Projet 4, Solution de l'équation de la chaleur par séparation de variavles

Zouiche Omar et David Czarnecki

28 Févrirer 2021

0.0.1 Solution de l'équation de la chaleur par séparation de variable

Nous allons utiliser et modifier le code chaleur1dspec.py et le fichier poly_insa_sepvar.pdf pour résoudre l'équation différentielle

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

par séparation de variables, dans le cas où $f = 0$ puis $f \neq 0$. Avec les conditions : - $ u(t,0) = u(t,L) = 0 $ - $ u(0,x) = u_0(x) $

0.0.2 $f = 0$

Avec $k = 1$ l'équation devient :

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

Qui est l'équation de la chaleur traitée auparavant.

0.0.3 $f \neq 0$

On pose :

$$u(t,x) = \psi(t)\varphi(x)$$

Ce qui donne en remplaçant u par le produit de fonctions posé :

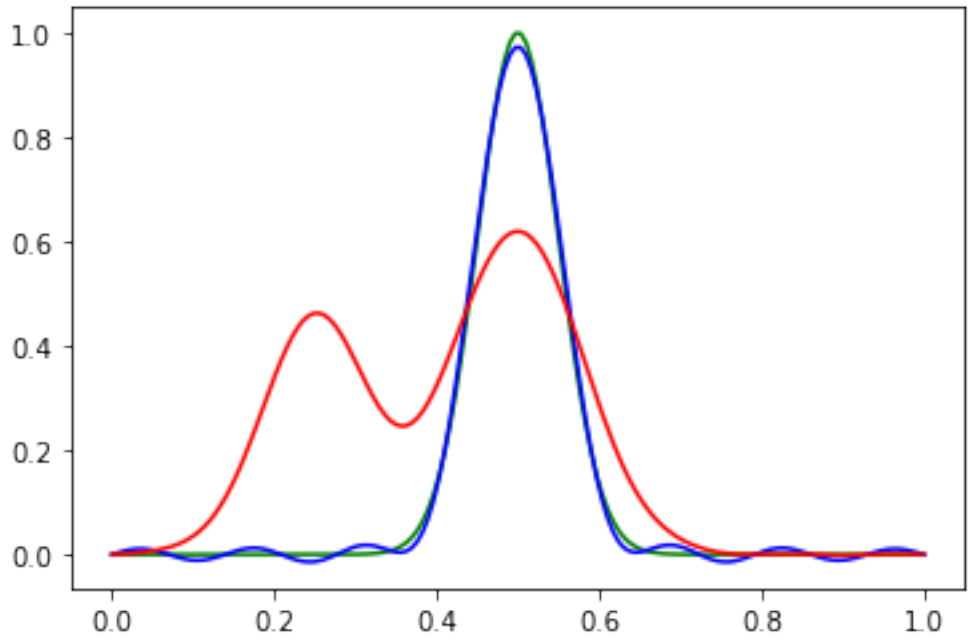
$$\psi'(t)\phi(x) - \psi(t)\phi''(x) = f(x)$$

Le reste de la méthode est détaillé dans le fichier poly_insa_sepvar.pdf. Voici les modifications aportées aux codes et leurs résultats :

[1]:
```
from math import sin,sqrt,exp,pi
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

#Initialisation
k = 0.1
s= 200
Lx = 1
```

```
Nx = 150 #le maillage spatial sert a la representation de la solution et aux␣
 ↪calculs des ps par integration numerique
hx = Lx/(Nx-1)
x = np.linspace(0,Lx,Nx)
modmax =15 #rang de la série


#introduire une boucle sur modmax et calculer l'erreur ||u(modmax)||
f=np.zeros(len(x))
u0=np.zeros(len(x))
u=np.zeros(len(x))
testx=np.zeros(modmax)
fbx=np.zeros((modmax,Nx))       #phi_n    #ligne : modmax, colonne : Nx
fmp=np.zeros(modmax)            #projection sur phi_n de f
imp=np.zeros(modmax)            #projection sur phi_n de u0
#rhs
#Coeur du programme

for i in range(1,Nx):
    f[i]=30*exp(-s*((x[i]-Lx/4)**2))  #(100*(x[i]**2)*((Lx-x[i])**2))/(Lx**4)
    u0[i]=exp(-s*((x[i]-Lx/2)**2)) #+exp(-2*s*((x[i]-Lx/
 ↪3)**2))+exp(-3*s*((x[i]-2*Lx/3)**2))
    u[i]=u0[i]

plt.plot(x,u0,'g')
#fb normalise
#
for m in range(1,modmax):
    for i in range(1,Nx):
        fbx[m][i]=sin(pi*x[i]/Lx*m)                 #phi_n     #ligne puis colonne
        testx[m]=testx[m]+fbx[m][i]*fbx[m][i]*hx
    testx[m]=sqrt(testx[m])                         #norme L2 de phi_n
    for i in range(1,Nx):                           #normalisation
        fbx[m][i]=fbx[m][i]/testx[m]


#verifier l'orthonormalite des fbx ?
#<fbx[m],fbx[n]>=delta_mn


#projection f second membre et u0 condi init sur fbx

for m in range(1,modmax):
    for i in range(1,Nx):
        fmp[m]+=f[i]*fbx[m][i]*hx    # <f,\phi_n> = f_n
        imp[m]+=u0[i]*fbx[m][i]*hx   # <u0,phi_n> = c_n
```

```
#somme serie

temps=0.0    #on doit retrouver la condition initiale
for i in range(1,Nx):
    u[i]=0
for m in range(1,modmax):
    al=(m**2)*(pi**2)/(Lx**2)*k
    coef=imp[m]*exp(-al*temps)
    for i in range(0,Nx-1):
         u[i]+=fbx[m][i]*coef

plt.plot(x,u,'blue')

temps=0.02 #la solution a n'importe quel temps sans avoir a calculer les iter␣
 ↪intermediaires
for i in range(1,Nx):
    u[i]=0
for m in range(1,modmax):
    al=(m**2)*(pi**2)/(Lx**2)*k
    coef=imp[m]*exp(-al*temps)
    coeff=fmp[m]*(1-exp(-al*temps))/al
    for i in range(0,Nx):
        u[i]+=fbx[m][i]*(coeff+coef)

plt.plot(x,u,'r')
plt.show()
```

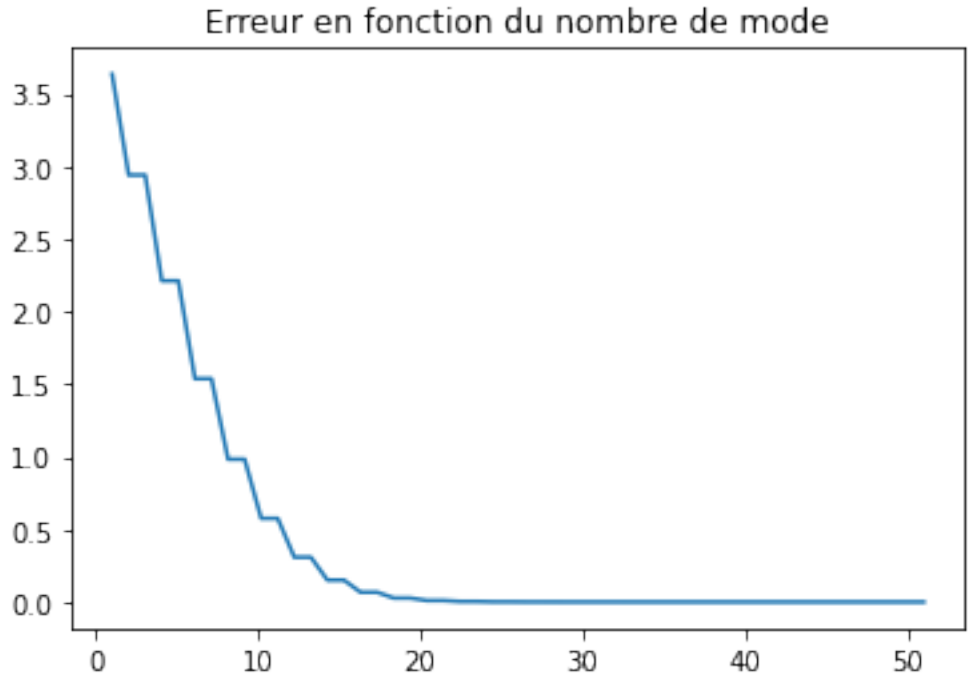


0.0.4 Erreur absolue L_2

[6]:
```
k = 0.1
s= 200
Lx = 1
Nx = 150 #le maillage spatial sert a la representation de la solution et aux␣
 ↪calculs des ps par integration numerique
hx = Lx/(Nx-1)
x = np.linspace(0,Lx,Nx)
err=np.zeros(50)
modmax1=np.linspace(1,51)


#introduire une boucle sur modmax et calculer l'erreur ||u(modmax)||
for modmax in range(1,51):
    f=np.zeros(len(x))
    u0=np.zeros(len(x))
    u=np.zeros(len(x))
    testx=np.zeros(modmax)
    fbx=np.zeros((modmax,Nx))       #phi_n
    fmp=np.zeros(modmax)            #projection sur phi_n de f
    imp=np.zeros(modmax)            #projection sur phi_n de u0
    #rhs
    #Coeur du programme

    for i in range(1,Nx):
        f[i]=30*exp(-s*((x[i]-Lx/4)**2))  #(100*(x[i]**2)*((Lx-x[i])**2))/(Lx**4)
        u0[i]=exp(-s*((x[i]-Lx/2)**2)) #+exp(-2*s*((x[i]-Lx/
 ↪3)**2))+exp(-3*s*((x[i]-2*Lx/3)**2))
        u[i]=u0[i]

    #plt.plot(x,u0,'g')
    #fb normalise
    #
    for m in range(1,modmax):
        for i in range(1,Nx):
            fbx[m][i]=sin(pi*x[i]/Lx*m)                 #phi_n
            testx[m]=testx[m]+fbx[m][i]*fbx[m][i]*hx
        testx[m]=sqrt(testx[m])                         #norme L2 de phi_n
        for i in range(1,Nx):                           #normalisation
            fbx[m][i]=fbx[m][i]/testx[m]


    #verifier l'orthonormalite des fbx ?
    #<fbx[m],fbx[n]>=delta_mn
```

```
    #projection f second membre et u0 condi init sur fbx

    for m in range(1,modmax):
        for i in range(1,Nx):
            fmp[m]+=f[i]*fbx[m][i]*hx     # <f,\phi_n> = f_n
            imp[m]+=u0[i]*fbx[m][i]*hx    # <u0,phi_n> = c_n

    #somme serie

    temps=0.0    #on doit retrouver la condition initiale
    for i in range(1,Nx):
        u[i]=0
    for m in range(1,modmax):
        al=(m**2)*(pi**2)/(Lx**2)*k
        coef=imp[m]*exp(-al*temps)
        for i in range(0,Nx-1):
            u[i]+=fbx[m][i]*coef

    err[modmax-1]=np.linalg.norm(u-u0)

plt.plot(modmax1,err)
plt.title('Erreur en fonction du nombre de mode')
```

[6]: `Text(0.5, 1.0, 'Erreur en fonction du nombre de mode')`



0.0.5 Optimisation de modmax et de NX

En considérant $f(x) = 30e^{-200\frac{x}{4}}$ On veut : - Trouver combien de mode faut-il considérer si on veut une erreur absolue telle que : $\frac{||u(Nb_{modes},N_x)-u_{ex}||_{L2})}{||u_{ex}||} < 1.e^{-2}$ - Trouver le maillage spatial le plus grossier pour un calcul plus rapide indépendamment de la qualité des résulats.

[12]:
```
#Initialisation
k = 0.1
s= 200
Lx = 1
Nx = 200 #le maillage spatial sert a la representation de la solution et aux␣
 ↪calculs des ps par integration numerique
hx = Lx/(Nx-1)
x = np.linspace(0,Lx,Nx)
err=27
modmax=1


#introduire une boucle sur modmax et calculer l'erreur ||u(modmax)||
while err>=1.e-2 :
    modmax=modmax+1
    f=np.zeros(len(x))
    u0=np.zeros(len(x))
    u=np.zeros(len(x))
    testx=np.zeros(modmax)
    fbx=np.zeros((modmax,Nx))       #phi_n
    fmp=np.zeros(modmax)            #projection sur phi_n de f
    imp=np.zeros(modmax)            #projection sur phi_n de u0
    #rhs
    #Coeur du programme

    for i in range(1,Nx):
        f[i]=30*exp(-s*((x[i]-Lx/4)**2))  #(100*(x[i]**2)*((Lx-x[i])**2))/(Lx**4)
        u0[i]=exp(-s*((x[i]-Lx/2)**2))#+exp(-2*s*((x[i]-Lx/
 ↪3)**2))+exp(-3*s*((x[i]-2*Lx/3)**2))
        u[i]=u0[i]

    #plt.plot(x,u0,'g')
    #fb normalise
    #
    for m in range(1,modmax):
        for i in range(1,Nx):
            fbx[m][i]=sin(pi*x[i]/Lx*m)               #phi_n
            testx[m]=testx[m]+fbx[m][i]*fbx[m][i]*hx
```

```
        testx[m]=sqrt(testx[m])                          #norme L2 de phi_n
        for i in range(1,Nx):                            #normalisation
            fbx[m][i]=fbx[m][i]/testx[m]


    #verifier l'orthonormalite des fbx ?
    #<fbx[m],fbx[n]>=delta_mn


    #projection f second membre et u0 condi init sur fbx

    for m in range(1,modmax):
        for i in range(1,Nx):
            fmp[m]+=f[i]*fbx[m][i]*hx     # <f,\phi_n> = f_n
            imp[m]+=u0[i]*fbx[m][i]*hx    # <u0,phi_n> = c_n

    #somme serie

    temps=0.0    #on doit retrouver la condition initiale
    for i in range(1,Nx):
        u[i]=0
    for m in range(1,modmax):
        al=(m**2)*(pi**2)/(Lx**2)*k
        coef=imp[m]*exp(-al*temps)
        for i in range(0,Nx-1):
            u[i]+=fbx[m][i]*coef

    err=(np.linalg.norm(u-u0))/(np.linalg.norm(u0))

print('modmax = ',modmax)
```

```
modmax =  18
```

[10]:
```
import math
import numpy as np


#Initialisation
k = 0.1
s= 200
modmax = 1
Lx=1
Nx=5

while modmax!=18 :
    modmax = 1
```

```
   Nx = Nx+1  #le maillage spatial sert a la representation de la solution et␣
↪aux calculs des ps par integration numerique
   hx = Lx/(Nx-1)
   x = np.linspace(0,Lx,Nx)

   #introduire une boucle sur modmax et calculer l'erreur ||u(modmax)||
   f=np.zeros(len(x))
   u0=np.zeros(len(x))
   u=np.zeros(len(x))
   err=27

   while (err >= 1e-2) :
       modmax = modmax+1
       testx=np.zeros(modmax)
       fbx=np.zeros((modmax,Nx))       #phi_n
       fmp=np.zeros(modmax)            #projection sur phi_n de f
       imp=np.zeros(modmax)            #projection sur phi_n de u0
       #rhs
       #Coeur du programme

       for i in range(1,Nx):
           f[i]= 30*exp(-s*((x[i]-Lx/4)**2))  #(100*(x[i]**2)*((Lx-x[i])**2))/
↪(Lx**4)
           u0[i]=exp(-s*((x[i]-Lx/2)**2)) #exp(-2*s*((x[i]-Lx/
↪3)**2))+exp(-3*s*((x[i]-2*Lx/3)**2))
           u[i]=u0[i]

       #plt.plot(x,u0,'g')
       #fb normalise
       #
       for m in range(1,modmax):
           for i in range(1,Nx):
               fbx[m][i]=sin(pi*x[i]/Lx*m)                #phi_n
               testx[m]=testx[m]+fbx[m][i]*fbx[m][i]*hx
           testx[m]=sqrt(testx[m])                        #norme L2 de phi_n
           for i in range(1,Nx):                          #normalisation
               fbx[m][i]=fbx[m][i]/testx[m]


       #verifier l'orthonormalite des fbx ?
       #<fbx[m],fbx[n]>=delta_mn


       #projection f second membre et u0 condi init sur fbx

       for m in range(1,modmax):
           for i in range(1,Nx):
```

```
                fmp[m]+=f[i]*fbx[m][i]*hx     # <f,\phi_n> = f_n
                imp[m]+=u0[i]*fbx[m][i]*hx    # <u0,phi_n> = c_n

        #somme serie

        temps=0.0    #on doit retrouver la condition initiale
        for i in range(1,Nx):
            u[i]=0
        for m in range(1,modmax):
            al=(m**2)*(pi**2)/(Lx**2)*k
            coef=imp[m]*exp(-al*temps)
            for i in range(0,Nx-1):
                 u[i]+=fbx[m][i]*coef

        #plt.plot(x,u,'blue')



        err=np.linalg.norm(u-u0)/np.linalg.norm(u0)
        #plt.plot(x,u,'r')
        #plt.show()

print("Nx = ", Nx)
```

```
Nx =  19
```