TP10 Différences finies pour l'équation de la chaleur

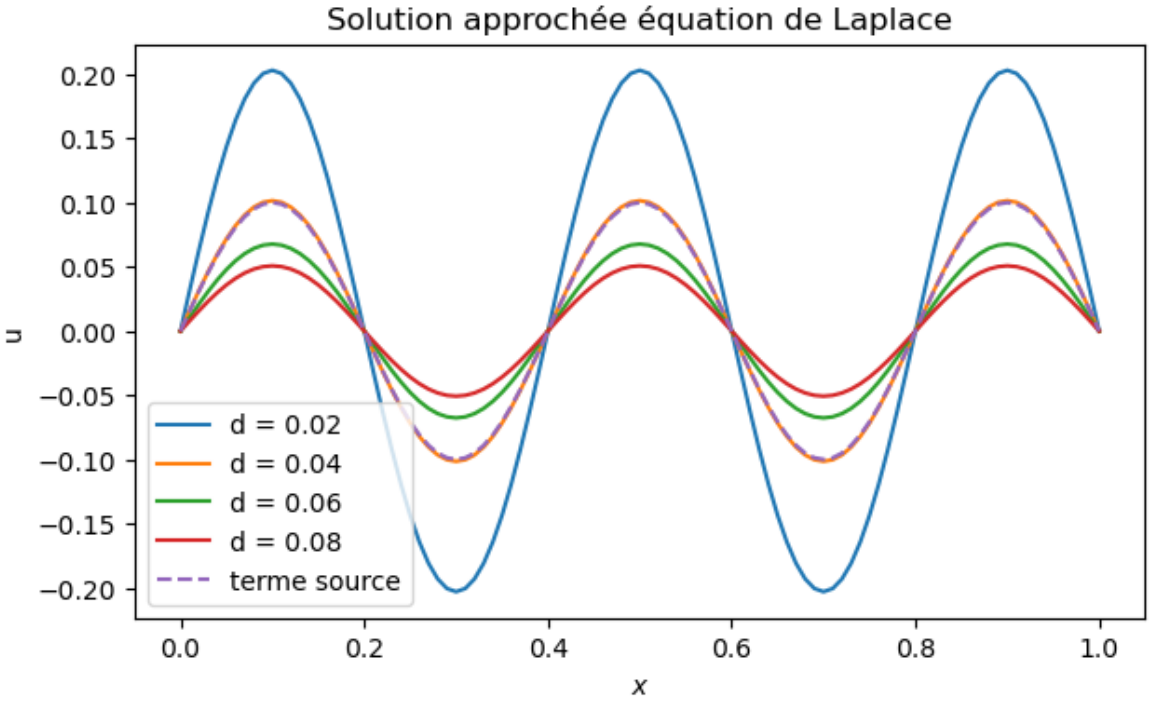
In [1]:
```
from math import *
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
```

Partie 1 : Équation de Laplace

In [2]:
```
def DFLaplace(f,J,d):
    #f : terme source, J : nb de points de discrétisation, d : coefficient de diff
    A = np.diag(2*np.ones(J-1)) - np.diag(np.ones(J-2),-1) - np.diag(np.ones(J-2),
    A = A * J**2
    x = np.linspace(0,1,J+1)
    F = f(x[1:-1])
    U = la.solve(d*A,F)
    return x,U
```

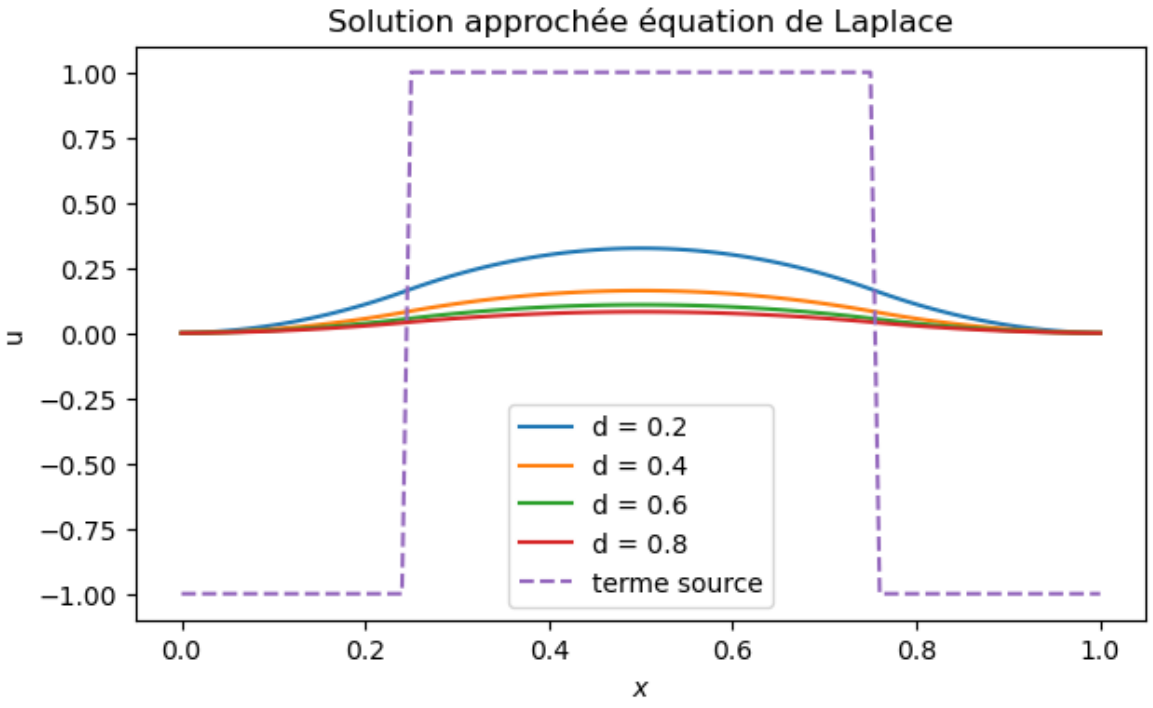
In [3]:
```
def f(x): #terme source continu
    return np.sin(5*pi*x)
```

In [4]:
```
U2 = np.zeros(101)
plt.figure(figsize = (7, 4))
for d in np.arange(0.02,0.1,0.02):
    x,U = DFLaplace(f,100,d)
    U2[1:-1] = U
    plt.plot(x,U2,label='d = '+str(round(d,3)))
plt.xlabel('$x$')
plt.ylabel('u')
plt.plot(x,f(x)/10,'--',label = 'terme source')
plt.legend()
plt.title('Solution approchée équation de Laplace')
plt.show()
```



Interprétation : Le terme source correspond à la chaleur apportée à un barreau métallique. La température aux deux extrémités du barreau est maintenue à 0. Les courbes en trait plein représentent la température du barreau à l'équilibre. On constate que plus le barreau est conducteur (d élevé) plus la température à l'équilibre est homogène et faible (l'énergie fournie est dissipée).

In [5]:
```
def f2(x): #terme source discontinu
        return np.array(-1+2*(x<=3./4)*(x>=1./4),dtype=float)

U2 = np.zeros(101)
plt.figure(figsize = (7, 4))
for d in np.arange(0.2,1,0.2):
    x,U = DFLaplace(f2,100,d)
    U2[1:-1] = U
    plt.plot(x,U2,label='d = '+str(round(d,3)))
plt.xlabel('$x$')
plt.ylabel('u')
plt.plot(x,f2(x),'--',label = 'terme source')
plt.legend()
plt.title('Solution approchée équation de Laplace')
plt.show()
```



Les courbes de température sont régulières en x malgré la discontinuité du terme source. Comme dans l'exemple précédent, plus d augmente, plus la température est homogène dans le barreau à l'équilibre.

Partie 2 : Équation de la chaleur

Schéma explicite

On cherche à approcher la solution instationnaire de l'EDP suivante:

$$\frac{\partial u(t,x)}{\partial t} = d\frac{\partial^2 u(t,x)}{\partial x^2} + s(x),$$

avec les conditions au bord $u(t,0) = u(t,1) = 0$ pour tout $t \geq 0$. En guise de discrétisation en espace du problème, on procède par différences finies avec un pas $\Delta x = 1/(N_x + 1)$. Une

version intermédiaire du problème est ainsi constitué d'un problème dit semi-discret, en l'occurence un système d'EDO posé dans $\mathbb{R}^{N_x}$:

$$U'(t) = -dAU(t) + S,$$

où l'on note $U(t) = (u_i(t))_{1 \leq i \leq N_x}$ tel que $u_i(t)$ approche $u(t, i\Delta x)$, et $u_0(t) = u_{N_x+1}(t) = 0$. Le second membre S vérifie $S_i = s(i\,\Delta x)$. La matrice A symétrique définie et positive déjà rencontrée maintes fois est celle du laplacien 1D:

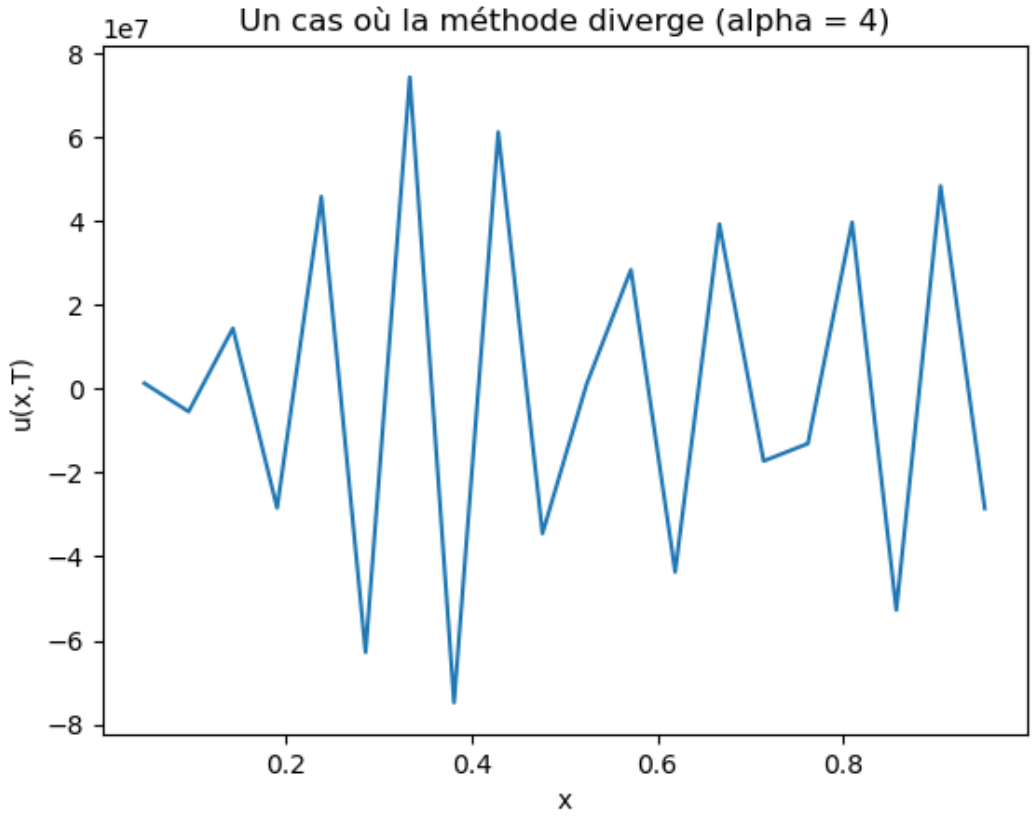
$$A = \frac{1}{\Delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 2 & -1 & 0 \\ \vdots & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix}.$$

In [6]:
```
def eqdiff(d,U,f,dx):
    # ici d est le coefficient de diffusion
    # après discrétisation on a remplacé notre équation de la chaleur par une équa
    # U' = -d A U + f dont on calcule ici le second membre
    # dx est le pas de discrétisation spatiale qui interviendra dans la matrice A.
    n = len(U)
    A = 1/(dx**2) * (2 * np.diag(np.ones(n),0) - np.diag(np.ones(n-1),1) - np.diag
    return - d*A.dot(U) + f


def S(x): # terme source
    return 100 * np.sin(5 * np.pi * x)

def S0(x) : # pas de terme source
    return 0

def U0(x): # donnée initiale
    return np.sin(np.pi * x) + 1 / 2 * np.sin(5 * np.pi * x)


def EulerExpliciteChaleur(d,t, U0, S, h, Nx, tf):
    # t est l'instant où la donnée initiale U0 est donnée, en pratique t = 0 dans
    # h est le pas de temps, le pas d'espace dx est calculé en fonction du nombre
    dx = 1 / (Nx + 1)
    X = dx * np.arange(1, Nx + 1)
    nbr_pas = int((tf-t) / h)
    U = np.zeros((Nx, nbr_pas + 1))
    U[:,0] = U0(X)
    for i in range (1, nbr_pas + 1):
        U[:,i] = U[:,i-1] + h * eqdiff(d,U[:,i-1],S(X),dx)
    return X,U
```

Testons cette méthode pour $d = 0.1$, $Nx = 20$ et un pas de temps $h = 0.1$. Cela correspond à

$$\alpha = d\Delta t/\Delta x^2 = 0.1 * 0.1 * 400 = 4.$$

In [7]:
```
X1,U1 = EulerExpliciteChaleur(0.1, 0, U0, S0, 0.1,  20, 2)
plt.plot(X1,U1[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,T)')
plt.title('Un cas où la méthode diverge (alpha = 4)')
plt.show()
```

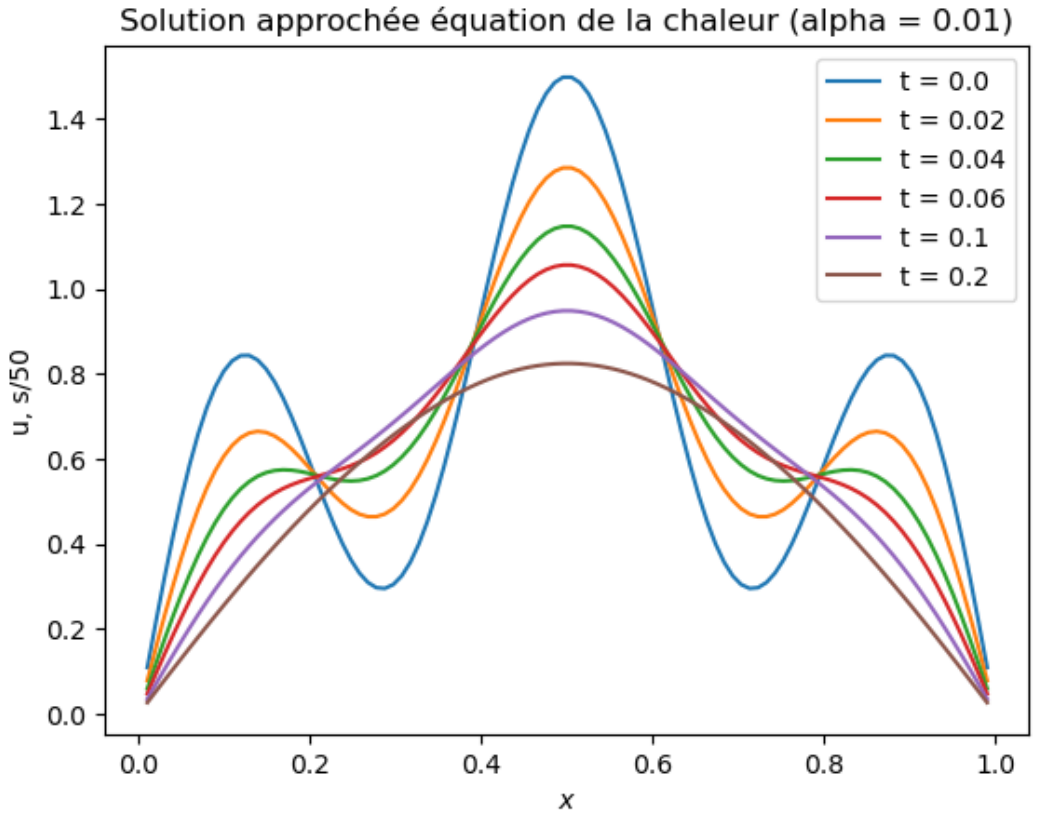


De toute évidence, la méthode diverge avec cette valeur de α. On va reprendre nos calculs en diminuant le pas de temps. Ci-dessous on a

$$\alpha = d\Delta t/\Delta x^2 = 10^{-6}/10^{-4} = 0.01.$$

In [8]:
```
X3,U3 = EulerExpliciteChaleur(0.1,0, U0, S0, 0.00001, 100, 1)
```

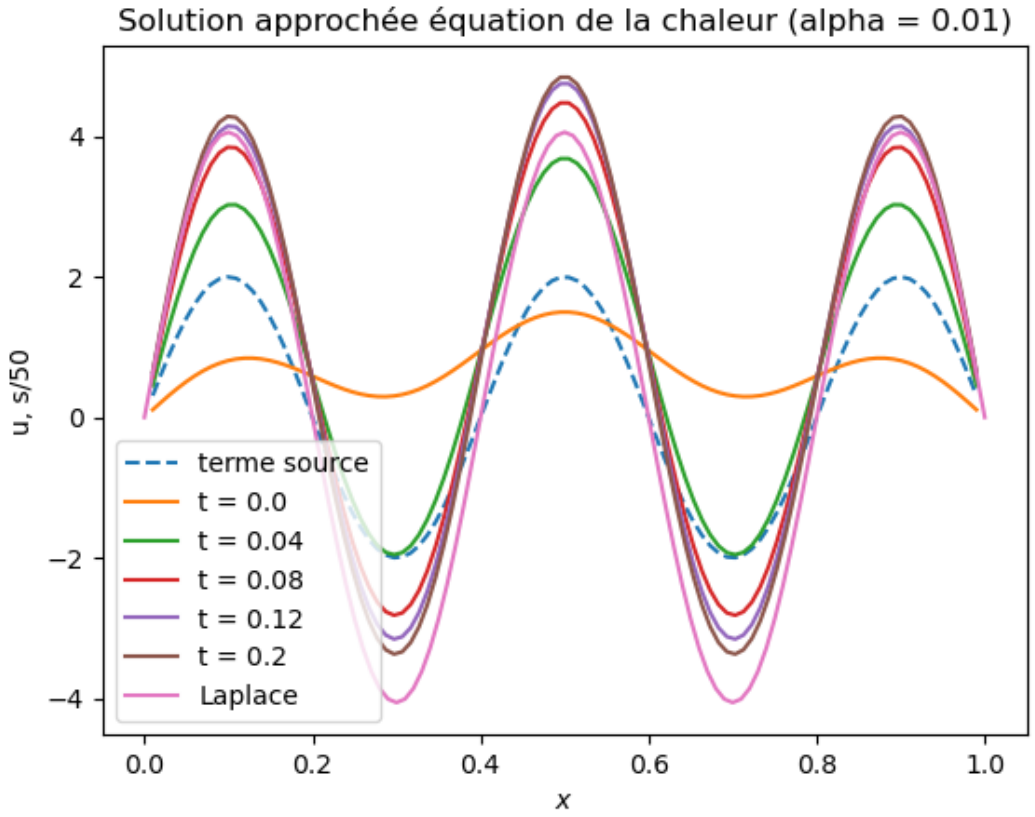
In [9]:
```
#plt.plot(X3,S(X3)/50, '--', label = 'terme source')
for i in [0,2000,4000,6000,10000, 20000]:
    plt.plot(X3,U3[:,i], label = 't = '+str(round(i*0.00001,3)))
plt.xlabel('$x$')
plt.ylabel('u, s/50')
plt.legend()
plt.title('Solution approchée équation de la chaleur (alpha = 0.01)')
plt.show()
```



Ici la méthode converge mais le calcul demande un temps CPU long à cause du pas de temps qui est très petit. Regardons maintenant ce qui se passe avec un terme source.

