TP7 Équations de transport

In [47]:
```
from math import *
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (5, 4)
```

Question 1

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

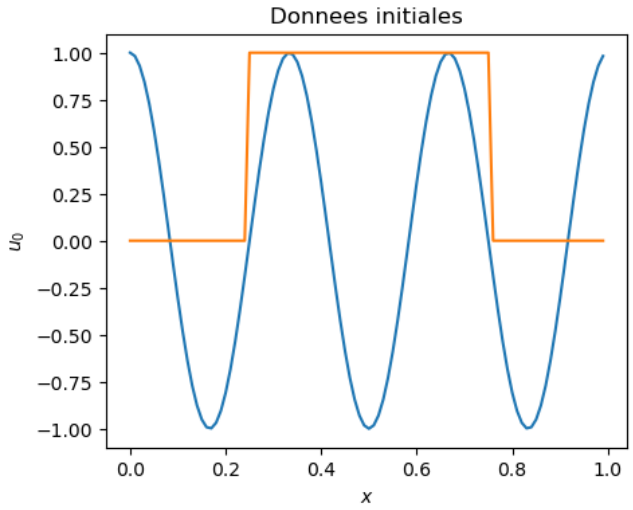
$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) + a(t,x)\dfrac{\partial u}{\partial x}(t,x) = 0 \quad x \in \mathbb{R}, t > 0, \\ u(0,x) = u_0(x) \quad x \in \mathbb{R}, \end{cases} \tag{1}$$

lorsque a est constante.

In [3]:
```
def DFD (J,T,a,dt,u0):
    # 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
    v = np.zeros_like(u0)
    N = floor(T/dt)
    U = np.zeros((N,J))
    U[0,:] = u0
    for n in range (1,N):
        v[:-1] = U[n-1,1:]
        v[-1] = U[n-1,0]
        U[n,:] = U[n-1,:] - a*dt/dx* (v-U[n-1,:])
    return U
```

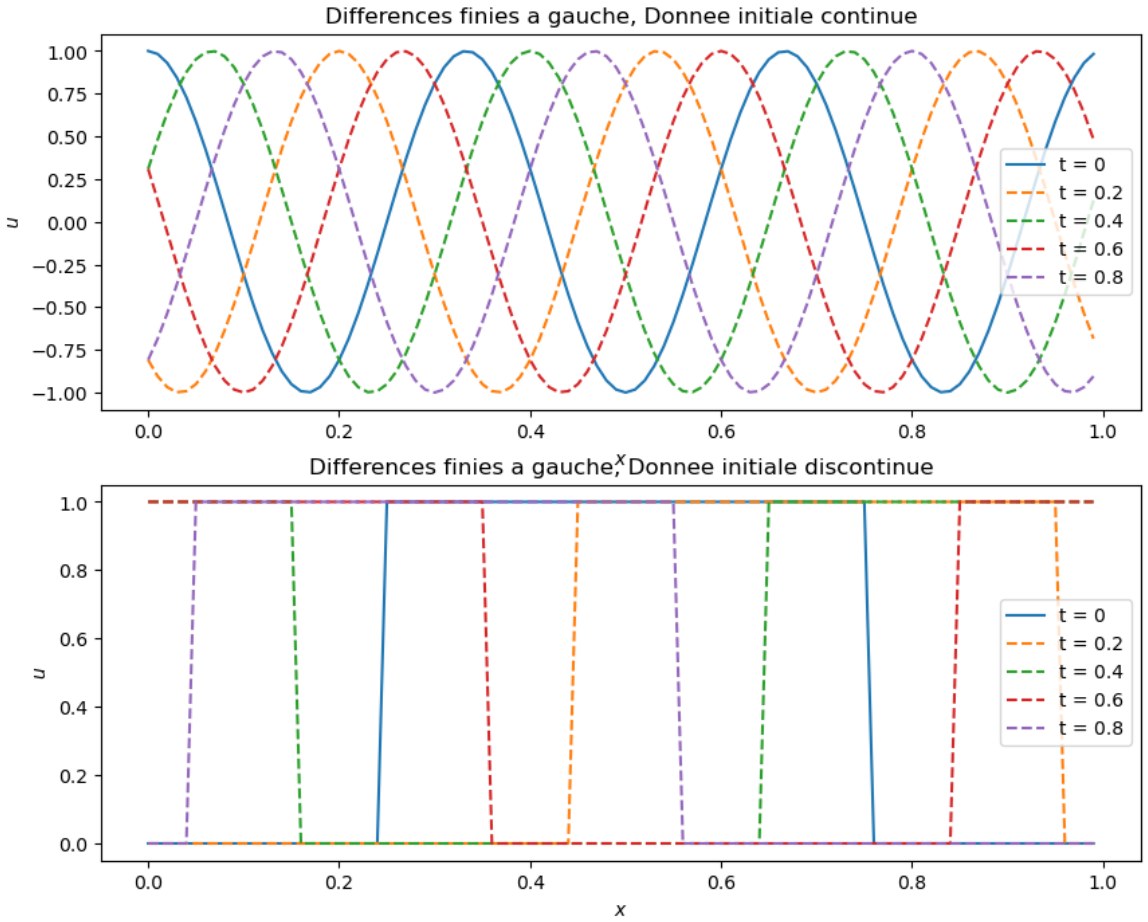
In [4]:
```
def DFG (J,T,a,dt,u0):
    # différences finies à gauche ;
    # J = nb de sous-intervalles de la subdivision en x, T = temps maximal
    # a = vitesse du transport (positive), u0 = donnee initiale (J valeurs)
    dx = 1/J
    v = np.zeros_like(u0)
    N = floor(T/dt)
    U = np.zeros((N,J))
    U[0,:] = 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)
    return U
```

In [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()
```



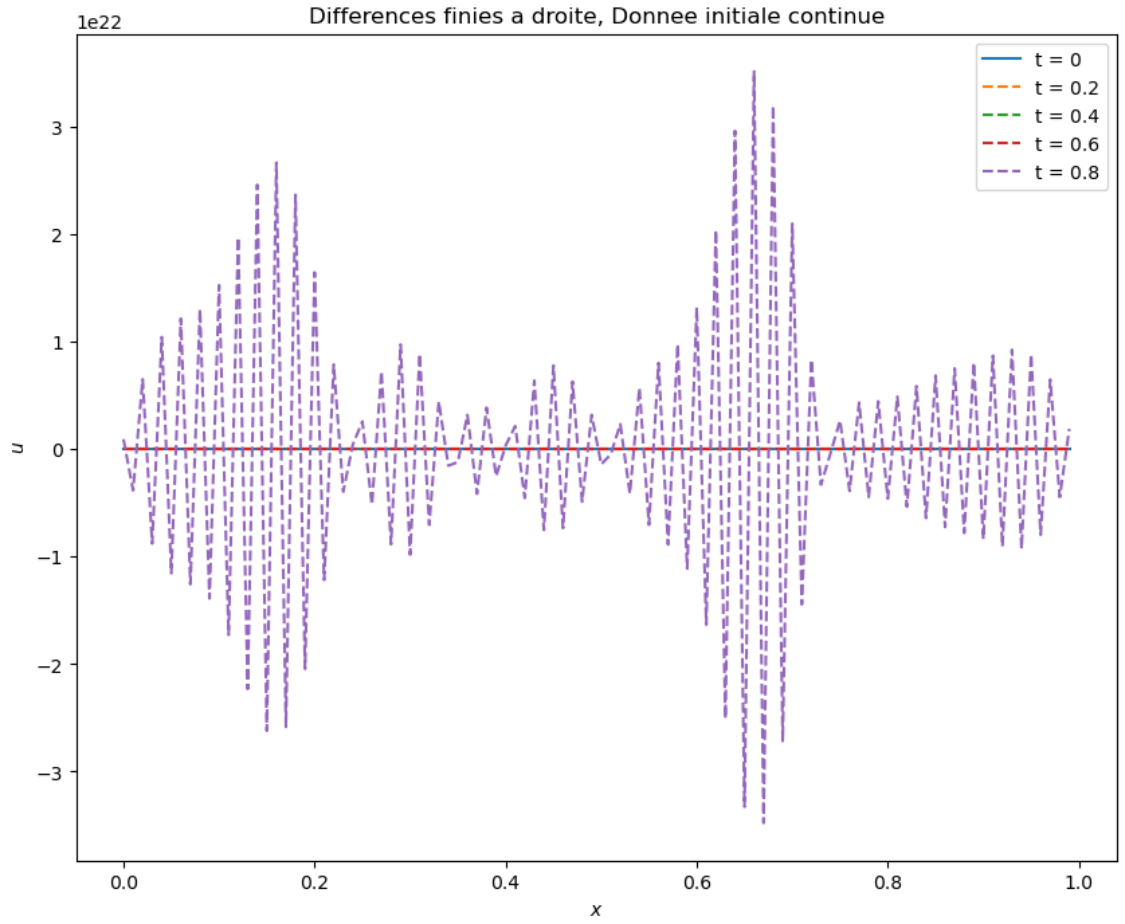
In [6]:
```
u0 = u0cont(x)
J = 100
a = 1
T = 1
dt = 1/100
U1 = DFG(J,T,a,dt,u0)

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

In [9]:
```
plt.clf()
plt.rcParams['figure.figsize'] = (10, 8)
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(20,N1,20):
    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(20,N2,20):
    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 $\lambda = \frac{a\Delta t}{\Delta x} = 1$.

In [21]:
```
u0 = u0cont(x)
J = 100
a = 1
T = 1
dt = 1/100
U2 = DFD(J,T,a,dt,u0)
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()
```



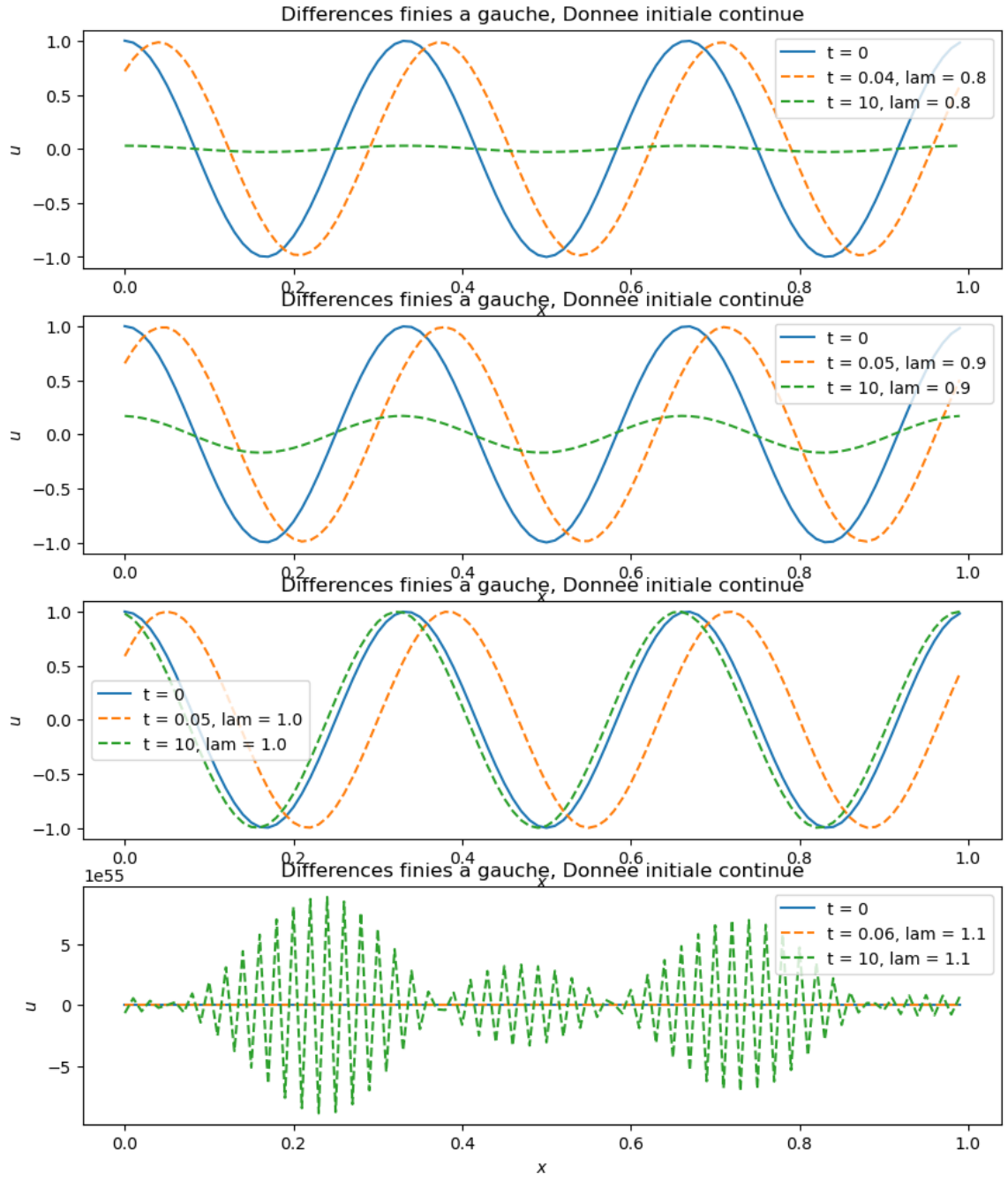
On observe ici une instabilité numérique de la solution approchée, qui prend des valeurs très grandes quand t s'approche de 1.

Question 2

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

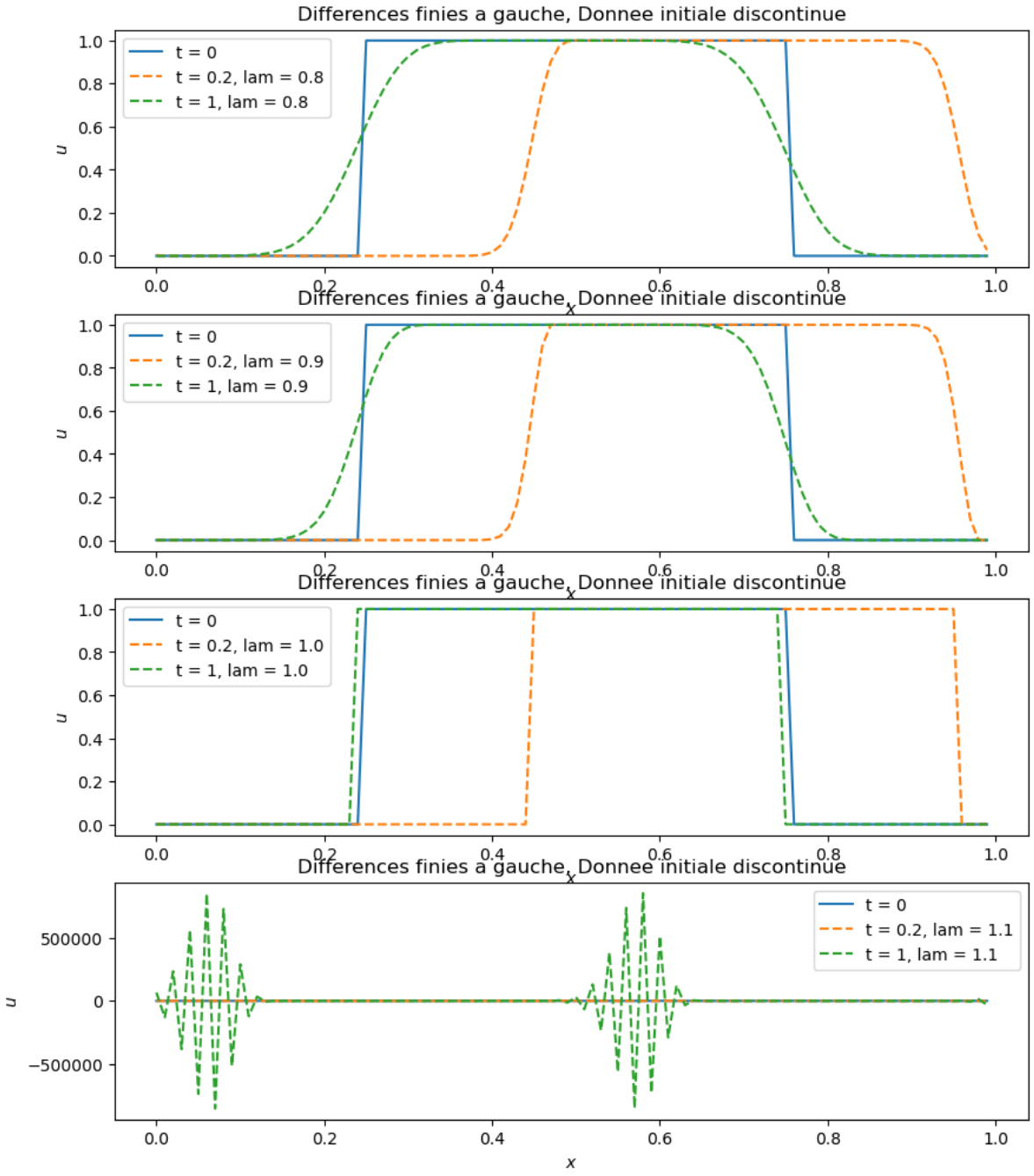
Schéma décentré à gauche

In [13]:
```
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(J,T,a,dt,u0)
    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()
```



In [23]:

```
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(J,T,a,dt,u0)
    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()
```



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 :

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

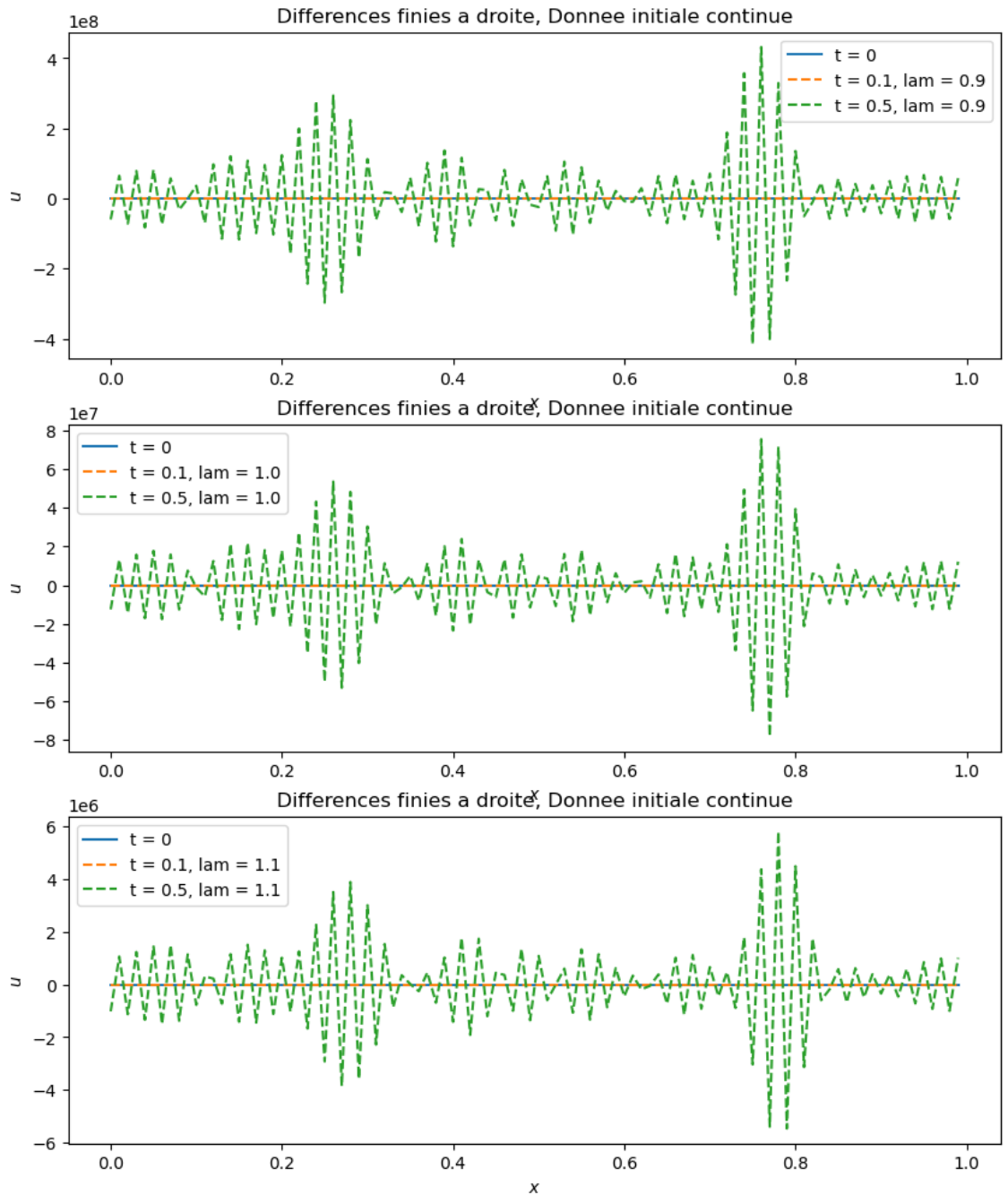
Lorsque $\lambda \in]0, 1[$, le schéma donne des résultats raisonnables. L'amplitude de l'oscillation de la solution régulière est atténuée avec le nombre d'itérations en temps. Pour l'autre test dans lequel la solution est discontinue, on observe un effet de régularisation. Ces deux effets sont dûs 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.

Schéma décentré à droite

In [19]:
```
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(J,T,a,dt,u0)
    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.

Question 3 : Analyse de stabilité

Voir corrigé

Prolongement : Schéma centré

Programmer le schéma différences finies centré défini par

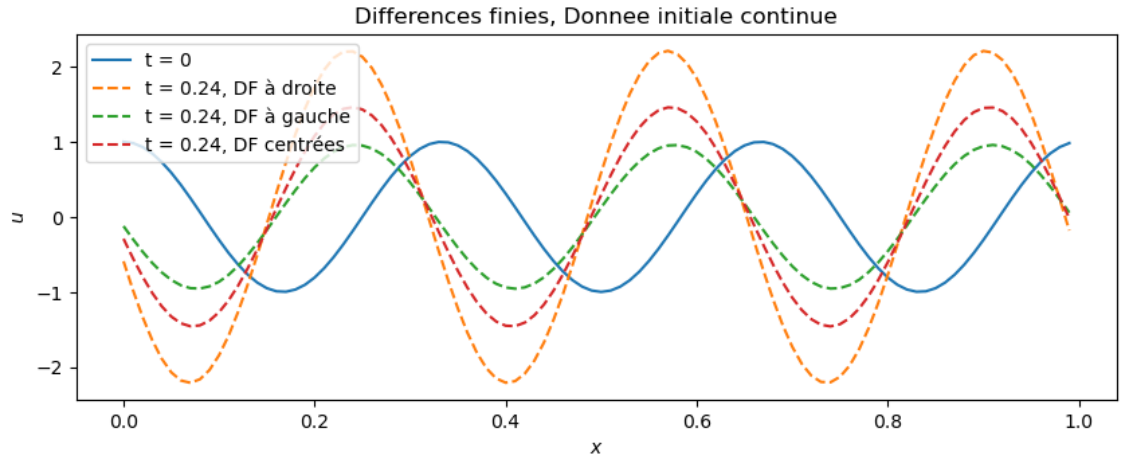
$$u_j^{n+1} = u_j^n - a\frac{\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n)$$

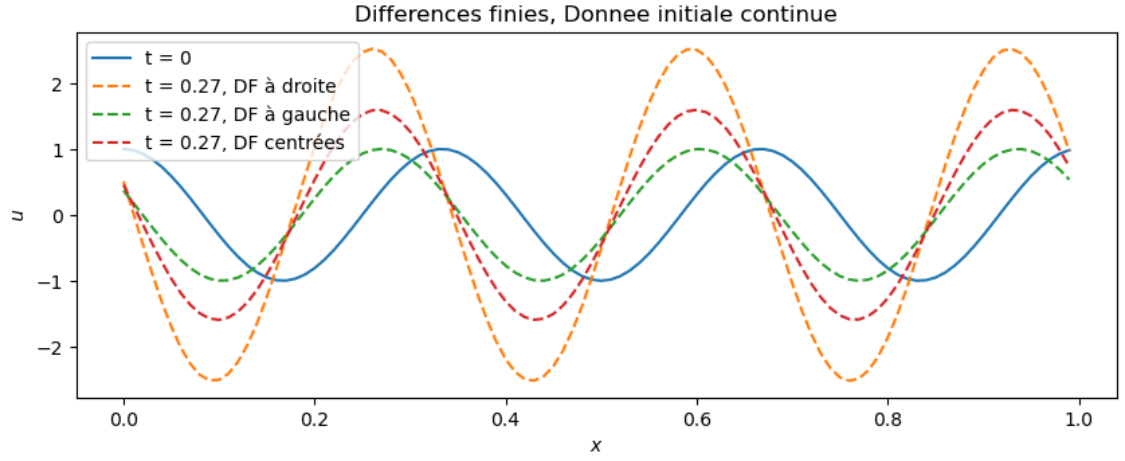
In [29]:
```
def DFC (J,T,a,dt,u0):
    # différences finies centrées ;
```

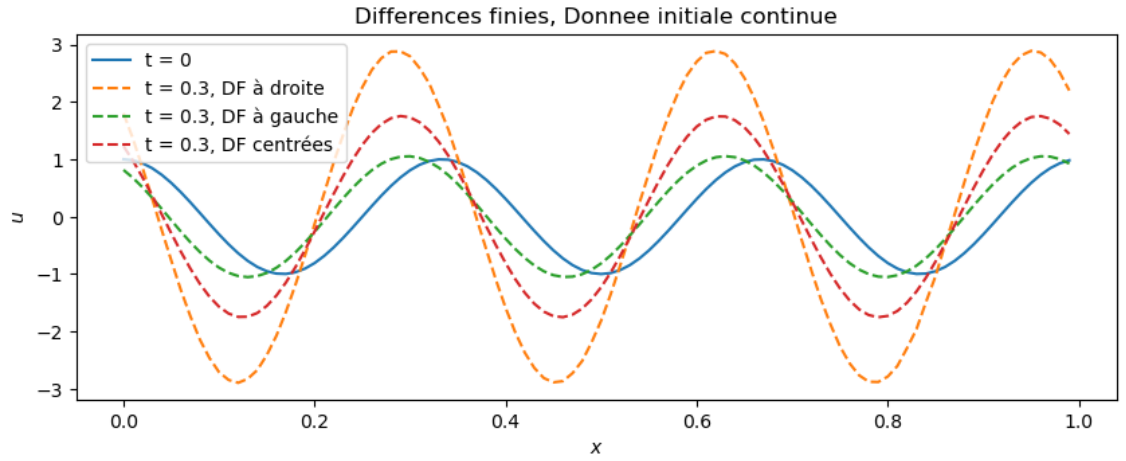
```
    # J = nb de sous-intervalles de la subdivision en x, T = temps maximal
    # a = vitesse du transport (positive), u0 = donnee initiale (J valeurs)
    dx = 1/J
    v1 = np.zeros_like(u0)
    v2 = v1.copy()
    N = floor(T/dt)
    U = np.zeros((N,J))
    U[0,:] = u0
    for n in range (1,N):
        v1[:-1] = U[n-1,1:]
        v1[-1] = U[n-1,0]
        v2[0] = U[n-1,-1]
        v2[1:] = U[n-1,:-1]
        U[n,:] = U[n-1,:] - a*dt/(2*dx)* (v1-v2)
    return U
```

In [32]:

```
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(J,T,a,dt,u0)
    U2 = DFG(J,T,a,dt,u0)
    U3 = DFC(J,T,a,dt,u0)
    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()
```

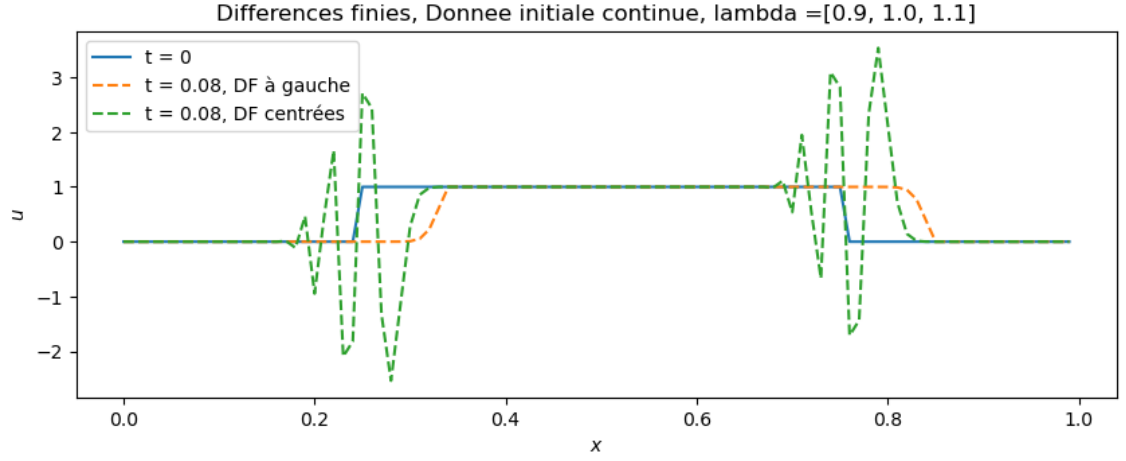


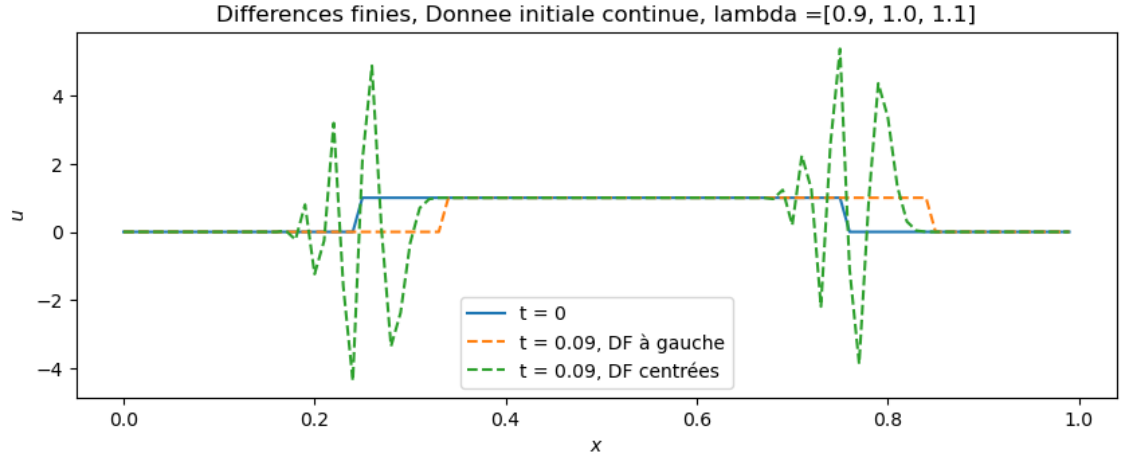


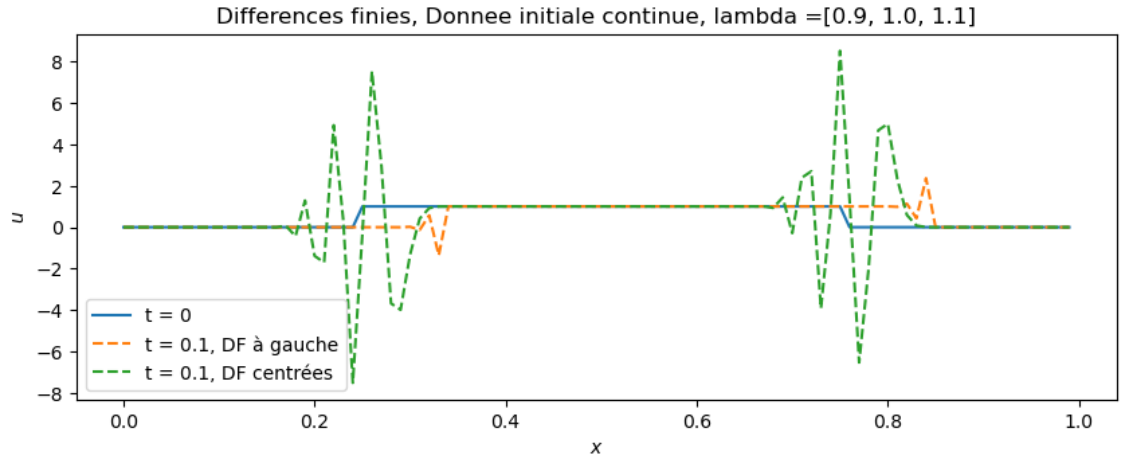


In [38]:

```
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(J,T,a,dt,u0)
    U2 = DFG(J,T,a,dt,u0)
    U3 = DFC(J,T,a,dt,u0)
    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()
```







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).

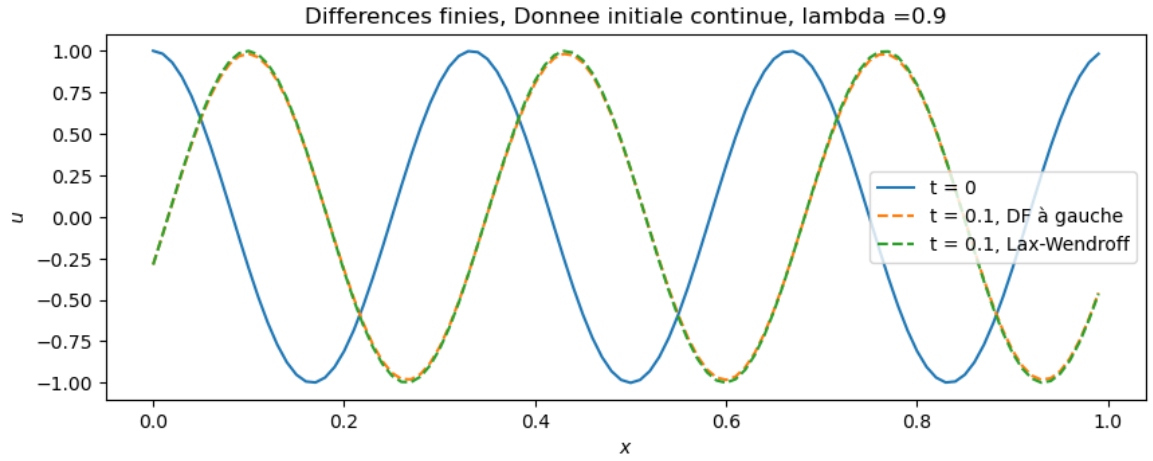
Schéma de Lax-Wendroff

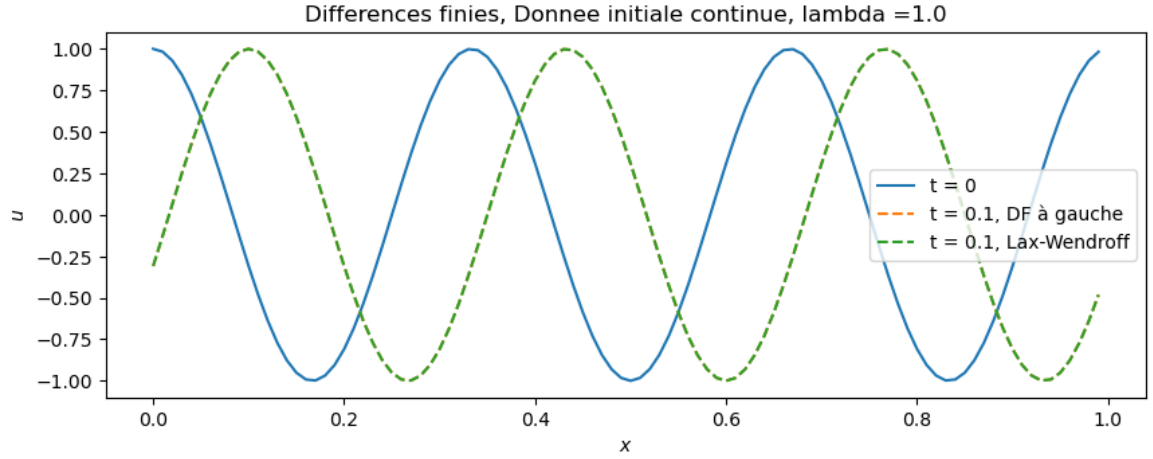
Programmer le schéma différences finies de Lax-Wendroff défini par

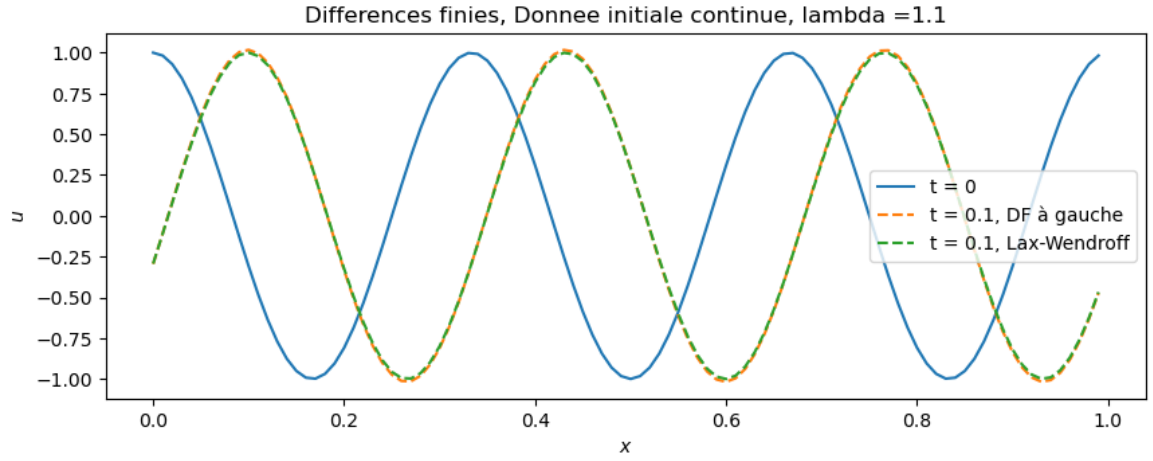
$$u_j^{n+1} = u_j^n - a\frac{\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) + a^2\frac{(\Delta t)^2}{2(\Delta x)^2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

In [40]:
```
def LW (J,T,a,dt,u0):
    # 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
    v1 = np.zeros_like(u0)
    v2 = v1.copy()
    N = floor(T/dt)
    U = np.zeros((N,J))
    U[0,:] = u0
    for n in range (1,N):
        v1[:-1] = U[n-1,1:]
        v1[-1] = U[n-1,0]
        v2[0] = U[n-1,-1]
        v2[1:] = U[n-1,:-1]
        U[n,:] = U[n-1,:] - a*dt/(2*dx)* (v1-v2) + a**2*dt**2/(2*dx**2) *(v1-2*U[n-1,:]+v2)
    return U
```

In [44]:
```
# 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(J,T,a,dt,u0)
    U2 = DFG(J,T,a,dt,u0)
    U4 = LW(J,T,a,dt,u0)
    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()
```

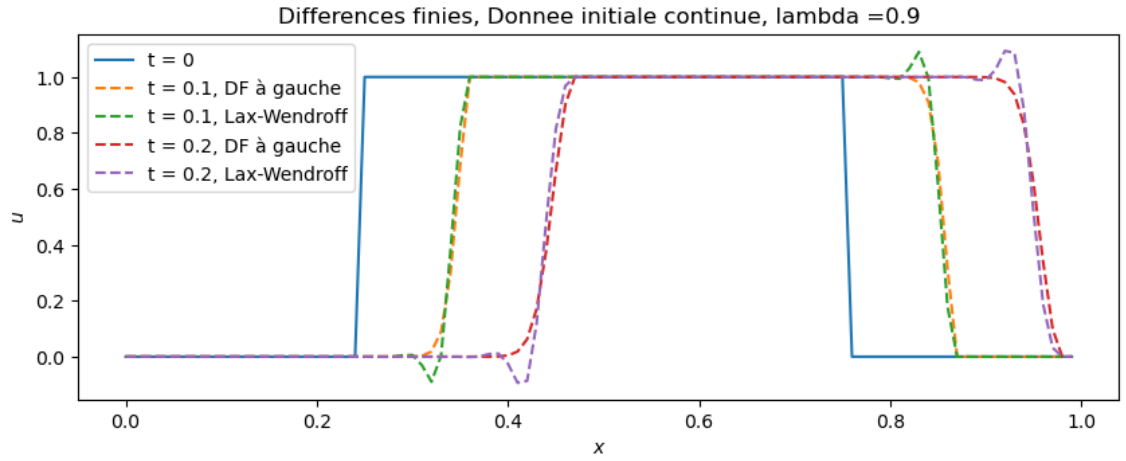


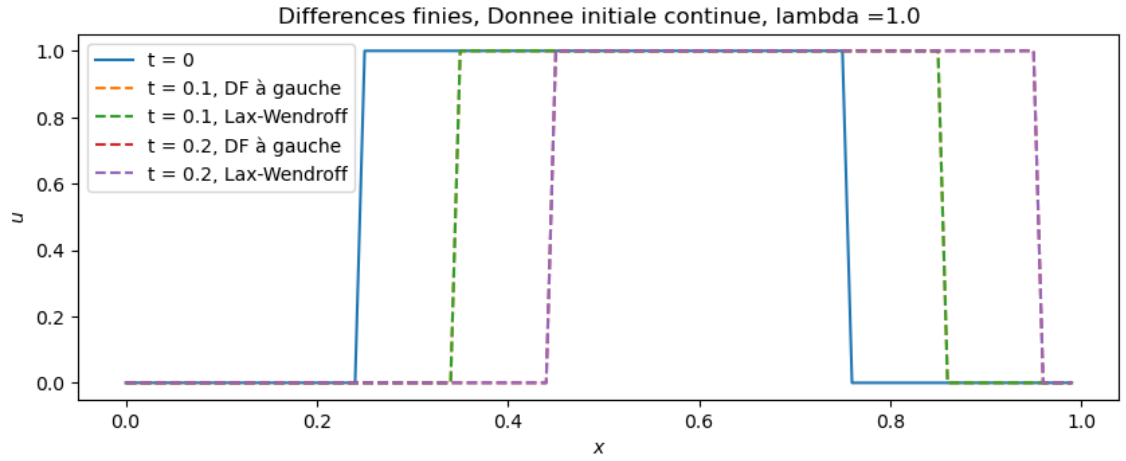


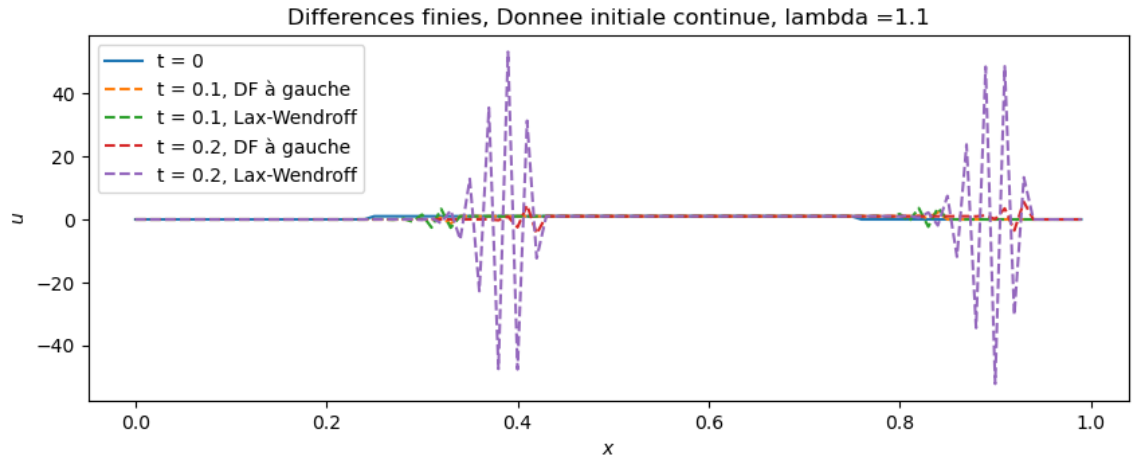


In [46]:

```
# 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(J,T,a,dt,u0)
    U2 = DFG(J,T,a,dt,u0)
    U4 = LW(J,T,a,dt,u0)
    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()
```







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).

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).

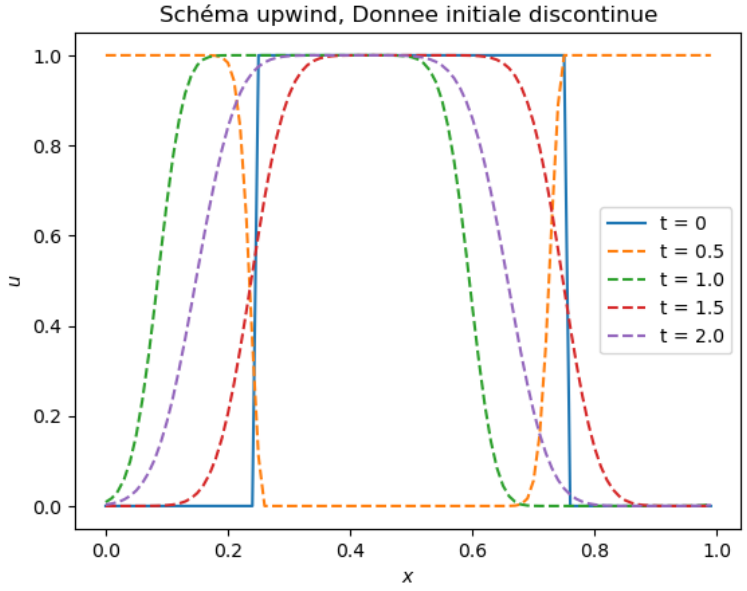
L'analyse de stabilité en norme 2 précédemment employée se trouve un peu complexifiée. On peut cependant s'appuyer sur le fait que le domaine en temps T est fixé et retenir un majorant $M > 0$ de $|a(t)|$ lorsque t parcourt $[0, T]$ et le pas de temps uniforme correspondant. En choisissant une suite 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 $t_n \leq T$.

In [65]:
```
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
```

In [103…]

```
# 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()
```



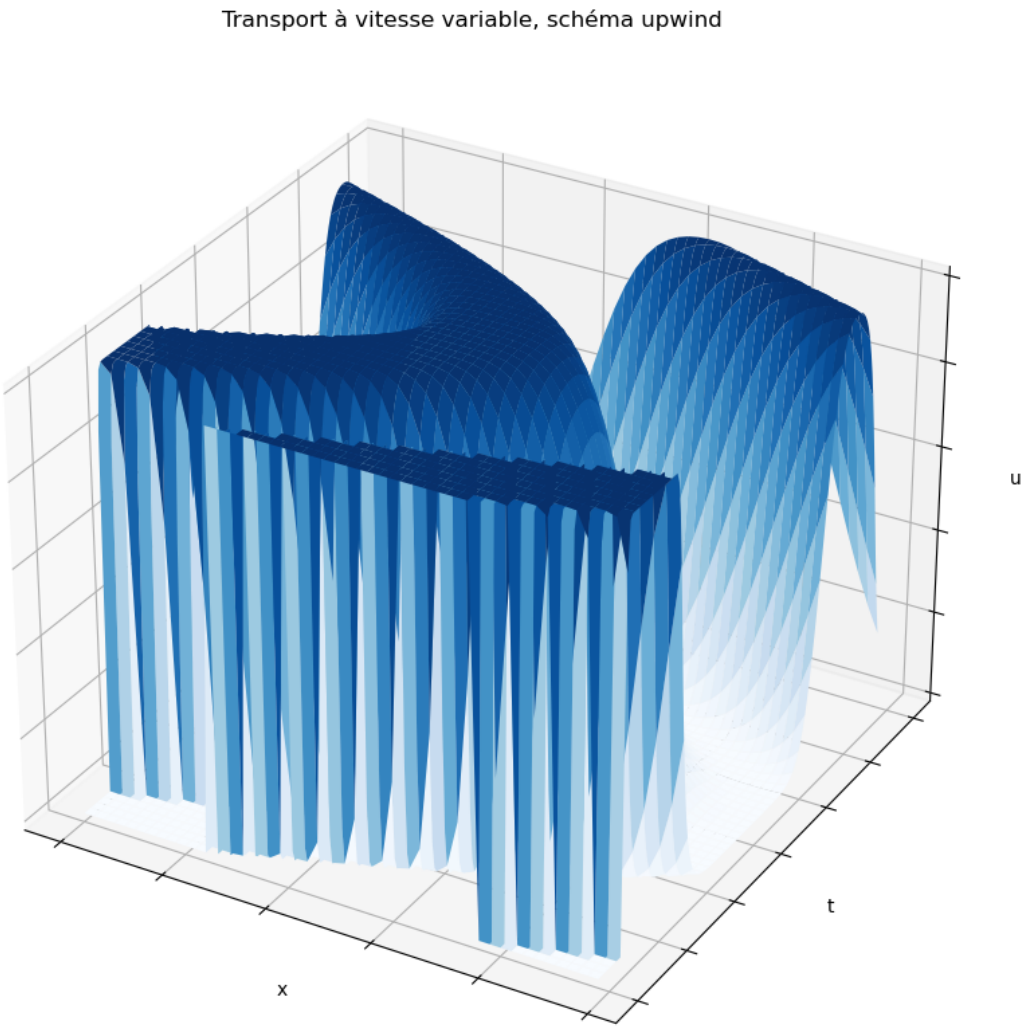
On observe (ou on devine) que le créneau se déplace d'abord vers la droite en ralentissant progressivement (jusqu'à $t = 1.5$ environ, en fait jusqu'à $t = \pi/2$) puis repart vers la gauche. Malgré le changement de signe de la vitesse, la solution numérique ne semble pas présenter d'instabilité. On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

In [98]:

```
# 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
```



In [ ]: