(a,T,u0,J,dt)

u0 = u0disc(x)
U2 = DFD(a,T,u0,J,dt)
```

[8]:
```
plt.clf()
plt.rcParams['figure.figsize'] = (10, 12)
plt.subplot(2,1,1)
plt.plot(x,u0cont(x),label= 't = 0')
N1 = np.shape(U1)[0]
N2 = np.shape(U2)[0]
for i in range(5,20,5):
    plt.plot(x,U1[i,:],'--',label = 't = '+str(round(i*dt,2)))
plt.title('Differences finies a gauche, Donnee initiale continue')
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend()

plt.subplot(2,1,2)
plt.plot(x,u0disc(x),label= 't = 0')
for i in range(5,20,5):
    plt.plot(x,U2[i,:],'--',label = 't = '+str(round(i*dt,2)))
plt.title('Differences finies a gauche, Donnee initiale discontinue')
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend()
plt.show()
plt.show()
```



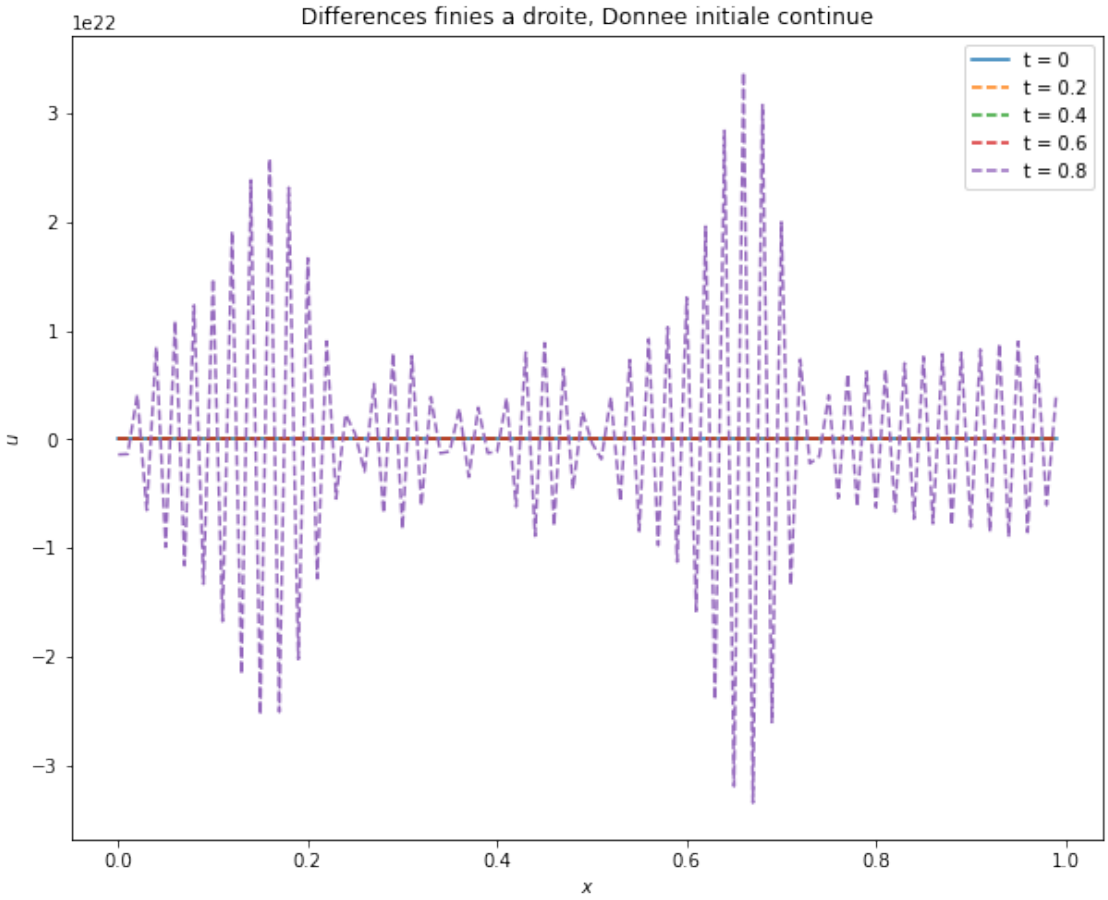


On observe ici que la solution numérique est égale à la solution exacte (translatée de la donnée initiale). On est en effet dans un cas très particulier où $a = 1$ et $\lambda = \frac{a\Delta t}{\Delta x} = 1$.

[9]:
```
u0 = u0cont(x)
J = 100
a = 1
T = 1
dt = 1/100
U2 = DFD(a,T,u0,J,dt)
```

```
N = floor(T/dt)

plt.rcParams['figure.figsize'] = (10,8)
plt.plot(x,u0cont(x),label= 't = 0')
for i in range(20,N,20):
    plt.plot(x,U2[i,:],'--',label = 't = '+str(round(i*dt,2)))
plt.title('Differences finies a droite, Donnee initiale continue')
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend()
plt.show()
```



On observe ici une instabilité numérique de la solution approchée, qui prend des valeurs très grandes quand t s'approche de 1.

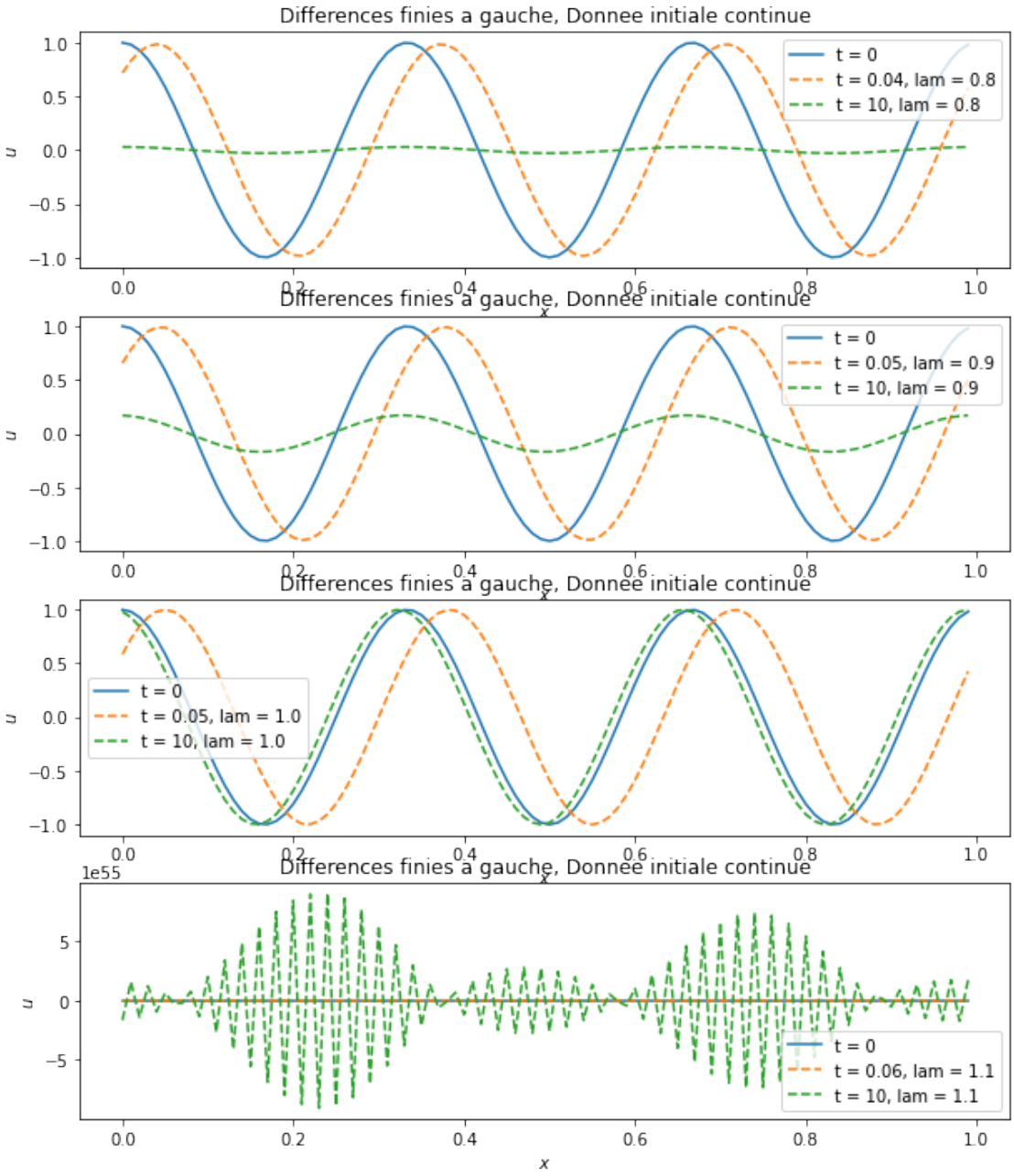
1.2 Question 2

Observer la dépendance des solutions approchées en fonction du paramètre $\lambda = a\frac{\Delta x}{\Delta t}$.