In [10]:
```
X2,U2 = EulerExpliciteChaleur(0.1,0, U0, S, 0.00001, 100, 1)
```

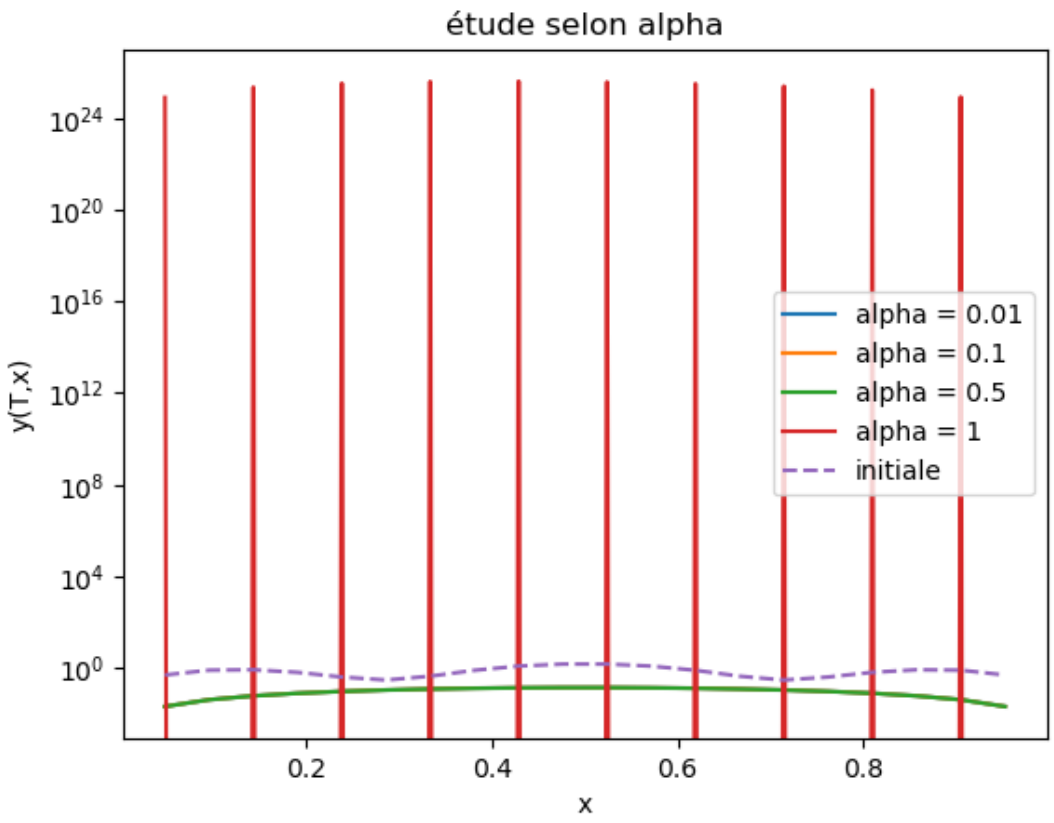
In [11]:
```
plt.plot(X2,S(X2)/50, '--', label = 'terme source')
for i in [0,4000,8000,12000, 20000]:
    plt.plot(X2,U2[:,i], label = 't = '+str(round(i*0.00001,3)))
plt.xlabel('$x$')
plt.ylabel('u, s/50')
x,U = DFLaplace(S,100,0.1)
UL = np.zeros(101)
UL[1:-1] = U
plt.plot(x,UL,label='Laplace')
plt.legend()
plt.title('Solution approchée équation de la chaleur (alpha = 0.01)')
plt.show()
```



Étude selon la valeur de α

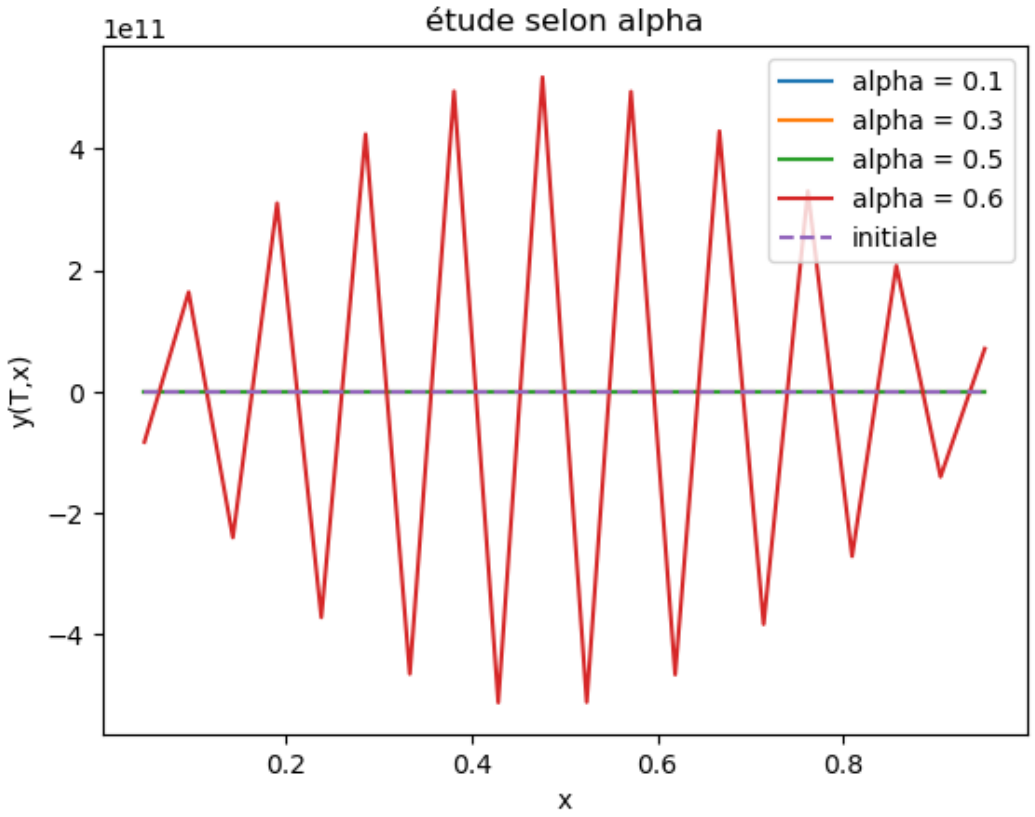
Nous faisons varier alpha entre 0.01 et 1 et nous observons la convergence de la méthode.

In [37]:
```
tab_alpha = [0.01,0.1,0.5,1]
Nx = 20
d = 0.1
tf = 2
for alpha in tab_alpha:
    h = alpha / (d*Nx**2)
    X,U = EulerExpliciteChaleur(d,0, U0, S0, h, Nx, tf)
    plt.semilogy(X,U[:,-1],label='alpha = '+str(alpha))
plt.semilogy(X,U0(X),'--',label='initiale')
plt.xlabel('x')
plt.ylabel('y(T,x)')
plt.legend()
plt.title('étude selon alpha')
plt.show()

#On trace en échelle logarithmique pour mieux distinguer ce qui se passe au voisin
```



On observe que toutes les courbes finales sont confondues sauf la courbe pour la valeur alpha = 1 qui fait diverger la méthode. On reprend pour alpha entre 0.1 et 0.6

In [12]:
```
tab_alpha = [0.1,0.3,0.5,0.6]
Nx = 20
d = 0.1
tf = 2
for alpha in tab_alpha:
    h = alpha / (d*Nx**2)
    X,U = EulerExpliciteChaleur(d,0, U0, S0, h, Nx, tf)
    plt.plot(X,U[:,-1],label='alpha = '+str(alpha))
plt.plot(X,U0(X),'--',label='initiale')
plt.xlabel('x')
plt.ylabel('y(T,x)')
plt.legend()
plt.title('étude selon alpha')
plt.show()
```



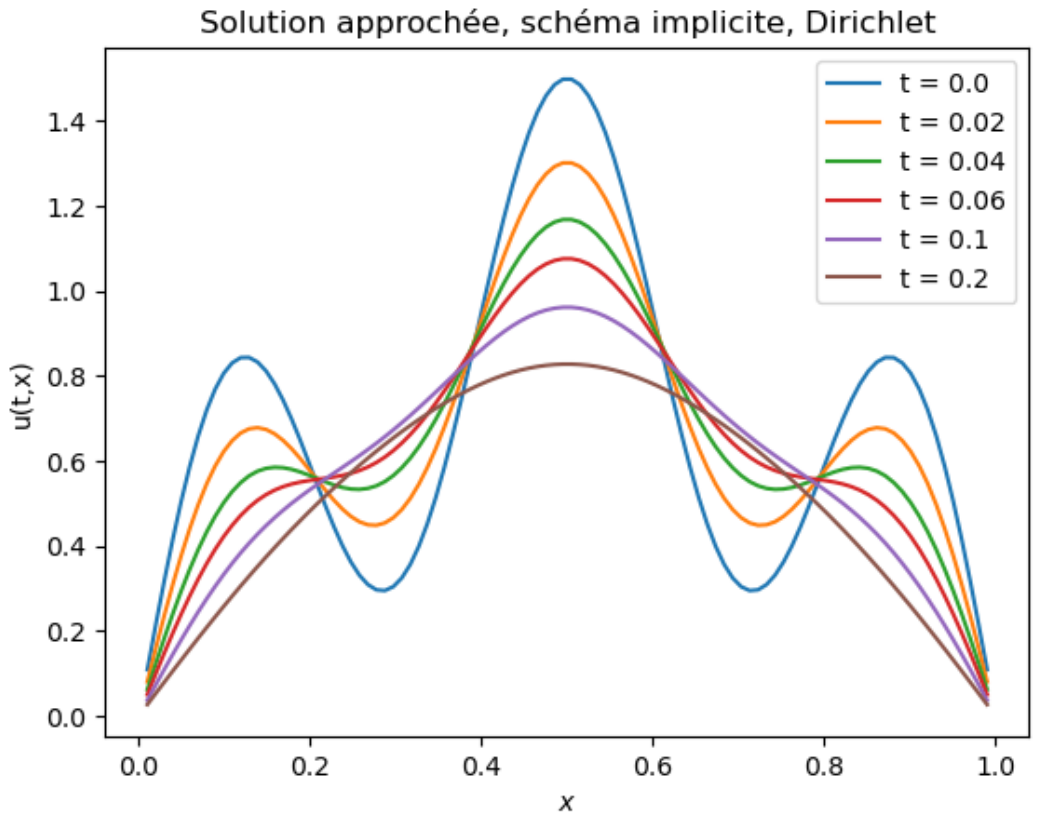
On observe que la méthode diverge dès que alpha > 0.5, ce qui est conforme à la théorie (cf étude de stabilité l^2 ou l^∞ du schéma).

Schéma implicite

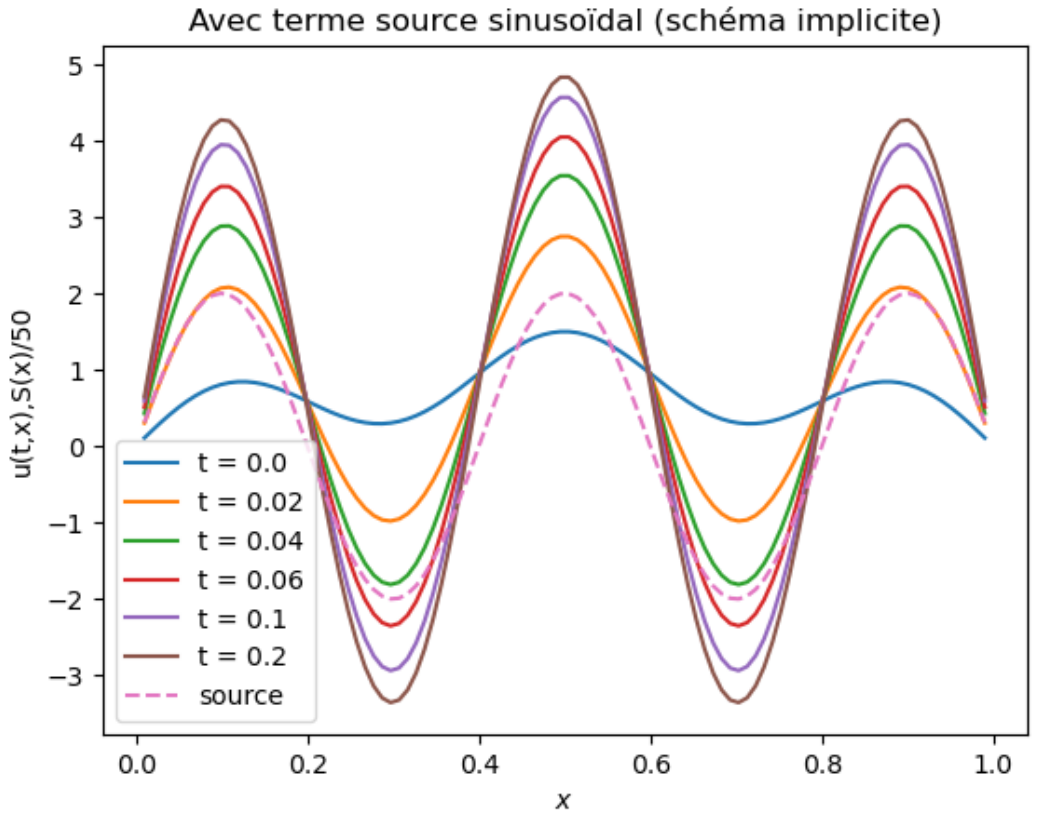
L'intérêt du schéma implicite est qu'il est stable sans condition CFL. On peut donc le mettre en oeuvre sans prendre des pas de temps très petits.

In [13]:
```
def S(x):
    return 100 * np.sin(5 * np.pi * x)


def U0(x):
    return np.sin(np.pi * x) + 1 / 2 * np.sin(5 * np.pi * x)


def EulerImpliciteChaleur(d,U0, S, h, Nx, tf):
    dx = 1 / (Nx + 1)
    X = dx * np.arange(1, Nx + 1)
    A = 1/(dx**2) * (2 * np.diag(np.ones(Nx),0) - np.diag(np.ones(Nx-1),1) - np.di
    nbr_pas = int(tf / h)
    U = np.zeros((Nx, nbr_pas + 1))
    U[:,0] = U0(X)
    for i in range (1, nbr_pas + 1):
        U[:,i] = la.solve(np.eye(Nx) + h *d* A,U[:,i-1] + h * S(X))
    return X,U
```

In [14]:
```
X3,U3 = EulerImpliciteChaleur(d,U0, S0, 0.01, 100, 1)
for i in [0,2,4,6,10, 20]:
    plt.plot(X3,U3[:,i], label = 't = '+str(round(i*0.01,3)))
plt.xlabel('$x$')
plt.ylabel('u(t,x)')
plt.legend()
plt.title('Solution approchée, schéma implicite, Dirichlet')
plt.show()
```



In [15]:
```
X3,U3 = EulerImpliciteChaleur(d,U0, S, 0.01, 100, 1)
for i in [0,2,4,6,10, 20]:
    plt.plot(X3,U3[:,i], label = 't = '+str(round(i*0.01,3)))
plt.xlabel('$x$')
plt.plot(X3,S(X3)/50,'--',label='source')
plt.ylabel('u(t,x),S(x)/50')
plt.legend()
plt.title('Avec terme source sinusoïdal (schéma implicite)')
plt.show()
```



Numériquement la solution semble converger vers la solution de l'équation de Laplace.

Cas de conditions de Neumann au bord

On impose maintenant la condition au bord

$$\frac{\partial y}{\partial x}(t,0) = \frac{\partial y}{\partial x}(t,1) = 0$$

qui correspond à une condition de "non-flux" (pas de perte de chaleur au bord).

On considère toujours une subdivision régulière en espace avec un pas $\Delta x = 1/(N_x+1)$. Le problème semi-discret est toujours un système d'EDO posé dans $\mathbb{R}^{N_x}$:

$$U'(t) = -dA_2U(t) + S,$$

où l'on note $U(t) = (u_i(t))_{1\leq i\leq N_x}$ tel que $u_i(t)$ approche $u(t, i\Delta x)$. Au lieu de poser $u_0(t) = u_{N_x+1}(t) = 0$, on impose maintenant $u_0(t) = u_1(t)$ et $u_{N_x+1}(t) = u_{N_x}(t)$. Ceci conduit à remplacer la matrice A du laplacien 1D habituelle par la matrice :

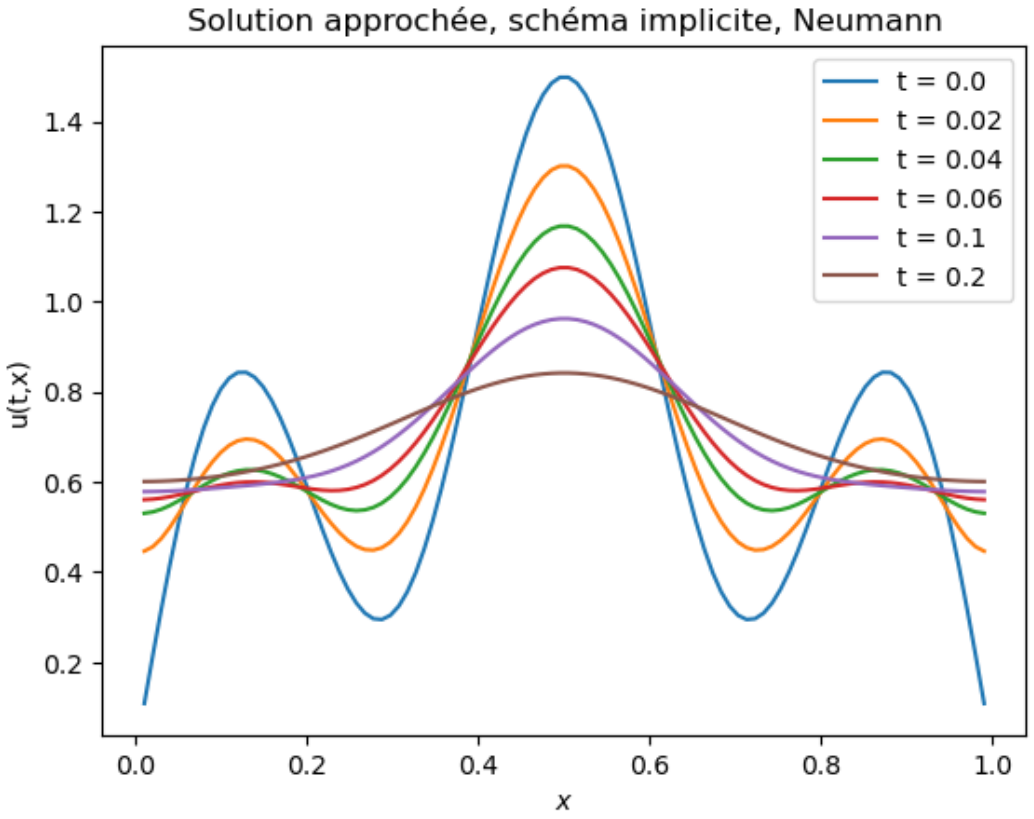
$$A_2 = \frac{1}{\Delta x^2}\begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 2 & -1 & 0 \\ \vdots & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 1 \end{pmatrix}.$$

In [16]:
```
def eqdiff_Neu(d,U,f,dx):
    # après discrétisation on a remplacé notre équation de la chaleur par une équa
    # U' = -d A2 U + f dont on calcule ici le second membre
    # dx est le pas de discrétisation spatiale qui interviendra dans la matrice A.
    n = len(U)
    A2 = 1/(dx**2) * (2 * np.diag(np.ones(n),0) - np.diag(np.ones(n-1),1) - np.dia
    A2[0,0] = 1/(dx**2)
    A2[n,n] = 1/(dx**2)
    return - d*A2.dot(U) + f

def EulerExpliciteChaleur_Neu(d,t, U0, S, h, Nx, tf):
    # t est l'instant où la donnée initiale U0 est donnée, en pratique t = 0 dans
    # h est le pas de te mps, le pas d'espace dx est calculé en fonction du nombre
    dx = 1 / (Nx + 1)
    X = dx * np.arange(1, Nx + 1)
    nbr_pas = int((tf-t) / h)
    U = np.zeros((Nx, nbr_pas + 1))
    U[:,0] = U0(X)
    for i in range (1, nbr_pas + 1):
        U[:,i] = U[:,i-1] + h * eqdiff_Neu(d,U[:,i-1],S(X),dx)
    return X,U
```

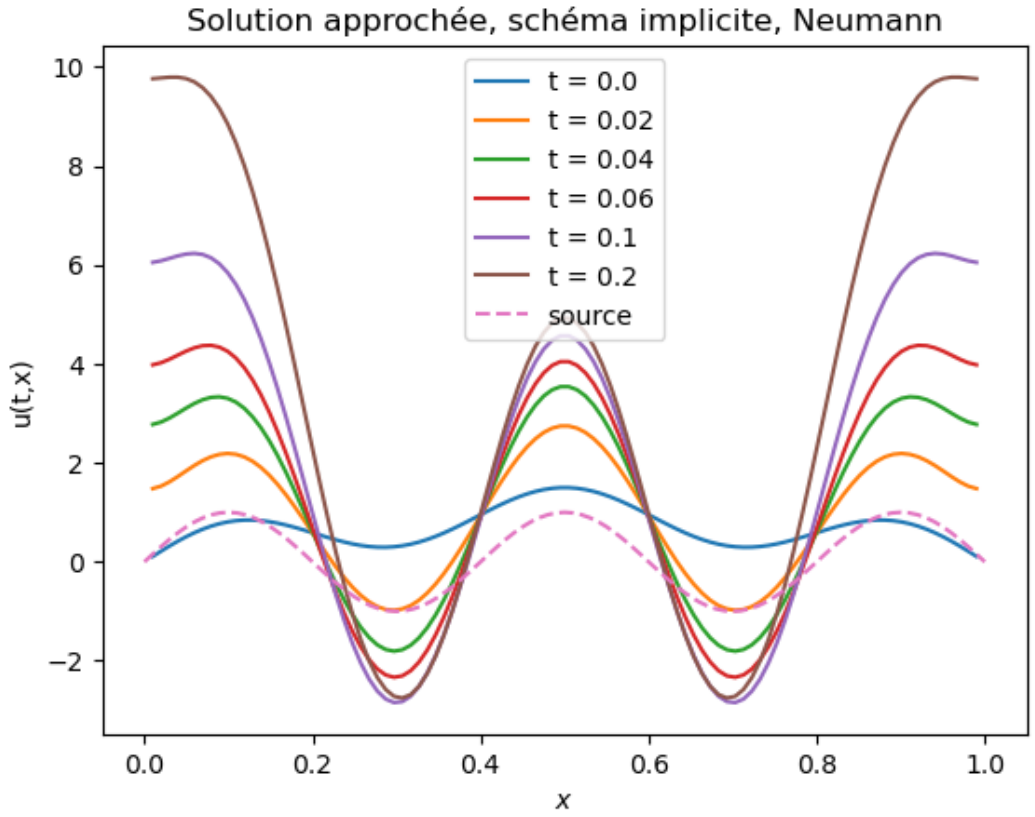
In [17]:
```
def EulerImpliciteChaleur_Neu(d,U0, S, h, Nx, tf):
    dx = 1 / (Nx + 1)
    X = dx * np.arange(1, Nx + 1)
    A2 = 1/(dx**2) * (2 * np.diag(np.ones(Nx),0) - np.diag(np.ones(Nx-1),1) - np.d
    A2[0,0] = 1/(dx**2)
    A2[Nx-1,Nx-1] = 1/(dx**2)
    nbr_pas = int(tf / h)
    U = np.zeros((Nx, nbr_pas + 1))
    U[:,0] = U0(X)
```

```
    for i in range (1, nbr_pas + 1):
        U[:,i] = la.solve(np.eye(Nx) + h * d * A2,U[:,i-1] + h * S(X))
    return X,U
```

In [18]:
```
X4,U4 = EulerImpliciteChaleur_Neu(d,U0, S0, 0.01, 100, 1)
for i in [0,2,4,6,10, 20]:
    plt.plot(X4,U4[:,i], label = 't = '+str(round(i*0.01,3)))
plt.xlabel('$x$')
plt.ylabel('u(t,x)')
plt.legend()
plt.title('Solution approchée, schéma implicite, Neumann')
plt.show()
```



In [19]:
```
X4,U4 = EulerImpliciteChaleur_Neu(d,U0, S, 0.01, 100, 1)
for i in [0,2,4,6,10, 20]:
    plt.plot(X4,U4[:,i], label = 't = '+str(round(i*0.01,3)))
plt.xlabel('$x$')
plt.ylabel('u(t,x)')
plt.plot(x,f(x),'--',label='source')
plt.legend()
plt.title('Solution approchée, schéma implicite, Neumann')
plt.show()
```



Cas de conditions au bord périodiques

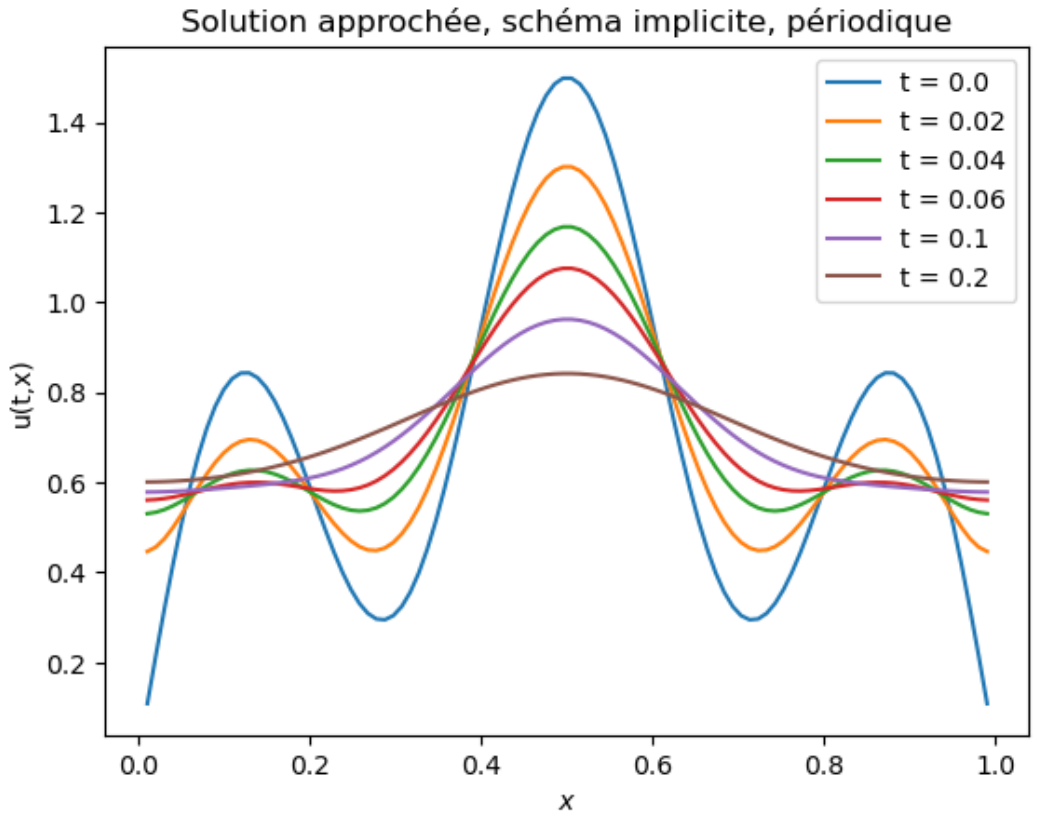
Cette fois la matrice A est remplacée par

$$A_3 = \frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & -1 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 2 & -1 & 0 \\ \vdots & & \ddots & \ddots & \ddots & -1 \\ -1 & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix}.$$

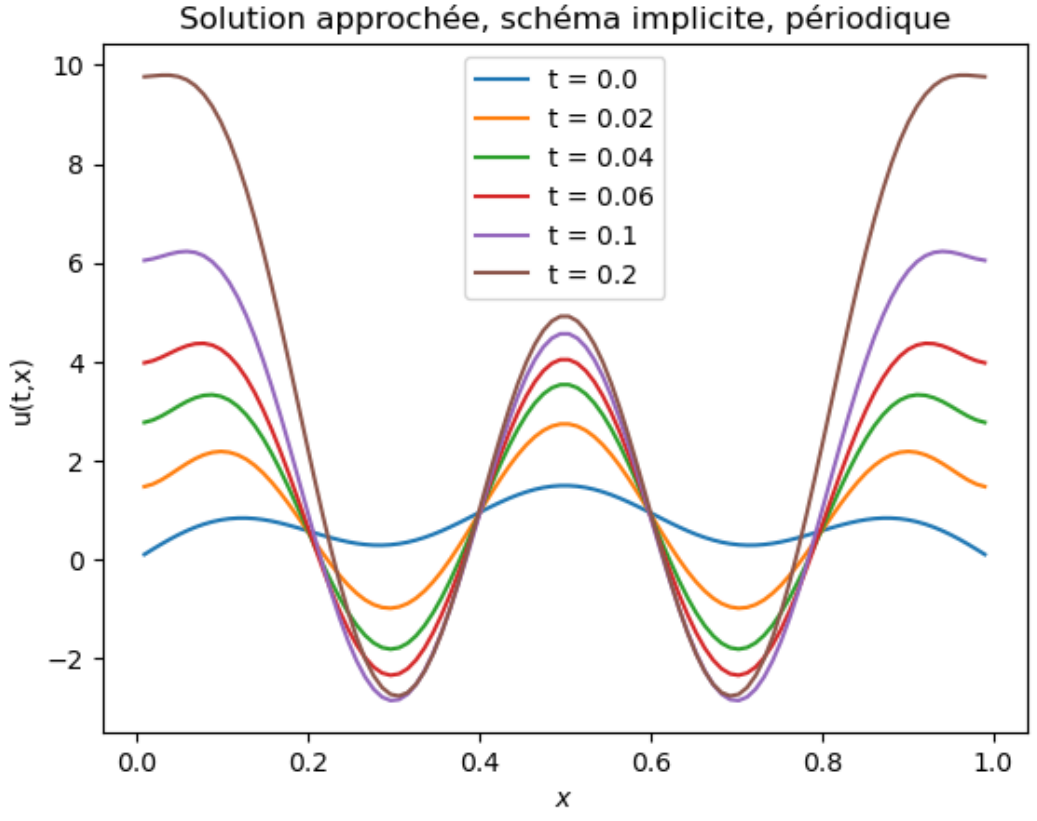
In [20]:
```
def EulerImpliciteChaleur_per(d,U0, S, h, Nx, tf):
    dx = 1 / (Nx + 1)
    X = dx * np.arange(1, Nx + 1)
    A3 = 1/(dx**2) * (2 * np.diag(np.ones(Nx),0) - np.diag(np.ones(Nx-1),1) - np.d
    A3[0,Nx-1] = -1/(dx**2)
    A3[Nx-1,0] = -1/(dx**2)
    nbr_pas = int(tf / h)
    U = np.zeros((Nx, nbr_pas + 1))
    U[:,0] = U0(X)
    for i in range (1, nbr_pas + 1):
        U[:,i] = la.solve(np.eye(Nx) + h * d * A3,U[:,i-1] + h * S(X))
    return X,U
```

In [21]:
```
X5,U5 = EulerImpliciteChaleur_per(d,U0, S0, 0.01, 100, 1)
for i in [0,2,4,6,10, 20]:
    plt.plot(X5,U5[:,i], label = 't = '+str(round(i*0.01,3)))
plt.xlabel('$x$')
plt.ylabel('u(t,x)')
plt.legend()
```

```
plt.title('Solution approchée, schéma implicite, périodique')
plt.show()
```



In [22]:
```
X5,U5 = EulerImpliciteChaleur_per(d,U0, S, 0.01, 100, 1)
for i in [0,2,4,6,10, 20]:
    plt.plot(X5,U5[:,i], label = 't = '+str(round(i*0.01,3)))
plt.xlabel('$x$')
plt.ylabel('u(t,x)')
plt.legend()
plt.title('Solution approchée, schéma implicite, périodique')
plt.show()
```



In [ ]: