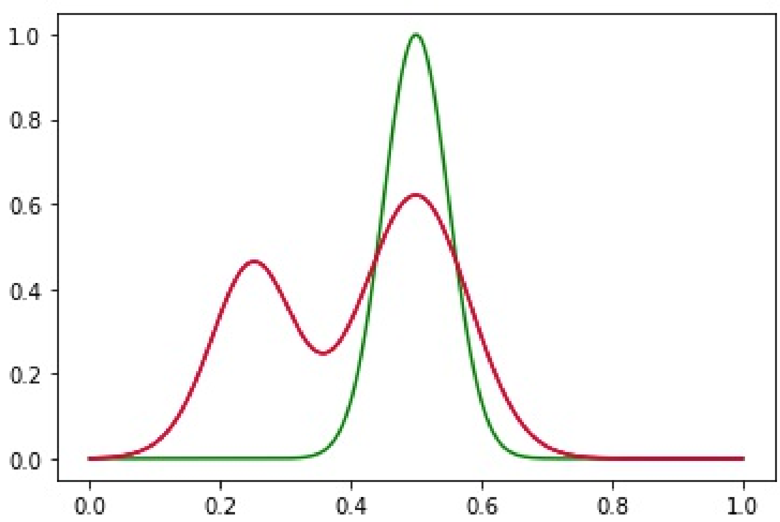
Sur l'équation de chaleur et la méthode de séparation des variables

Gaggini Lorenzo et Ferretti Thibault

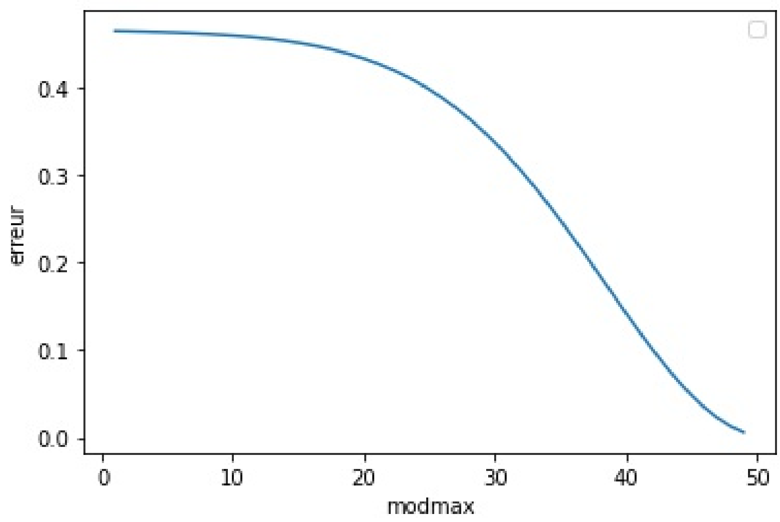
Dans ce rapport, on s'intéresse à la méthode de séparation de variables dans le cadre de la résolution d'équations aux dérivées partielles à travers l'exemple de l'équation de chaleur.

Dans le cadre de l'équation de chaleur, cette méthode nous amène à considérer le code chaleur1spec.py qui, une fois adapté, nous permet d'évaluer une solution approchée de l'EDP en fonction des conditions initiale et du nombre de mode considéré, pour une précision de maillage donnée.

Ci-aprés les différentes figures obtenue, et le code associé.



En vert la condition initiales, en rouge la solution approché



Algorithme `chaleur1spec.py` adapté

```
from math import sin,sqrt,exp,pi
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from numpy import linalg as LA
#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)
uu = []
uuu = []
err = []
Merr = []
test=50

#introduire une boucle sur modmax et calculer l'erreur ||u(modmax)||
for modmax in range(1,test):
    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]

    #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


    temps=0.02  #la solution a n'importe quel temps sans avoir a calculer les iter intermediaires
    m=1
    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)

for i in range(0,modmax):
    Merr.append(LA.norm(u[test]-u[i]))
    if abs(u[test]-u[i])<(10**(-1)*u[test]):
        print('il faut ', modmax, 'modes')
```