1.2.1 Schéma décentré à gauche

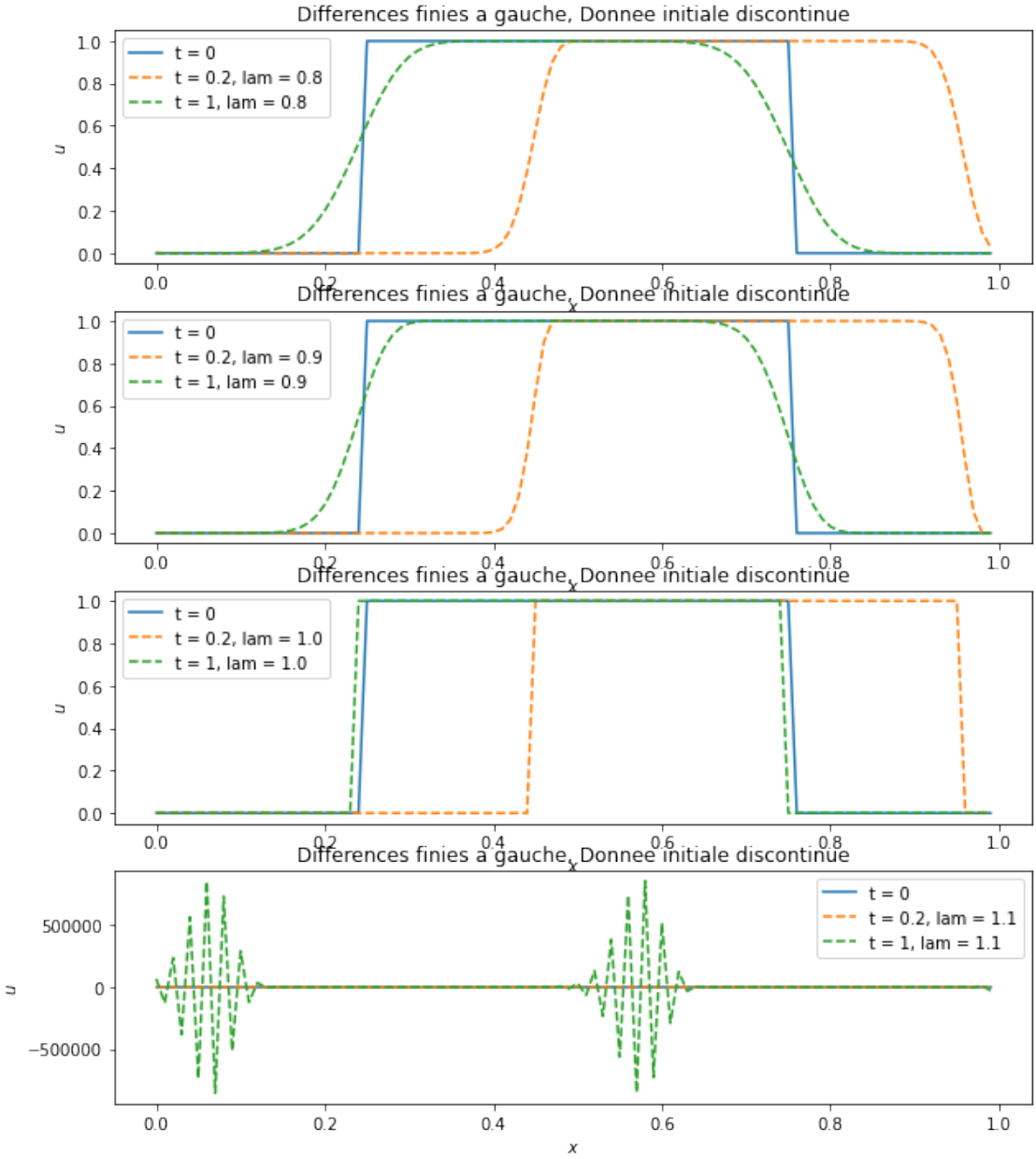
[10]:
```
u0 = u0cont(x)
a = 1
J = 100
T = 10
lam = [0.8, 0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester

plt.rcParams['figure.figsize'] = (10, 12)
for l in range(4):
    dt = lam[l]/(J*abs(a))
    U1 = DFG(a,T,u0,J,dt)
    plt.subplot(4,1,l+1)
    plt.plot(x,u0,label= 't = 0')
    plt.plot(x,U1[5,:],'--',label = 't = '+str(round(5*dt,2))+', lam =␣
 ↪'+str(lam[l]))
    plt.plot(x,U1[-1,:],'--',label = 't = '+str(T)+', lam = '+str(lam[l]))
    plt.title('Differences finies a gauche, Donnee initiale continue')
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
plt.show()
```



[11]:
```
a = 1
J = 100
x = np.linspace(0,1,J+1)[0:J]
u0 = u0disc(x)
T = 1
lam = [0.8, 0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester
plt.rcParams['figure.figsize'] = (10, 12)
```

```
for l in range(4):
    dt = lam[l]/(J*abs(a))
    U1 = DFG(a,T,u0,J,dt)
    plt.subplot(4,1,l+1)
    plt.plot(x,u0,label= 't = 0')
    N = floor(T/dt)
    n = floor(N/5)
    plt.plot(x,U1[n,:],'--',label = 't = '+str(round(n*dt,2))+', lam =␣
 ↪'+str(lam[l]))
    plt.plot(x,U1[-1,:],'--',label = 't = '+str(T)+', lam = '+str(lam[l]))
    plt.title('Differences finies a gauche, Donnee initiale discontinue')
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
plt.show()
```



1.2.2 Observations

On constate que le schéma calcule exactement la solution lorsque $\lambda = 1$. Cela s'explique par le fait que dans ce cas on a $u_j^{n+1} = u_{j-1}^n$ et donc la solution numérique vérifie cette propriété de la solution exacte :

$$u(j\Delta x, (n+1)\Delta t) = u((j-1)\Delta x, n\Delta t) = u_0(j\Delta x - (n+1)\Delta t).$$

Lorsque $\lambda \in]0, 1[$, le schéma donne des résultats raisonnables. L'amplitude de l'oscillation de la

solution régulière est atténuée avec le nombre d'itérations en temps. Pour l'autre test dans lequel la solution est discontinue, on observe un effet de régularisation. Ces deux effets sont dûs à la diffusion numérique du schéma, qui est une propriété du schéma qui favorise sa stabilité.

Lorsque $\lambda > 1$, la solution numérique est violemment malmenée.

Rappels de la théorie : Les observations précédentes sont directement liées aux propriétés de stabilité du schéma numérique. On rappelle quelques éléments de théorie valables pour les données numériques à valeurs dans $\ell^2(\mathbb{Z})$ (ce qui n'est pas le cas si elles sont périodiques de période N auquel cas il faudrait spécifier la transformée de Fourier discrète adaptée). Dans le cas d'un domaine non-borné en espace et de données discrètes dans $\ell^2(\mathbb{Z})$ repose sur la méthode dite de Fourier–Von Neumann. Rappelons-en brièvement le principe: on définit la transformée de Fourier $\hat{u}(\xi) = \sum_{j\in\mathbb{Z}} u_j e^{-i\xi j\Delta x}$ de sorte que le schéma numérique se réécrit en variable fréquentielle ξ (le détail du calcul est laissé à votre soin) par

$$\hat{u}^{n+1}(\xi) = (1 - \lambda + \lambda e^{-i\xi\Delta x})\hat{u}^n(\xi), \quad \xi \in \mathbb{R}. \tag{2}$$

Le schéma se résout explicitement en variable ξ à travers de simples suites géométriques de raison $\alpha(\xi\Delta x) = (1 - \lambda + \lambda e^{-i\xi\Delta x})$, de sorte que

$$\hat{u}^n(\xi) = (\alpha(\xi\Delta x))^n \hat{u}^0(\xi).$$

Par l'isomorphisme de Fourier on peut reconstruire la solution dans le domaine spatial par la formule d'inversion sur la fonction $2\pi/\Delta x$-périodique $\hat{u}^n$:

$$u_j^n = \frac{\Delta x}{2\pi} \int_{-\pi/\Delta x}^{\pi/\Delta x} \hat{u}^n(\xi) e^{i\xi j\Delta x} d\xi.$$

Il est temps de rappeler la définition de la stabilité du schéma, essentielle pour l'analyse des schémas aux différences finies:

Definition.
Le schéma numérique est dit stable en norme ℓ^2 si pour tout $T > 0$, il existe une constante $C > 0$ telle que pour toute donnée initiale $u^0 \in \ell^2(\mathbb{Z})$ et tout couple $(\Delta x, \Delta t) \in \mathcal{S}$, on ait pour tout $n \in \mathbb{N}$ tel que $n\Delta t \leq T$:

$$\|u^n\|_2 \leq C\|u^0\|_2.$$

Le domaine de stabilité $\mathcal{S}$ est ici une partie non-vide de $(\mathbb{R}_+^\star)^2$ adhérente à $(0,0)$ qui est généralement caractérisée par la condition dite CFL du schéma (Courant-Friedrichs-Lewy).

Par la formule de Plancherel, on a ici :

$$\|u^n\|_2^2 = \sum_{j\in\mathbb{Z}} |u_j|^2 = \frac{\Delta x}{2\pi} \int_{-\pi/\Delta x}^{\pi/\Delta x} |\hat{u}^n(\xi)|^2 d\xi.$$

Supposons la propriété suivante satisfaite:

$$\forall (\Delta x, \Delta t) \in \mathcal{S},\ \forall \xi \in [-\pi/\Delta x, \pi/\Delta x], \quad |\alpha(\xi\Delta x)| \leq 1, \tag{3}$$

on aura alors la majoration

$$\|u^n\|_2^2 \leq \frac{\Delta x}{2\pi} \int_{-\pi/\Delta x}^{\pi/\Delta x} |\hat{u}^0(\xi)|^2 d\xi = \|u^0\|_2^2.$$

La stabilité du schéma sera alors garantie (avec de plus une constante $C = 1$ indépendante de T).

Supposons au contraire qu'il existe $y_0 \in [-\pi, \pi]$ tel que $|\alpha(y_0)| > 1$. Par continuité de α, la même inégalité stricte sera valable sur un voisinage ouvert de y_0, noté dans la suite $\mathcal{U} \subset [-\pi, \pi]$. On notera $m > 1$ un minorant sur $\mathcal{U}$. Considérons alors pour chaque Δx envisagé, la donnée initiale (v_j^0) telle que $\hat{v}^0(\xi) = \chi_{\mathcal{U}/\Delta x}(\xi)$ sur $[-\pi/\Delta x, \pi/\Delta x]$, c'est à dire:

$$v_j^0 = \frac{1}{2\pi} \int_{\mathcal{U}} e^{ijy} dy, \quad j \in \mathbb{Z}.$$

Alors par Plancherel $\|v^0\|_2^2 = \dfrac{\lambda(\mathcal{U})}{2\pi}$ et la formule de récurrence obtenue précédemment donne

$$\|v^n\|_2 \geq m^n \|v^0\|_2,$$

Étant donné que n tend vers l'infini comme $T/\Delta t$ si Δt tend vers 0 avec $n\Delta t$ proche de T, la propriété de stabilité ne peut pas être vérifiée, quel que soit le choix de $\mathcal{S}$.

La condition (3) peut être affaiblie en $|\alpha(\xi\Delta x)| \leq 1 + \nu\Delta t$, avec alors la constante de stabilité de la forme $C = \mathrm{e}^{\nu T}$.

Comme mentionné plus haut, pour analyser le schéma programmé dans le TP, il faudrait en réalité étudier la stabilité pour des données discrètes (u_j^0) $N-$périodiques (et non plus dans $\ell^2(\mathbb{Z})$) et simplement adapter l'analyse de Fourier. On laisse ces reflexions de côté pour se concentrer plutôt sur la caractérisation de la stabilité utilisée dans la proposition (3).

Pour le schéma (DFg), le calcul donne $\alpha(y) = 1 - \lambda + \lambda e^{-iy}$ qui décrit le cercle du plan complexe de centre $1 - k$ et de rayon k lorsque y parcourt $\mathbb{R}$. En particulier on peut montrer que pour tout $y \in \mathbb{R}$, $|\alpha(y)| \leq 1$ si et seulement si $0 \leq \lambda \leq 1$. Le domaine de stabilité $\mathcal{S}$ est alors l'ensemble suivant :

$$\mathcal{S} = \{(\Delta x, \Delta t) \in (\mathbb{R}_+^\star)^2 \mid 0 \leq a\Delta t \leq \Delta x\}.$$

Dans le cas du schéma (DFd), le même calcul donne $\alpha(y) = 1 + k\lambda - \lambda e^{iy}$ et la condition équivaut à $-1 \leq \lambda \leq 0$. le domaine de stabilité est donc

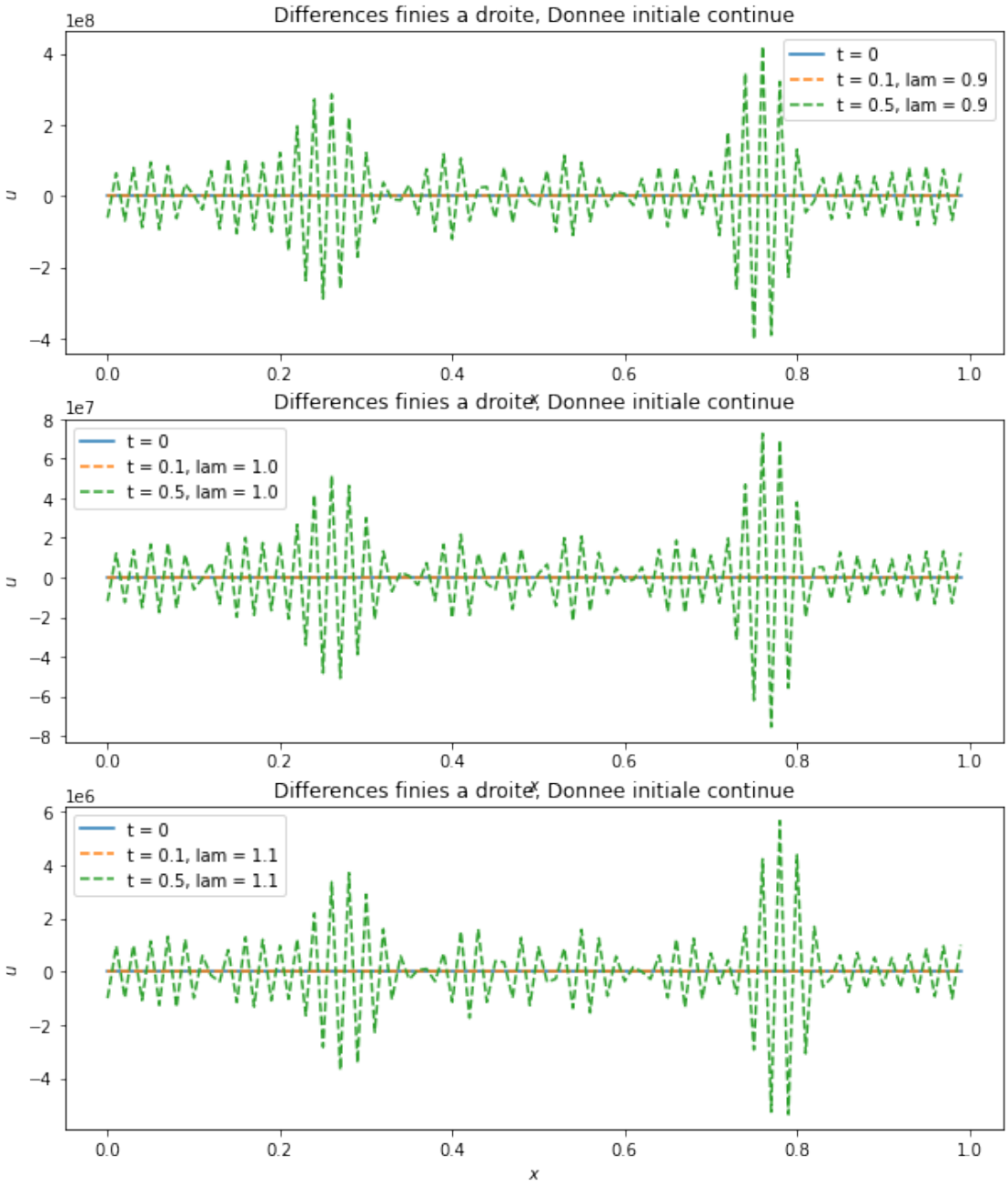
$$\mathcal{S} = \{(\Delta x, \Delta t) \in (\mathbb{R}_+^\star)^2 \mid -\Delta x \leq a\Delta t \leq 0\}.$$

En conséquence, si $a > 0$ le schéma (DFd) est inconditionnellement instable (ce qui signifie que $\mathcal{S} = \emptyset$). De même si $a < 0$ le schéma (DFg) est inconditionnellement instable. Par contre, lorsque $a > 0$ le schéma (DFg) est stable sous la condition $a\Delta t \leq \Delta x$. Lorsque $a < 0$ le schéma (DFd) est stable sous la condition $-\Delta x \leq a\Delta t$.

1.2.3 Schéma décentré à droite

[12]:
```
a = 1
J = 100
x = np.linspace(0,1,J+1)[0:J]
u0 = u0cont(x)
# u0 = u0disc(x)
T = 0.5
lam = [0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester
```

```
for l in range(3):
    dt = lam[l]/(J*abs(a))
    U1 = DFD(a,T,u0,J,dt)
    plt.subplot(3,1,l+1)
    N = floor(T/dt)
    n = floor(N/5)
    plt.plot(x,u0,label= 't = 0')
    plt.plot(x,U1[n,:],'--',label = 't = '+str(round(n*dt,2))+', lam =␣
 ↪'+str(lam[l]))
    plt.plot(x,U1[-1,:],'--',label = 't = '+str(T)+', lam = '+str(lam[l]))
    plt.title('Differences finies a droite, Donnee initiale continue')
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
plt.show()
```



On observe, quelle que soit la valeur de lambda, que la solution prend des valeurs très grandes quand t augmente. Les résultats numériques montrent ainsi une instabilité de ce schéma.

1.3 Prolongement : Schéma centré

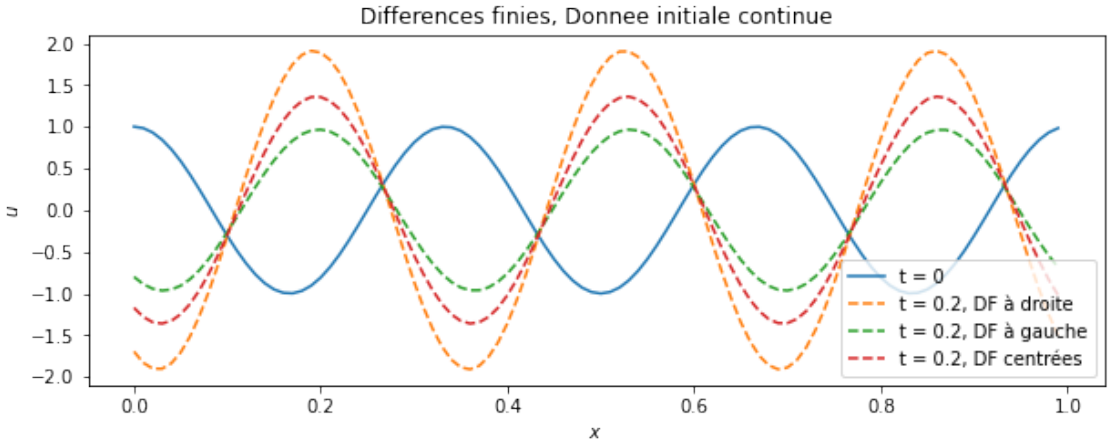
Programmer le schéma différences finies centré défini par

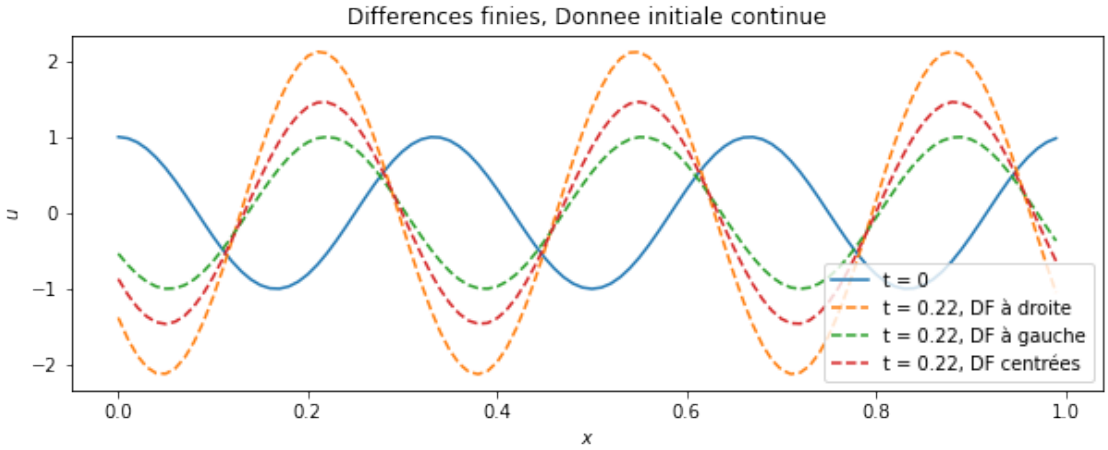
$$u_j^{n+1} = u_j^n - a\frac{\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n)$$

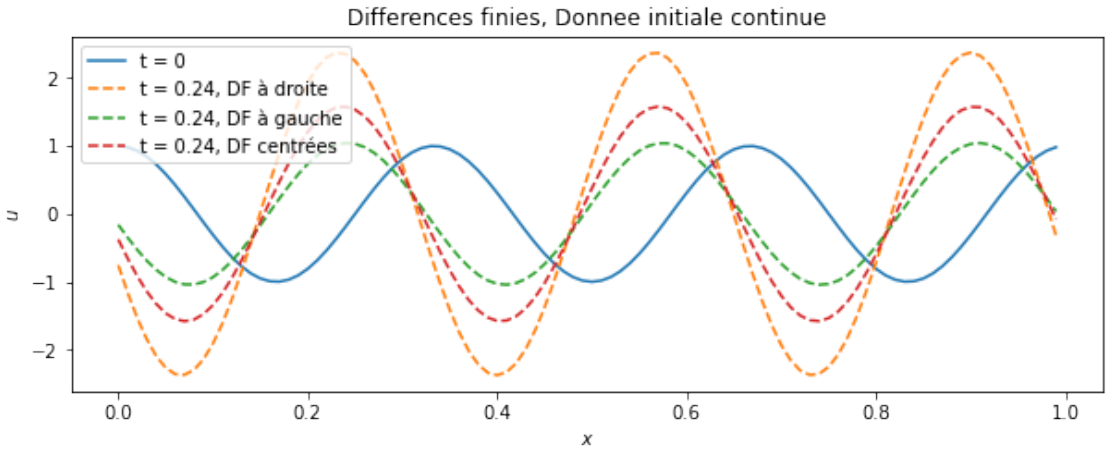
[13]:
```
def DFC (a,T,u0,J,dt):
    # différences finies centrées ;
    # J = nb de sous-intervalles de la subdivision en x, T = temps maximal
    # a = vitesse du transport (positive), u0 = donnee initiale (J valeurs)
    dx = 1/J
    N = floor(T/dt) # observer que l'instant de calcul
                    # final ne sera pas toujours exactement T
    v1 = np.zeros_like(u0) # facilitera les mides à jours
    v2 = np.zeros_like(u0) # facilitera les mides à jours
    U = np.zeros((N,J))   # stockage final
    U[0,:] = u0
    u = u0
    for n in range (1,N):
        v1[0:-1] = u[1:]
        v1[-1] = u[0]
        v2[1:] = u[:-1]
        v2[0] = u[-1]
        u = u - a*dt/(2*dx)* (v1-v2)
        U[n,:] = u
    return U
```

[14]:
```
a = 1
J = 100
x = np.linspace(0,1,J+1)[0:J]
u0 = u0cont(x)
# u0 = u0disc(x)
T = 0.5
N = floor(T/dt)
n = floor(N/2)
lam = [0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester
for l in range(3):
    dt = lam[l]/(J*abs(a))
    U1 = DFD(a,T,u0,J,dt)
    U2 = DFG(a,T,u0,J,dt)
    U3 = DFC(a,T,u0,J,dt)
    plt.subplot(3,1,l+1)
    plt.plot(x,u0,label= 't = 0')
    plt.plot(x,U1[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à droite')
    plt.plot(x,U2[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à gauche')
    plt.plot(x,U3[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF centrées')
    plt.title('Differences finies, Donnee initiale continue')
```

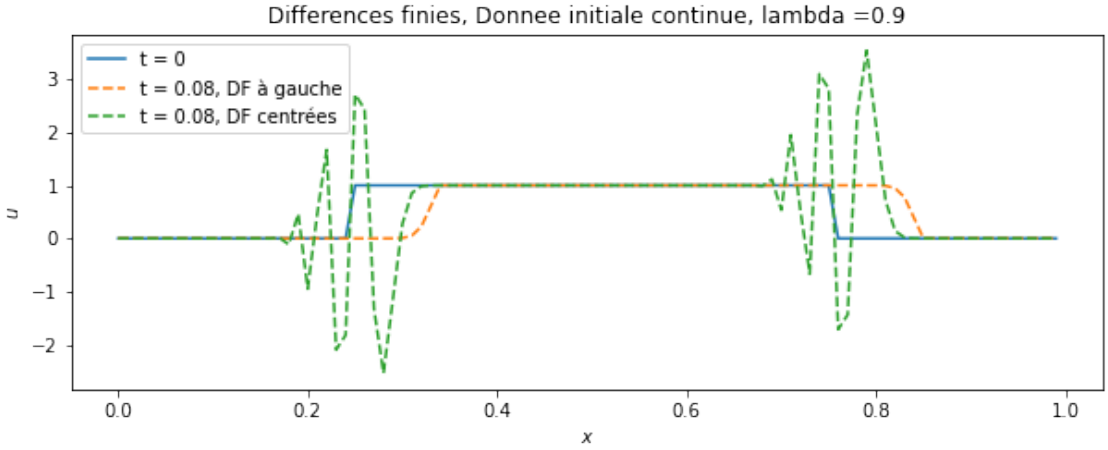
```
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
    plt.show()
```

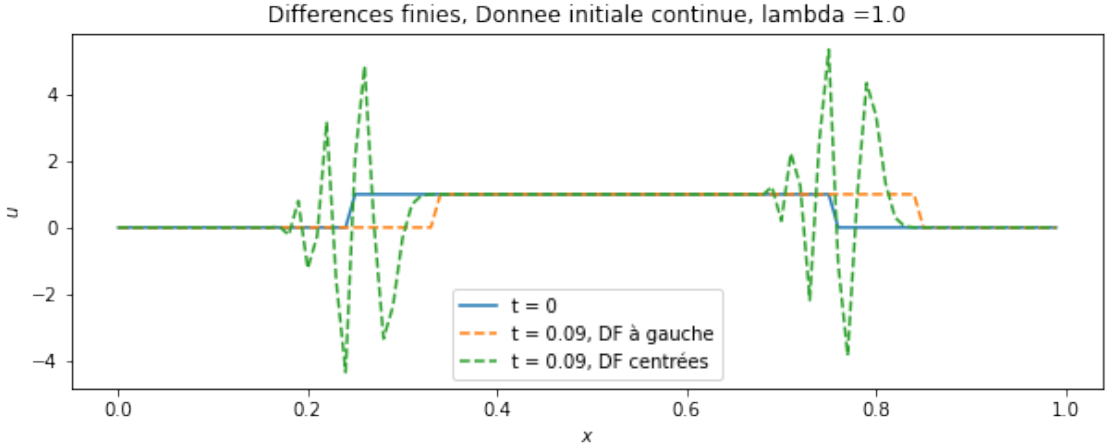


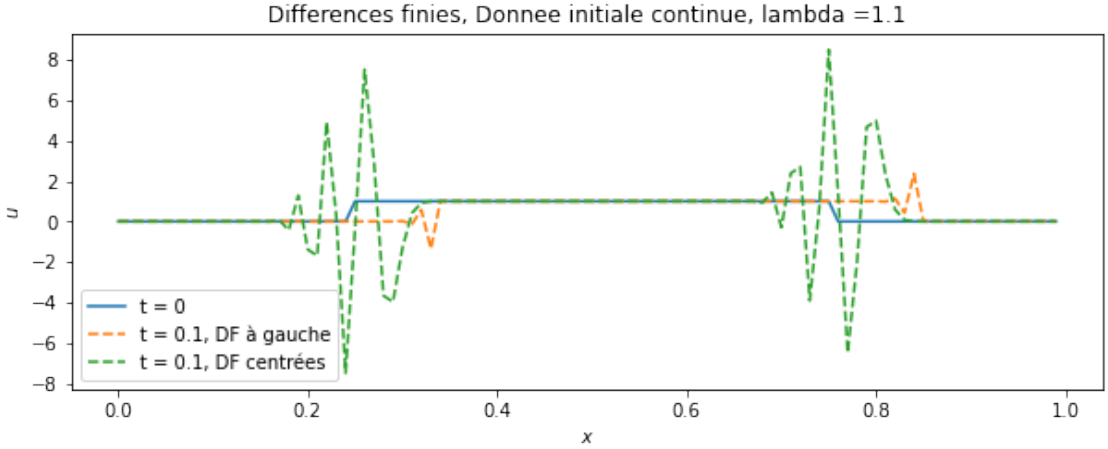




[15]:
```
a = 1
J = 100
x = np.linspace(0,1,J+1)[0:J]
# u0 = u0cont(x)
u0 = u0disc(x)
T = 0.5
N = floor(T/dt)
n = floor(N/5)
lam = [0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester
for l in range(3):
    dt = lam[l]/(J*abs(a))
    U1 = DFD(a,T,u0,J,dt)
    U2 = DFG(a,T,u0,J,dt)
    U3 = DFC(a,T,u0,J,dt)
    plt.subplot(3,1,l+1)
    plt.plot(x,u0,label= 't = 0')
   # plt.plot(x,U1[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à droite')
    plt.plot(x,U2[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à gauche')
    plt.plot(x,U3[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF centrées')
    plt.title('Differences finies, Donnee initiale continue, lambda␣
 ↪='+str(lam[l]))
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
    plt.show()
```


Differences finies, Donnee initiale continue, lambda =0.9
t = 0
t = 0.08, DF à gauche
t = 0.08, DF centrées
u
x


Differences finies, Donnee initiale continue, lambda =1.0
t = 0
t = 0.09, DF à gauche
t = 0.09, DF centrées
u
x


Differences finies, Donnee initiale continue, lambda =1.1
t = 0
t = 0.1, DF à gauche
t = 0.1, DF centrées
u
x

On observe sur tous les tests une instabilité pour le schéma centré : la solution numérique calculée prend des valeurs trop grandes lorsque le temps augmente. Ceci est en accord avec le calcul du domaine de stabilité (voir corrigé).

1.4 Schéma de Lax-Wendroff

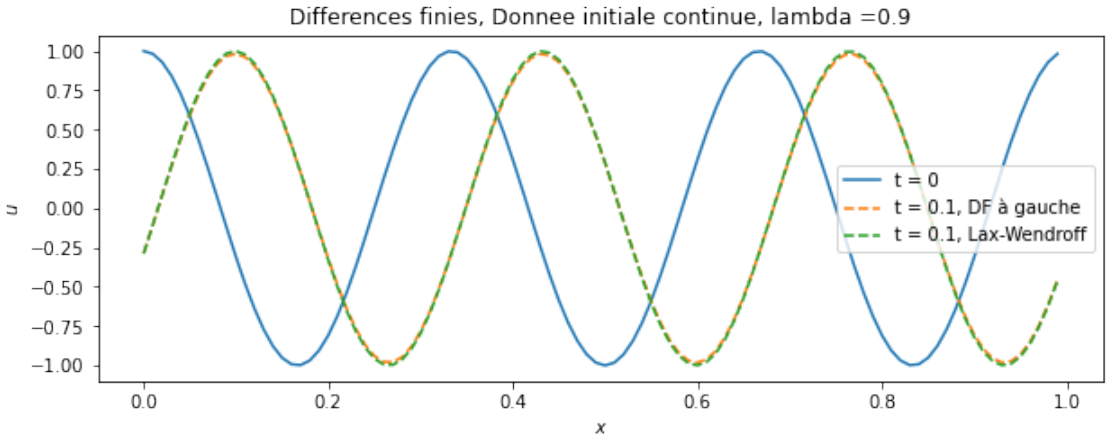
Programmer le schéma différences finies de Lax-Wendroff défini par

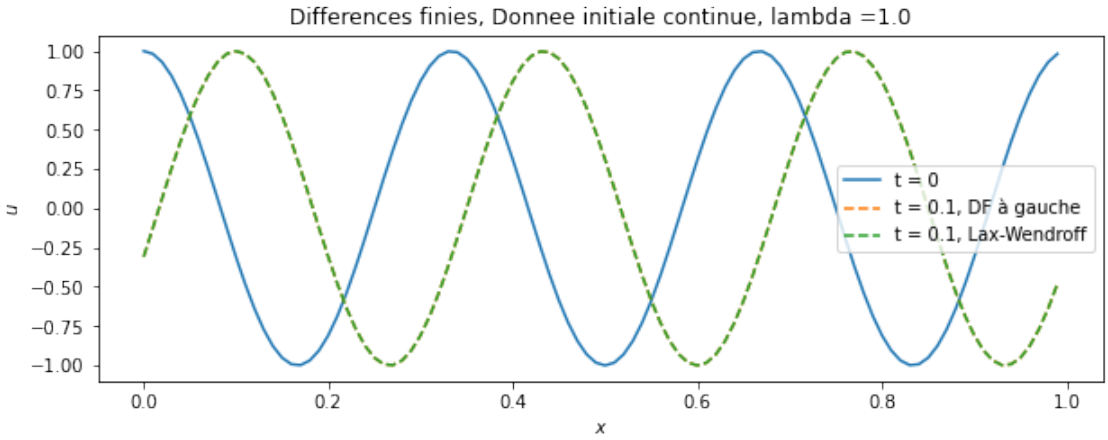
$$u_j^{n+1} = u_j^n - a\frac{\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) + a^2\frac{(\Delta t)^2}{2(\Delta x)^2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

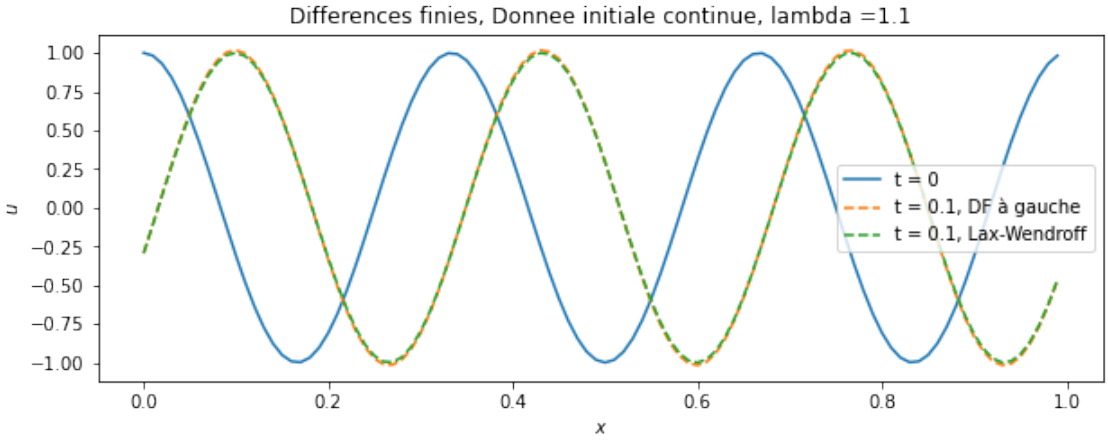
[16]:
```
def LW (a,T,u0,J,dt):
    # différences finies centrées ;
    # J = nb de sous-intervalles de la subdivision en x, T = temps maximal
    # a = vitesse du transport (positive), u0 = donnee initiale (J valeurs)
    dx = 1/J
    N = floor(T/dt) # observer que l'instant de calcul
                    # final ne sera pas toujours exactement T
    v1 = np.zeros_like(u0) # facilitera les mises à jours
    v2 = np.zeros_like(u0) # facilitera les mises à jours
    U = np.zeros((N,J))   # stockage final
    U[0,:] = u0
    u = u0
    for n in range (1,N):
        v1[0:-1] = u[1:]
        v1[-1] = u[0]
        v2[1:] = u[:-1]
        v2[0] = u[-1]
        u = u - a*dt/(2*dx)* (v1-v2) + a**2*dt**2/(2*dx**2) *(v1-2*u+v2)
        U[n,:] = u
    return U
```

[17]:
```
# Test du schéma de Lax-Wendroff sur donnée continue
a = 1
J = 100
x = np.linspace(0,1,J+1)[0:J]
u0 = u0cont(x)
#u0 = u0disc(x)
T = 0.5
lam = [0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester
for l in range(3):
    dt = lam[l]/(J*abs(a))
    N = floor(T/dt)
    n = floor(N/5)
    U1 = DFD(a,T,u0,J,dt)
    U2 = DFG(a,T,u0,J,dt)
    U4 = LW(a,T,u0,J,dt)
```

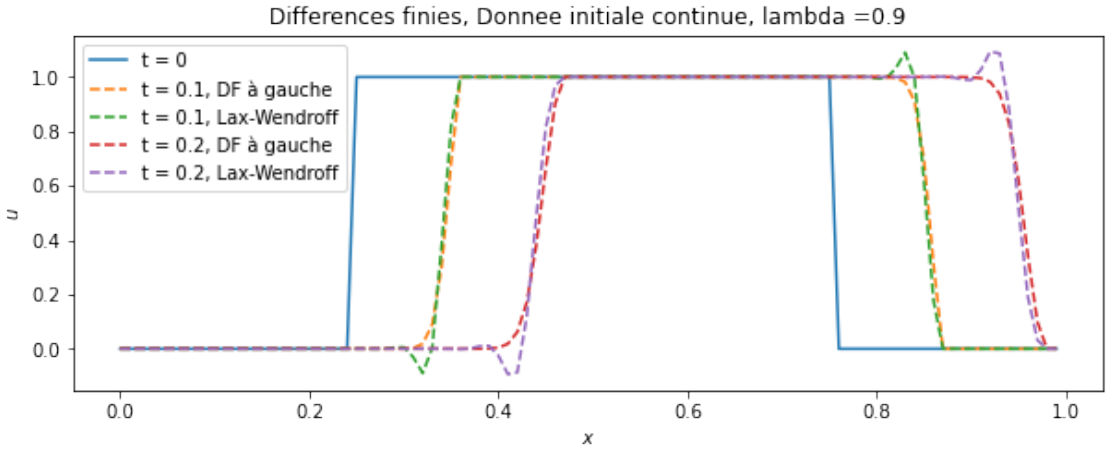
```
    plt.subplot(3,1,l+1)
    plt.plot(x,u0,label= 't = 0')
   # plt.plot(x,U1[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à droite')
    plt.plot(x,U2[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à gauche')
    plt.plot(x,U4[n,:],'--',label = 't = '+str(round(n*dt,2))+', Lax-Wendroff')
    plt.title('Differences finies, Donnee initiale continue, lambda␣
↪='+str(lam[l]))
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
    plt.show()
```

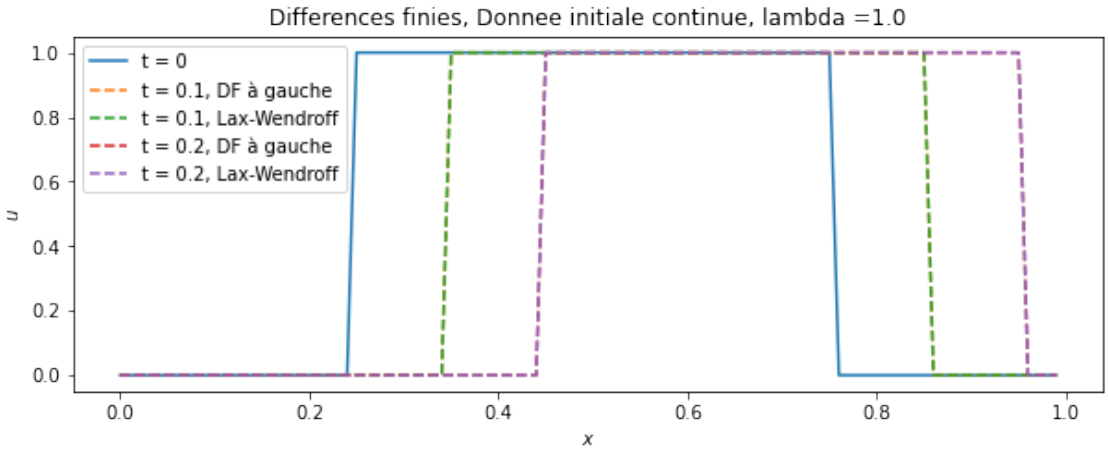


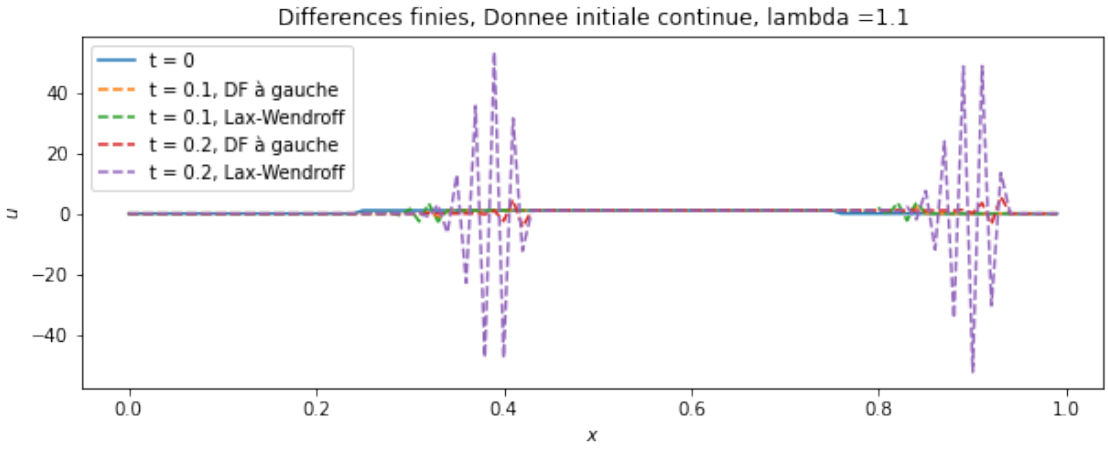




[18]:
```
# Test du schéma de Lax-Wendroff sur donnée discontinue
a = 1
J = 100
x = np.linspace(0,1,J+1)[0:J]
# u0 = u0cont(x)
u0 = u0disc(x)
T = 0.5
lam = [0.9, 1.0, 1.1] # valeurs du paramètre lambda à tester
for l in range(3):
    dt = lam[l]/(J*abs(a))
    N = floor(T/dt)
    n = floor(N/5)
    U1 = DFD(a,T,u0,J,dt)
    U2 = DFG(a,T,u0,J,dt)
    U4 = LW(a,T,u0,J,dt)
    plt.subplot(3,1,l+1)
    plt.plot(x,u0,label= 't = 0')
   # plt.plot(x,U1[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à droite')
    plt.plot(x,U2[n,:],'--',label = 't = '+str(round(n*dt,2))+', DF à gauche')
    plt.plot(x,U4[n,:],'--',label = 't = '+str(round(n*dt,2))+', Lax-Wendroff')
    plt.plot(x,U2[2*n,:],'--',label = 't = '+str(round(2*n*dt,2))+', DF à␣
↪gauche')
    plt.plot(x,U4[2*n,:],'--',label = 't = '+str(round(2*n*dt,2))+',␣
↪Lax-Wendroff')
    plt.title('Differences finies, Donnee initiale continue, lambda␣
↪='+str(lam[l]))
    plt.xlabel('$x$')
    plt.ylabel('$u$')
    plt.legend()
    plt.show()
```


Differences finies, Donnee initiale continue, lambda =0.9
t = 0
t = 0.1, DF à gauche
t = 0.1, Lax-Wendroff
t = 0.2, DF à gauche
t = 0.2, Lax-Wendroff
u
x
0.0
0.2
0.4
0.6
0.8
1.0


Differences finies, Donnee initiale continue, lambda =1.0
t = 0
t = 0.1, DF à gauche
t = 0.1, Lax-Wendroff
t = 0.2, DF à gauche
t = 0.2, Lax-Wendroff
u
x
0.0
0.2
0.4
0.6
0.8
1.0


Differences finies, Donnee initiale continue, lambda =1.1
t = 0
t = 0.1, DF à gauche
t = 0.1, Lax-Wendroff
t = 0.2, DF à gauche
t = 0.2, Lax-Wendroff
u
x
40
20
0
−20
−40
0.0
0.2
0.4
0.6
0.8
1.0

On constate que le schéma de Lax-Wendroff approche bien la solution lorsque $\lambda \leq 1$ mais devient instable lorsque $\lambda > 1$. Ceci est en accord avec le calcul du domaine de stabilité (voir corrigé).

1.5 Schéma "upwind" lorsque a change de signe

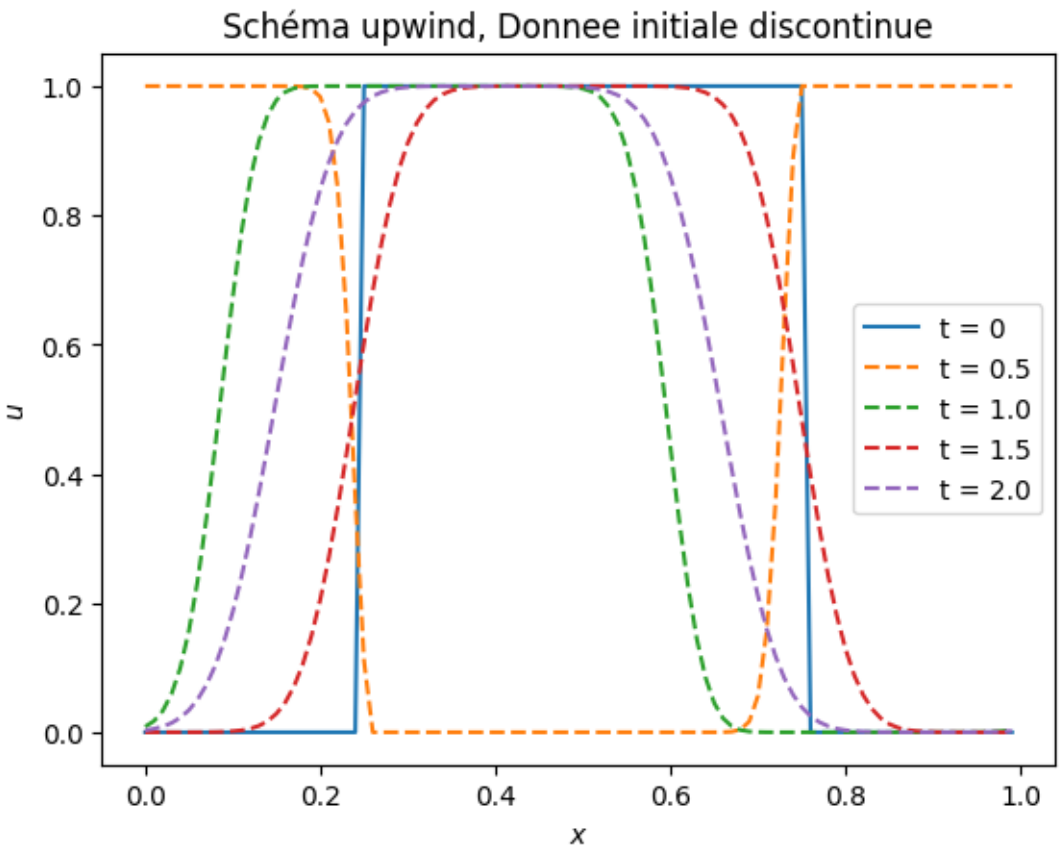
Lorsque a ne dépend que de t, on peut utiliser le décentrement à gauche ou à droite selon le signe de $a(t_n)$, l'idée générale étant qu"on doit"remonter le courant" (d'où le nom : schéma upwind). Il faut prendre garde que le pas de temps à considérer pour avoir un schéma stable dépend alors de $a(t_n)$ et donc de n, de sorte que si $a(t_n)$ devient très grand au cours du temps, il faudra réduire Δt en proportion (rien n'empêche de choisir Δt dépendant de l'itération de temps n, les t_n étant alors non-équirépartis).

L'analyse de stabilité en norme 2 précédemment employée se trouve un peu complexifiée. On peut cependant s'appuyer sur le fait que le domaine en temps T est fixé et retenir un majorant $M > 0$ de $|a(t)|$ lorsque t parcourt $[0, T]$ et le pas de temps uniforme correspondant. En choisissant une suite de pas de temps majorée par $\frac{\Delta x}{M}$, on peut facilement montrer par récurrence que $\|u_n\|_2 \leq \|u_0\|_2$, pour tout $t_n \leq T$.

[19]:
```
def upwind(J,T,a,u0):
# a est une fonction ne dépendant que de t, par ex a = np.cos
    A = a(np.linspace(0,T,100))
    dx = 1/J
    amax = max(abs(A))
    N = floor(T*amax/dx)
    v1 = np.zeros_like(u0)
    v2 = v1.copy()
    U = np.zeros((N,J))
    U[0,:] = u0
    listeT = [0]
    t = 0
    n = 0
    while (t<T and n<N-1):
        dt =  dx / max (1, abs(a(t)))
        # adaptation du pas , avec saturation pour eviter
        #les trop grands pas de temps si |a| est petit
        dt = min(dt, T-t)
        t += dt
        listeT.append(t)
        n += 1
        v1[:-1] = U[n-1,1:] # schéma décentré à droite
        v1[-1] = U[n-1,0]
        v2[0] = U[n-1,-1] # schéma décentré à gauche
        v2[1:] = U[n-1,:-1]
        if a(t) >=0: # on décentre à gauche
            U[n,:] = U[n-1,:] - a(t)*dt/(dx)* (U[n-1,:]-v2)
        else:
            # on decentre a droite
```

```
            U[n,:] = U[n-1,:] - a(t)*dt/(dx)* (v1-U[n-1,:])
    return U,listeT
```

[20]:
```
# Test du schéma upwind sur donnée discontinue
a = np.cos
J = 100
x = np.linspace(0,1,J+1)[0:J]
# u0 = u0cont(x)
u0 = u0disc(x)
plt.style.use('default')
U1,listeT = upwind(J,3,a,u0)
N = np.shape(U1)[0]
plt.plot(x,u0,label= 't = 0')
plt.plot(x,U1[50,:],'--',label = 't = '+str(round(listeT[50],2)))
plt.plot(x,U1[100,:],'--',label = 't = '+str(round(listeT[100],2)))
plt.plot(x,U1[150,:],'--',label = 't = '+str(round(listeT[150],2)))
plt.plot(x,U1[200,:],'--',label = 't = '+str(round(listeT[200],2)))
plt.title('Schéma upwind, Donnee initiale discontinue')
plt.xlabel('$x$')
plt.ylabel('$u$')
plt.legend()
plt.show()
```

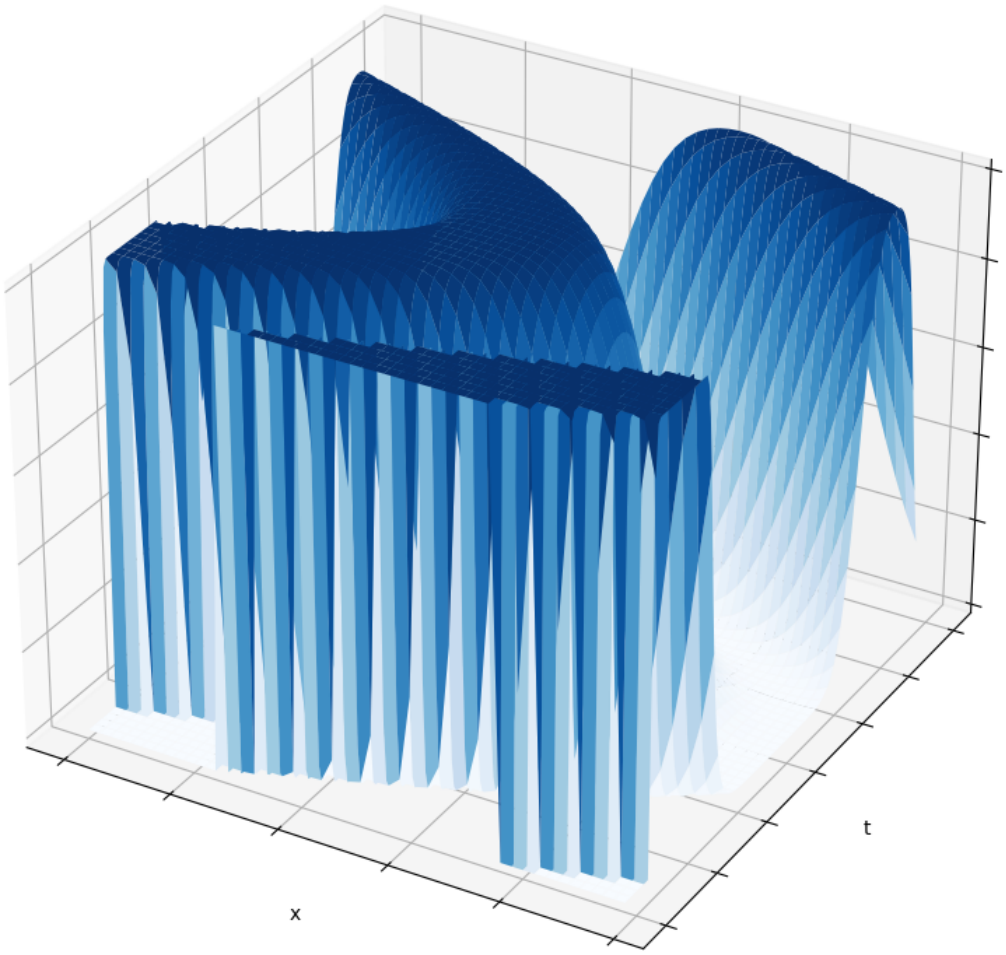


On observe (ou on devine) que le créneau se déplace d'abord vers la droite en ralentissant progressivement (jusqu'à $t = 1.5$ environ, en fait jusqu'à $t = \pi/2$) puis repart vers la gauche. Malgré le changement de signe de la vitesse, la solution numérique ne semble pas présenter d'instabilité. On va faire une représentation en 3D pour mieux visualiser ce changement de sens de la vitesse.

[21]:
```
# Test du schéma upwind sur donnée discontinue et représentation graphique
plt.style.use('_mpl-gallery')
plt.rcParams['figure.figsize'] = (10,8)

from matplotlib import cm
a = np.cos
J = 50
x = np.linspace(0,1,J+1)[0:J]
# u0 = u0cont(x)
u0 = u0disc(x)
lam = 0.9
U1,listeT = upwind(J,3,a,u0)
N = np.shape(U1)[0]
print(N)
X = np.ones((N,J))
for n in range(N):
    X[n] = x
Y = np.ones((N,J))
for j in range(J):
    Y[:,j] = listeT
# Plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(X,Y, U1, cmap=cm.Blues)
#ax.plot_surface(X,Y, U1)
ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])
ax.set(xlabel='x',ylabel='t',zlabel='u')
plt.title('Transport à vitesse variable, schéma upwind')
plt.show()
```

```
150
```


Transport à vitesse variable, schéma upwind
x
t

[ ]: