Projet 2, Déformation d'un toit et équation de la chaleur

Zouiche Omar et David Czarnecki

7 Févrirer 2021

Problème On s'interesse à une EDP elliptique. On a le toit d'un hangar et on veut en savoir plus sur sa déformation verticale sous chargement, on a les données suivantes : - Le toit est un carré de côté $L = 10m$ - La déformation du toit est exprimée par la fonction $u(x, y)$ - Le chargement est exprimée par la fonction $f(x)$ - La rigidité du toit est $k > 0$ en N/m - La déformation est modélisée par :

$$-\Delta u = \frac{f(x)}{k}$$

- Données suplémentaires :

$$f(x)/k = 1 = 0.05(L - x) * x$$

On sait aussi que le toit est en appui sur tout l'axe y, donc sa déformation sur cet axe est constante i.e

$$\frac{\partial^2 u(x,y)}{\partial y^2} = 0$$

, la modélisation de la déformation devient :

$$\frac{\partial^2 u(x,y)}{\partial x^2} = \frac{f(x)}{k}$$

On peut remarquer c'est un cas particulier de l'équation de la chaleur quand

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

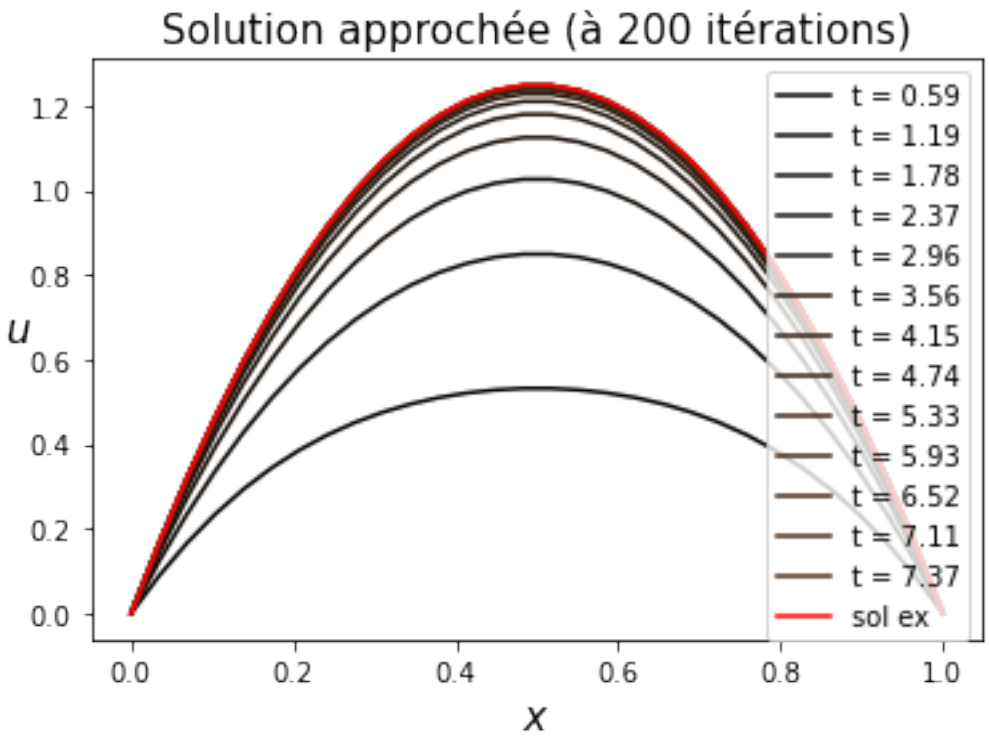
La méthode pour résoudre une équation de la chaleur nous donne une partie de la solution. Nous allons modifier le code heat1d.py pour trouver une solution

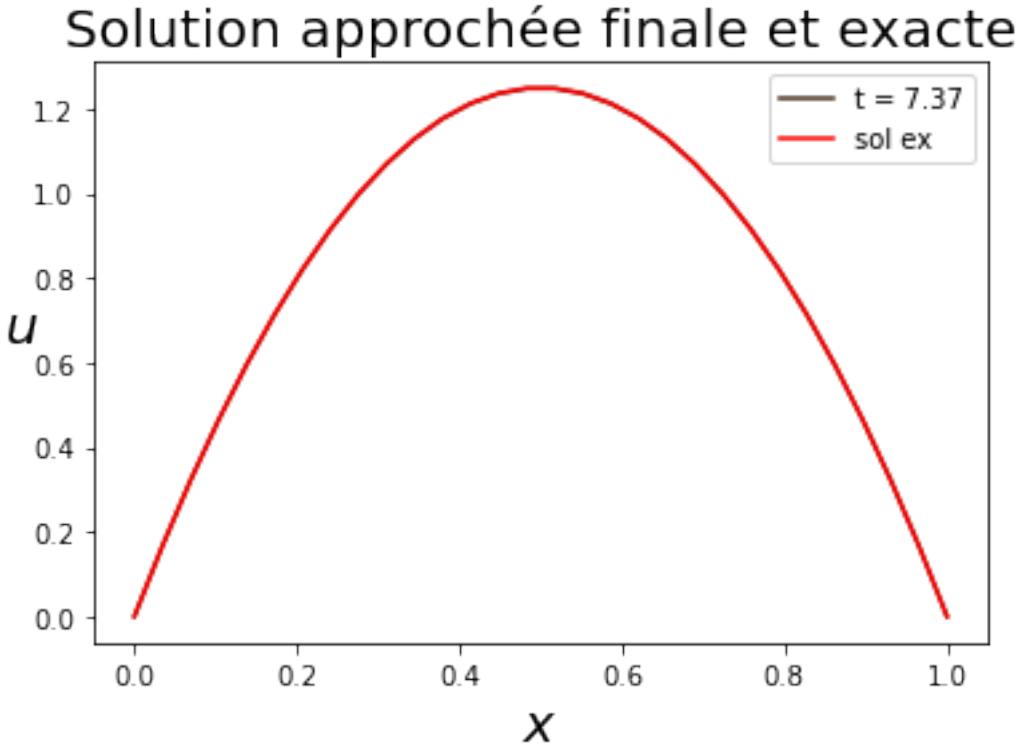
[4]:
```
import numpy as np
import matplotlib.pyplot as plt
import math


# PHYSICAL PARAMETERS
K = 0.1     #Diffusion coefficient
L = 1.0     #Domain size
Time = 20.  #Integration time


# NUMERICAL PARAMETERS
```

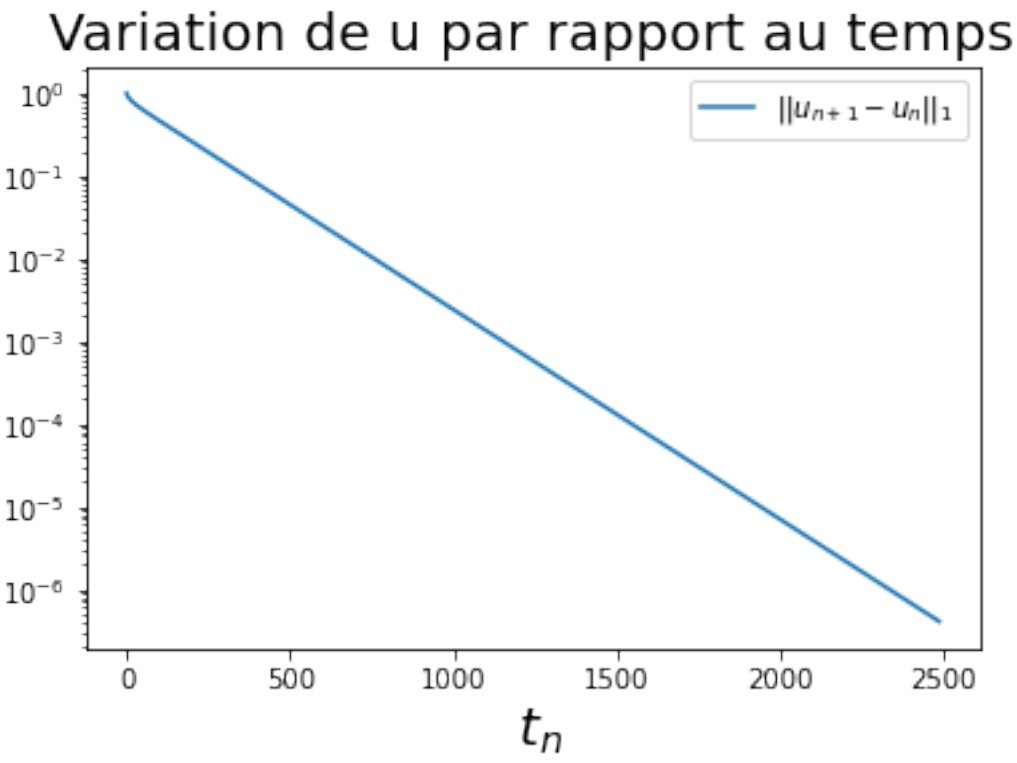
```
NX = 30    #Number of grid points
NT = 10000   #Number of time steps
ifre=200
eps=0.001
dx = L/(NX-1)  #Grid step (space)
dt = dx**2/(4*K+dx**2)   #Grid step (time)
print('Number of time steps :',NT,' dt = ',dt)

# EXACT SOLUTION
def sol_ex(x):
    return (-x**2)/(2*K) + (L*x)/(2*K)

#------------- MAIN PROGRAM -------------#

# Initialisation
x = np.linspace(0.0,1.0,NX)
T = np.zeros((NX)) #np.sin(2*np.pi*x)
F = np.ones((NX))
rest = []
RHS = np.ones((NX))
plt.figure()

# Main loop en temps
n=0
res=1
res0=1
while(n<NT  and res>10**(-10)):
    n+=1
# Discretization of the second order derivative (Laplacian)
    res=0
    for j in range (1, NX-1):
        RHS[j] = dt*(K*(T[j-1]-2*T[j]+T[j+1])/(dx**2)+F[j])
        res+=(RHS[j])**2

    for j in range (1, NX-1):
        T[j] += RHS[j]

    if (n == 1 ):
        res0=math.sqrt(res)

    rest.append(res)


    if (n%ifre == 0 ):
        #print(n,res)
        plotlabel = "t = %1.2f" %(n * dt)
```

```
        plt.plot(x,T, label=plotlabel,color = plt.get_cmap('copper')(float(n)/
↪NT))


plotlabel = "t = %1.2f" %(n * dt)
plt.plot(x,T, label=plotlabel,color = plt.get_cmap('copper')(float(n)/NT))
plt.plot(x,sol_ex(x),label = 'sol ex', color = 'r')
plt.xlabel(u'$x$', fontsize=15)
plt.ylabel(u'$u$', fontsize=15, rotation=0)
plt.title(u'Solution approchée (à 200 itérations)', fontsize = 15)
plt.legend()

# On trace la solution exacte et finale
# On trace ||u_n+1 _ u_n||
plt.figure()
plt.plot(x,T, label=plotlabel,color = plt.get_cmap('copper')(float(n)/NT))
plt.plot(x,sol_ex(x),label = 'sol ex', color = 'r')
plt.xlabel(u'$x$', fontsize=20)
plt.ylabel(u'$u$', fontsize=20, rotation=0)
plt.title(u"Solution approchée finale et exacte", fontsize = 20)
plt.legend()

plt.figure()
plt.plot(np.array(rest/rest[0]), label = "$|| u_{n+1} - u_n ||_1$ ")
plt.xlabel(' $t_n$', fontsize = 20)
plt.yscale('log')
plt.legend()
plt.title("Variation de u par rapport au temps", fontsize = 20)

plt.show()
```

```
Number of time steps : 10000  dt =  0.002963841138114997
```


Solution approchée (à 200 itérations)
t = 0.59
t = 1.19
t = 1.78
t = 2.37
t = 2.96
t = 3.56
t = 4.15
t = 4.74
t = 5.33
t = 5.93
t = 6.52
t = 7.11
t = 7.37
sol ex
u
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Solution approchée finale et exacte
t = 7.37
sol ex
u
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2



On cherche l'erreur de convergence :

[6]:
```
def norm_l2 (x, y, ndim):
    norm = np.zeros(ndim)
    for k in range(1,n):
        norm[k] = (abs(x[k]-y[k]))**2
    return math.sqrt(np.sum(norm))

pas= np.zeros(NX)
erreur = np.zeros(NX)

for n in range(3,NX):
    h=L/(n-1)
    pas[n] = h
    x = np.linspace(0,1,n)
    erreur[n]= h*norm_l2(T, sol_ex(x), n)

plt.figure()

plt.plot(pas[3:],erreur[3:])
plt.title('Erreur de CV')
```

```
plt.xlabel(u'Pas $h$', fontsize=20)
plt.ylabel(u'$Erreur$', fontsize=20)
plt.grid('True')
```