Faculté des sciences de Montpellier
Rapport séance 3

Étudiant: Giscard Leonel Zouakeu

Problème

On commence par considérer un domaine Ωinfini dans la direction $s = s_2$ et de la Longueur L dans la direction $s = s_1$.
On considère le modèle dans ce domaine:

$$\begin{aligned} \mathbf{u}_t + V(t,s)\mathbf{u}_s - \nu \mathbf{u}_{ss} &= -\lambda \mathbf{u} + \mathbf{f}(t,s) \quad s \in \Omega =]0, L[\quad t \geq 0 \\ \mathbf{u}(t,0) &= \mathbf{u}_t \\ \mathbf{u}_s(t,L) &= 0 \\ \mathbf{u}(0,s) &= \mathbf{u}_0(s) \end{aligned}$$

Par la séance 2 nous avons:

$$\mathbf{u}_{\text{ex}} = \exp(-10(s - \frac{L}{2})^2)$$

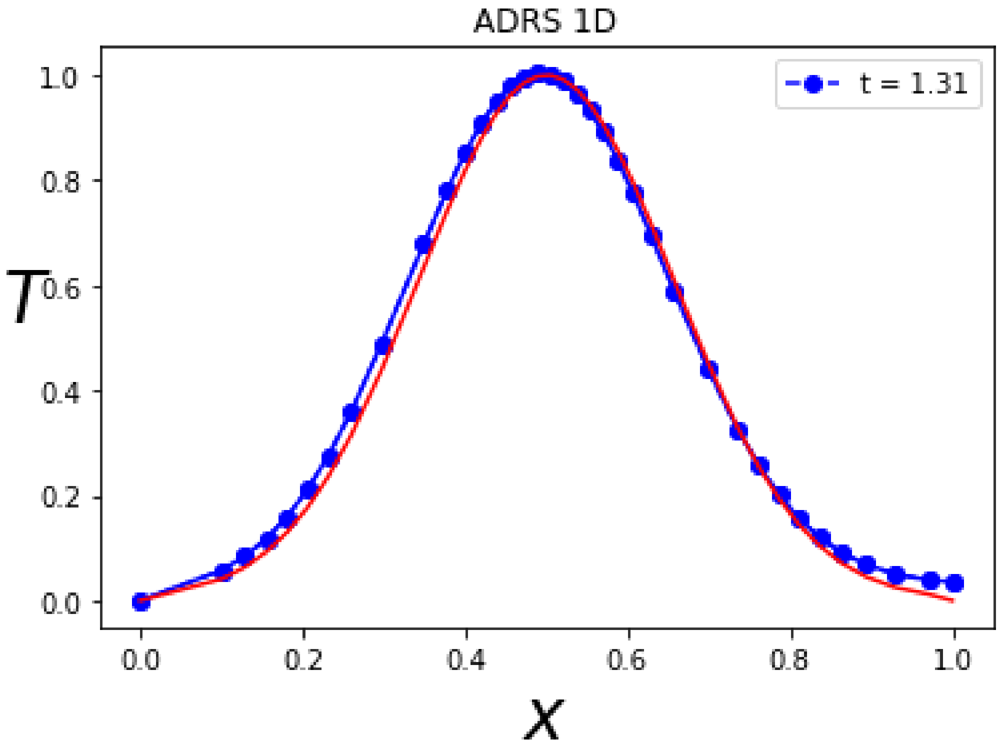
compréhension et modification du code multimeshadrs.py

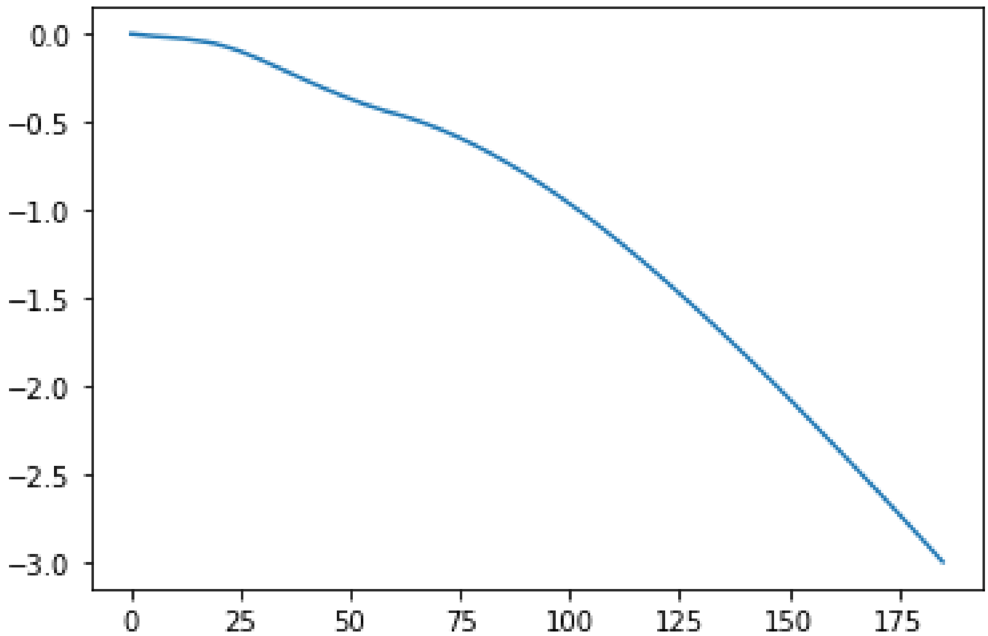
```
import math
import numpy as np
import matplotlib.pyplot as plt

#u,t = -V u,x + k u,xx  -lamda u + f

iplot=1

# PHYSICAL PARAMETERS
K = 0.01     #Diffusion coefficient
xmin = 0.0
xmax = 1.0
Time = 10.  #Integration time

V=1.
lamda=1

#mesh adaptation param

niter_refinement=30      #niter different calculations
hmin=0.01
hmax=0.1
err=0.01

# NUMERICAL PARAMETERS
NX = 4    #Number of grid points : initialization
NT = 10000   #Number of time steps max
ifre=1000000  #plot every ifre time iterations
eps=0.001     #relative convergence ratio

errorL2=np.zeros((niter_refinement))
```

```python
errorH1=np.zeros((niter_refinement))
itertab=np.zeros((niter_refinement))
hloc = np.ones((NX))*hmax

iter=0
NX0=0
while( np.abs(NX0-NX) > 2 and iter<niter_refinement):

  iter+=1
  itertab[iter]=1./NX

  iplot=iter-2

  x = np.linspace(xmin,xmax,NX)
  T = np.zeros((NX))

#mesh adaptation using local metric
  if(iter>0):
    xnew=[]
    Tnew=[]
    nnew=1
    xnew.append(xmin)
    Tnew.append(T[0])
    while(xnew[nnew-1] < xmax-hmin):
      for i in range(0,NX-1):
        if(xnew[nnew-1] >= x[i] and xnew[nnew-1] <= x[i+1] and xnew[nnew-1]<xmax-hmin):
          hll=(hloc[i]*(x[i+1]-xnew[nnew-1])+hloc[i+1]*(xnew[nnew-1]-x[i]))/(x[i+1]-x[i])
          hll=min(max(hmin,hll),hmax)
          nnew+=1
#         print(nnew,hll,min(xmax,xnew[nnew-2]+hll))
          xnew.append(min(xmax,xnew[nnew-2]+hll))
#solution interpolation for initialization (attention initial solution on first mesh in the row)
          un=(T[i]*(x[i+1]-xnew[nnew-1])+T[i+1]*(xnew[nnew-1]-x[i]))/(x[i+1]-x[i])
          Tnew.append(un)

    NX0=NX
    NX=nnew
    x = np.linspace(xmin,xmax,NX)
    x[0:NX]=xnew[0:NX]
    print(x)
    T = np.zeros((NX))
    T[0:NX]=Tnew[0:NX]
#   T[NX-1]=0

  rest = []
  F = np.zeros((NX))
  RHS = np.zeros((NX))
  hloc = np.ones((NX))*hmax*0.5
  metric = np.ones((NX))

```

```
Tex = np.zeros((NX))
for j in range (1,NX-1):
  Tex[j] = np.exp(-20*(x[j]-(xmax+xmin)*0.5)**2)

dt=1.e30
for j in range (1,NX-1):
  Tx=(Tex[j+1]-Tex[j-1])/(x[j+1]-x[j-1])
  Txip1=(Tex[j+1]-Tex[j])/(x[j+1]-x[j])
  Txim1=(Tex[j]-Tex[j-1])/(x[j]-x[j-1])
  Txx=(Txip1-Txim1)/(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))
  F[j]=V*Tx-K*Txx+lamda*Tex[j]
  dt=min(dt,0.5*(x[j+1]-x[j-1])**2/(V*np.abs(x[j+1]-x[j-1])+4*K+np.abs(F[j])*(x[j+1]-x[j-1])**2))

print('NX=',NX,'Dt=',dt)

if(iplot==1):
  plt.figure(1)

#time step loop
n=0
res=1
res0=1
t=0
while(n<NT and res/res0>eps and t<Time):
  n+=1
  t+=dt
#discretization of the advection/diffusion/reaction/source equation
  res=0
  for j in range (1, NX-1):
    visnum=0.5*(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))*np.abs(V)
    xnu=K+visnum
    Tx=(T[j+1]-T[j-1])/(x[j+1]-x[j-1])
    Txip1=(T[j+1]-T[j])/(x[j+1]-x[j])
    Txim1=(T[j]-T[j-1])/(x[j]-x[j-1])
    Txx=(Txip1-Txim1)/(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))

    RHS[j] = dt*(-V*Tx+xnu*Txx-lamda*T[j]+F[j])
    metric[j]=min(1./hmin**2,max(1./hmax**2,abs(Txx)/err))
    res+=abs(RHS[j])

  metric[0]=metric[1]
  metric[NX-1]=metric[NX-2]

  for j in range (0, NX-1):
    metric[j]=0.5*(metric[j]+metric[j+1])
  metric[NX-1]=metric[NX-2]

  hloc[0:NX]=np.sqrt(1./metric[0:NX])

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

```
      RHS[j]=0

    T[NX-1]=1.2*T[NX-2]-0.2*T[NX-3]

    if (n == 1 ):
      res0=res

    rest.append(res)
  #Plot every ifre time steps
    if (n%ifre == 0 or (res/res0)<eps):
      print('iter=',n,'residual=',res)
      if(iplot==1):
        plotlabel = "t = %1.2f" %(n * dt)
        plt.plot(x[0:NX],T[0:NX], label=plotlabel,linestyle='--', marker='o', color='b')


  print('iter=',n,'time=',t,'residual=',res)
  if(iplot==1):
    plt.plot(x[0:NX],T[0:NX],marker='o', color='b')
    plt.plot(x[0:NX],Tex[0:NX],color='r')
    plt.xlabel(u'$x$', fontsize=26)
    plt.ylabel(u'$T$', fontsize=26, rotation=0)
    plt.title(u'ADRS 1D')
    plt.legend()
    plt.figure(2)
    plt.plot(np.log10(rest/rest[0]))


#  errL2=np.sqrt(np.dot(T-Tex,T-Tex))
  errH1h=0
  errL2h=0
  for j in range (1, NX-1):
    Texx=(Tex[j+1]-Tex[j-1])/(x[j+1]-x[j-1])
    Tx=(T[j+1]-T[j-1])/(x[j+1]-x[j-1])
    errL2h+=(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))*(T[j]-Tex[j])**2
    errH1h+=(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))*(Tx-Texx)**2




  print('norm error L2, H1=',errL2h,errH1h)

if(iplot==-1):
  plt.figure(3)
  plt.plot(itertab,np.log10(errorL2))
  plt.plot(itertab,np.log10(errorH1))

plt.show()
```


ADRS 1D
t = 1.31
T
X
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
0
25
50
75
100
125
150
175

Contrôle local de métrique(cf POLY.pdf Bijan Mohammadi)

Dans cette approche, on modifié le choix du produit scalaire euclidien qui sert à définir les distances dans le maillage pour pouvoirs générer les éléments dans une nouvelle métrique suivant certains critères. On cherchera la métrique qui équirépartira l'erreur d'interpolation. Présentons cette technique dans le cadre de la dimension 1. On dispose de la majoration suivante pour l'erreur d'interpolation en élément P1:

$$||u - \prod_h u||_0 \leq ch^2|u"| \tag{1}$$

Ou $\prod_h u$ est l'interpolation linéaire de u, h la taille des éléments. LA métrique local en chaque noeuds x du maillage est définie par:

$$\lambda_i = \min(\max(\frac{1}{\epsilon}|u"|, \frac{1}{h_{\min}^2}), \frac{1}{h_{\max}^2}) \tag{2}$$

Ou $\epsilon, h_{\min}, h_{\max}$ sont respectivement l'erreur et les longueurs minimale et maximale désirées des mailles.

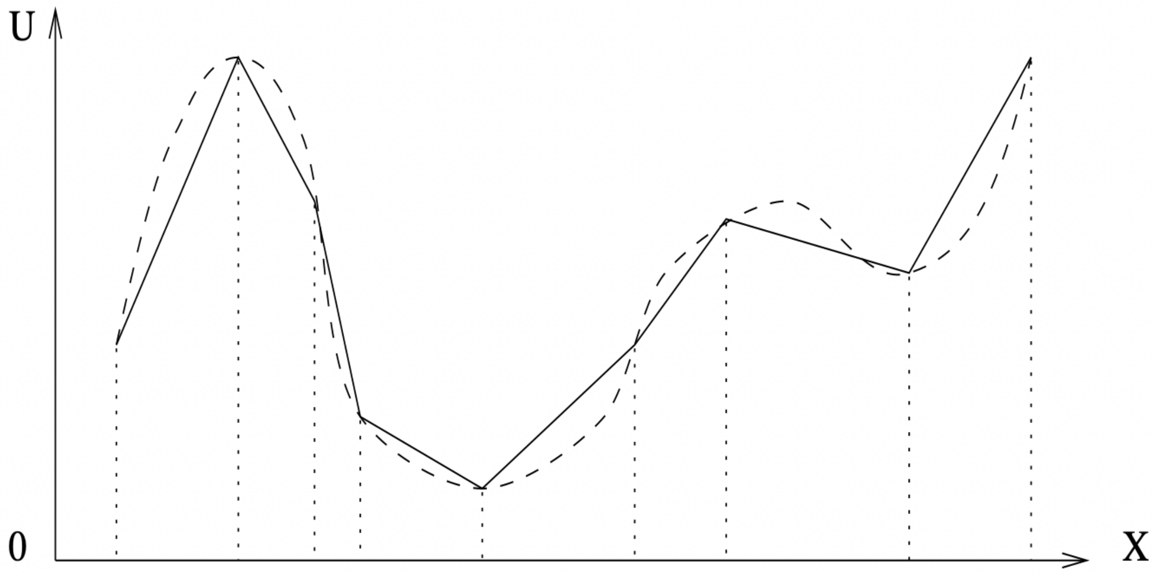


FIGURE 18.1 – Approximation $P1$ Lagrange de u.

Pour plus d'explication , référence poly.pdf de Monsieur Bijan Mohammadi

Identification de la définition de la métrique dans le code

Dans notre code ci-dessus, nous identifions la définition de la métrique à la ligne 118 .

```
metric[j]=min(1./hmin**2,max(1./hmax**2,abs(Txx)/err))
```

Implémentons la loi (3)

La longueur du segment (s_i, s_{i+1}) dans la metrique λ est définie par :

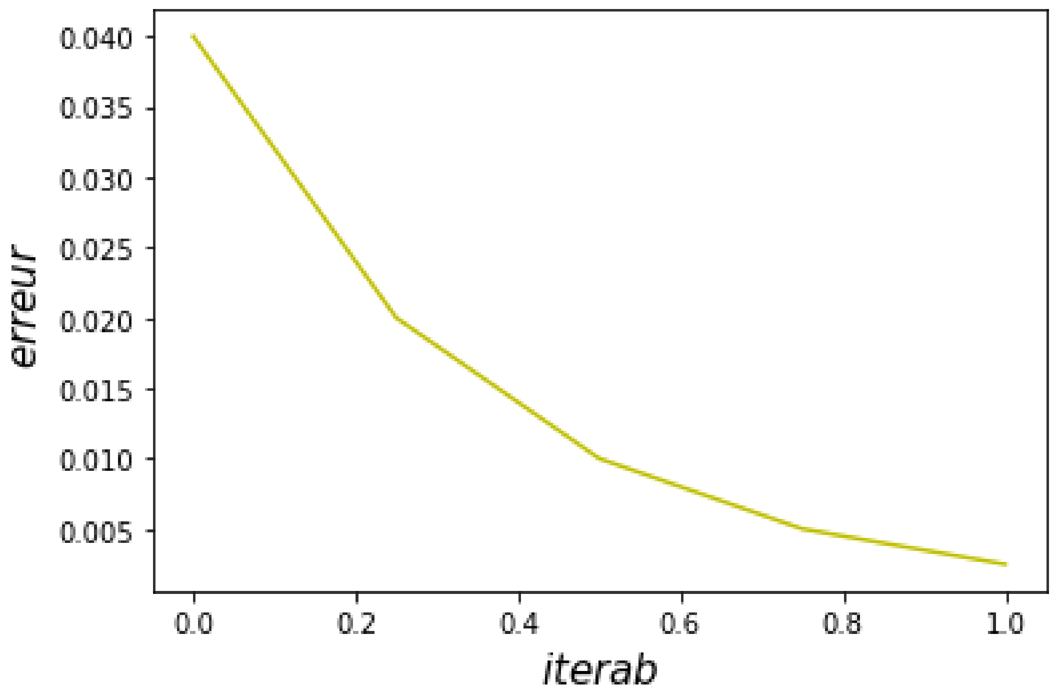
$$l(s_i, s_{i+1}) = (s_i, s_{i+1})^2 \frac{\lambda_{i+1} + \lambda_i}{2} = 1 \tag{3}$$

```
    for j in range (0, NX-1):
      metric[j]=0.5*(metric[j]+metric[j+1])*(x[j+1]-x[j])**2
    metric[NX-1]=metric[NX-2]

    hloc[0:NX]=np.sqrt(1./metric[0:NX])
```

Traçons l'erreur pour err= 0.04, 0.02, 0.01, 0.005, 0.0025

```
import math
import numpy as np
import matplotlib.pyplot as plt

#u,t = -V u,x + k u,xx  -lamda u + f

iplot=1

# PHYSICAL PARAMETERS
K = 0.01     #Diffusion coefficient
xmin = 0.0
xmax = 1.0
Time = 10.  #Integration time

V=1.
lamda=1

#mesh adaptation param

niter_refinement=5      #niter different calculations
hmin=0.01
hmax=0.1
err=0.01

# NUMERICAL PARAMETERS
NX = 4    #Number of grid points : initialization
NT = 10000   #Number of time steps max
ifre=1000000  #plot every ifre time iterations
eps=0.001     #relative convergence ratio

errorL2=np.zeros((niter_refinement))
errorH1=np.zeros((niter_refinement))
itertab=np.zeros((niter_refinement))
hloc = np.ones((NX))*hmax

iter=0
NX0=0

for i in range(1,5):
  itertab[i]=i/NX
print(itertab)
erreur=[0.04, 0.02, 0.01, 0.005, 0.0025]
```

```


plt.xlabel(u'$iterab$', fontsize=15)
plt.ylabel(u'$erreur$', fontsize=15)
plt.plot(itertab,erreur,'y')
```



La courbe de l'erreur décroît si Nx (itertab) augmente.