



Faculté des sciences de Montpellier
Rapport séance 3

Étudiant: Giscard Leonel Zouakeu

Problème

On commence par considérer un domaine Ω infini 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:

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

compréhension et modification du code multimeshadr.py

```
1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 #u,t = -V u,x + k u,xx -lamda u + f
6
7 iplot=1
8
9 # PHYSICAL PARAMETERS
10 K = 0.01 #Diffusion coefficient
11 xmin = 0.0
12 xmax = 1.0
13 Time = 10. #Integration time
14
15 V=1.
16 lamda=1
17
18 #mesh adaptation param
19
20 niter_refinement=30 #niter different calculations
21 hmin=0.01
22 hmax=0.1
23 err=0.01
24
25 # NUMERICAL PARAMETERS
26 NX = 4 #Number of grid points : initialization
27 NT = 10000 #Number of time steps max
28 ifre=1000000 #plot every ifre time iterations
29 eps=0.001 #relative convergence ratio
30
31 errorL2=np.zeros((niter_refinement))
```

```

32 errorH1=np.zeros((niter_refinement))
33 itertab=np.zeros((niter_refinement))
34 hloc = np.ones((NX))*hmax
35
36 iter=0
37 NX0=0
38 while( np.abs(NX0-NX) > 2 and iter<niter_refinement):
39
40     iter+=1
41     itertab[iter]=1./NX
42
43     iplot=iter-2
44
45     x = np.linspace(xmin,xmax,NX)
46     T = np.zeros((NX))
47
48     #mesh adaptation using local metric
49     if(iter>0):
50         xnew=[]
51         Tnew=[]
52         nnew=1
53         xnew.append(xmin)
54         Tnew.append(T[0])
55         while(xnew[nnew-1] < xmax-hmin):
56             for i in range(0,NX-1):
57                 if(xnew[nnew-1] >= x[i] and xnew[nnew-1] <= x[i+1] and
58                    xnew[nnew-1]<xmax-hmin):
59                     h11=(hloc[i]*(x[i+1]-xnew[nnew-1])+hloc[i+1]*(xnew[
60                        nnew-1]-x[i]))/(x[i+1]-x[i])
61                     h11=min(max(hmin,h11),hmax)
62                     nnew+=1
63                     # print(nnew,h11,min(xmax,xnew[nnew-2]+h11))
64                     xnew.append(min(xmax,xnew[nnew-2]+h11))
65 #solution interpolation for initialization (attention initial
66    solution on first mesh in the row)
67    un=(T[i]*(x[i+1]-xnew[nnew-1])+T[i+1]*(xnew[nnew-1]-x
68    [i]))/(x[i+1]-x[i])
69    Tnew.append(un)
70
71    NX0=NX
72    NX=nnew
73    x = np.linspace(xmin,xmax,NX)
74    x[0:NX]=xnew[0:NX]
75    print(x)
76    T = np.zeros((NX))
77    T[0:NX]=Tnew[0:NX]
78    # T[NX-1]=0
79
80    rest = []
81    F = np.zeros((NX))
82    RHS = np.zeros((NX))
83    hloc = np.ones((NX))*hmax*0.5
84    metric = np.ones((NX))

```

```

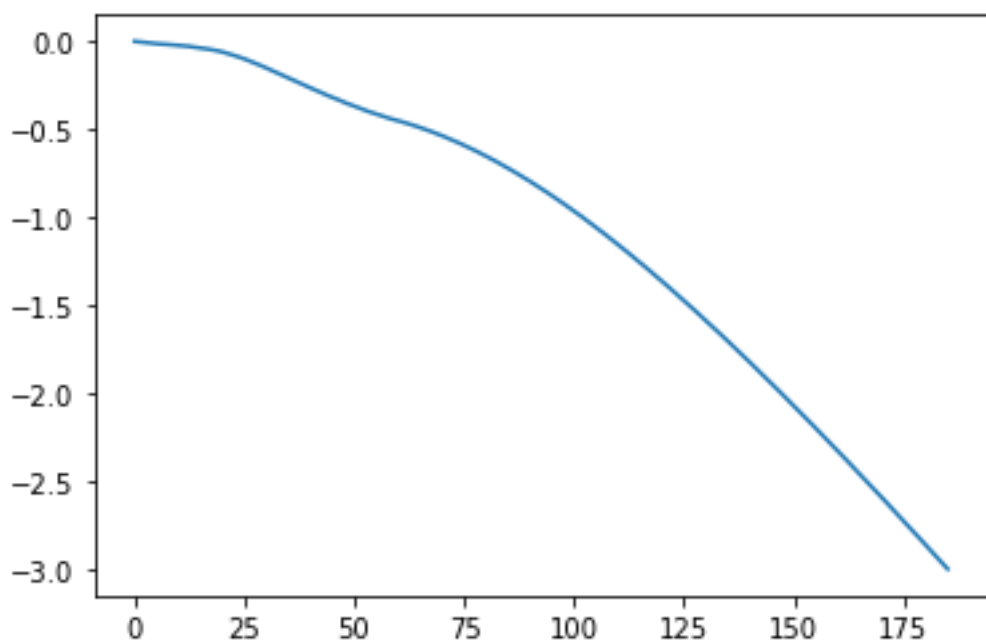
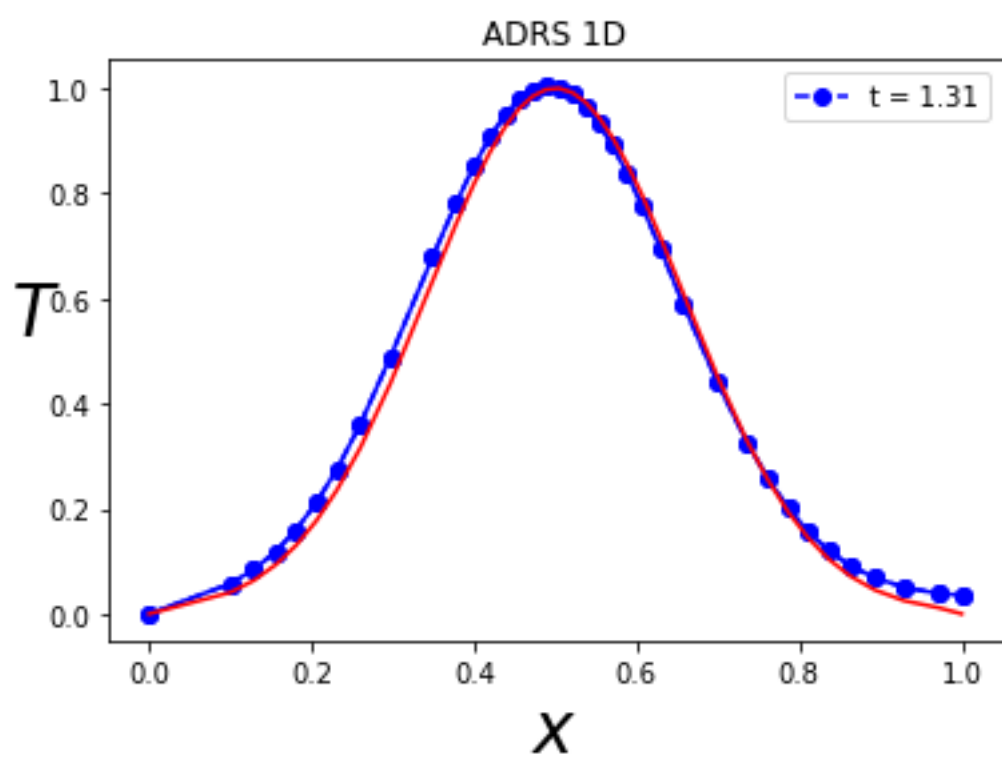
82 Tex = np.zeros((NX))
83 for j in range (1,NX-1):
84     Tex[j] = np.exp(-20*(x[j]-(xmax+xmin)*0.5)**2)
85
86 dt=1.e30
87 for j in range (1,NX-1):
88     Tx=(Tex[j+1]-Tex[j-1])/(x[j+1]-x[j-1])
89     Txip1=(Tex[j+1]-Tex[j])/(x[j+1]-x[j])
90     Txim1=(Tex[j]-Tex[j-1])/(x[j]-x[j-1])
91     Txx=(Txip1-Txim1)/(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))
92     F[j]=V*Tx-K*Txx+lamda*Tex[j]
93     dt=min(dt,0.5*(x[j+1]-x[j-1])**2/(V*np.abs(x[j+1]-x[j-1])
94         +4*K*np.abs(F[j])*(x[j+1]-x[j-1])**2))
95
96 print('NX=',NX,'Dt=',dt)
97
98 if(iplot==1):
99     plt.figure(1)
100
101 #time step loop
102 n=0
103 res=1
104 res0=1
105 t=0
106 while(n<NT and res/res0>eps and t<Time):
107     n+=1
108     t+=dt
109 #discretization of the advection/diffusion/reaction/source
110 #equation
111 res=0
112 for j in range (1, NX-1):
113     visnum=0.5*(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))*np.abs(V
114 )
115     xnu=K+visnum
116     Tx=(T[j+1]-T[j-1])/(x[j+1]-x[j-1])
117     Txip1=(T[j+1]-T[j])/(x[j+1]-x[j])
118     Txim1=(T[j]-T[j-1])/(x[j]-x[j-1])
119     Txx=(Txip1-Txim1)/(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))
120
121     RHS[j] = dt*(-V*Tx+xnu*Txx-lamda*T[j]+F[j])
122     metric[j]=min(1./hmin**2,max(1./hmax**2,abs(Txx)/err))
123     res+=abs(RHS[j])
124
125 metric[0]=metric[1]
126 metric[NX-1]=metric[NX-2]
127
128 for j in range (0, NX-1):
129     metric[j]=0.5*(metric[j]+metric[j+1])
130 metric[NX-1]=metric[NX-2]
131
132 hloc[0:NX]=np.sqrt(1./metric[0:NX])
133
134 for j in range (1, NX-1):
135     T[j] += RHS[j]

```

```

132     RHS[j]=0
133
134     T[NX-1]=1.2*T[NX-2]-0.2*T[NX-3]
135
136     if (n == 1 ):
137         res0=res
138
139     rest.append(res)
140     #Plot every ifre time steps
141     if (n%ifre == 0 or (res/res0)<eps):
142         print('iter=',n,'residual=',res)
143         if (iplot==1):
144             plotlabel = "t = %1.2f" %(n * dt)
145             plt.plot(x[0:NX],T[0:NX], label=plotlabel,linestyle='--',
146                     'marker='o', color='b')
147
148     print('iter=',n,'time=',t,'residual=',res)
149     if (iplot==1):
150         plt.plot(x[0:NX],T[0:NX],marker='o', color='b')
151         plt.plot(x[0:NX],Tex[0:NX],color='r')
152         plt.xlabel(u'$x$', fontsize=26)
153         plt.ylabel(u'$T$', fontsize=26, rotation=0)
154         plt.title(u'ADRS 1D')
155         plt.legend()
156         plt.figure(2)
157         plt.plot(np.log10(rest/rest[0]))
158
159
160     # errL2=np.sqrt(np.dot(T-TeX,T-TeX))
161     errH1h=0
162     errL2h=0
163     for j in range (1, NX-1):
164         Texx=(Tex[j+1]-Tex[j-1])/(x[j+1]-x[j-1])
165         Tx=(T[j+1]-T[j-1])/(x[j+1]-x[j-1])
166         errL2h+=(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))*(T[j]-Tex[j])
167         **2
168         errH1h+=(0.5*(x[j+1]+x[j])-0.5*(x[j]+x[j-1]))*(Tx-Texx)**2
169
170
171
172     print('norm error L2, H1=',errL2h,errH1h)
173
174     if (iplot==1):
175         plt.figure(3)
176         plt.plot(itertab,np.log10(errorL2))
177         plt.plot(itertab,np.log10(errorH1))
178
179     plt.show()

```



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 pouvoir 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:

$$||u - \prod_h u||_0 \leq ch^2 |u''| \quad (1)$$

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:

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

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

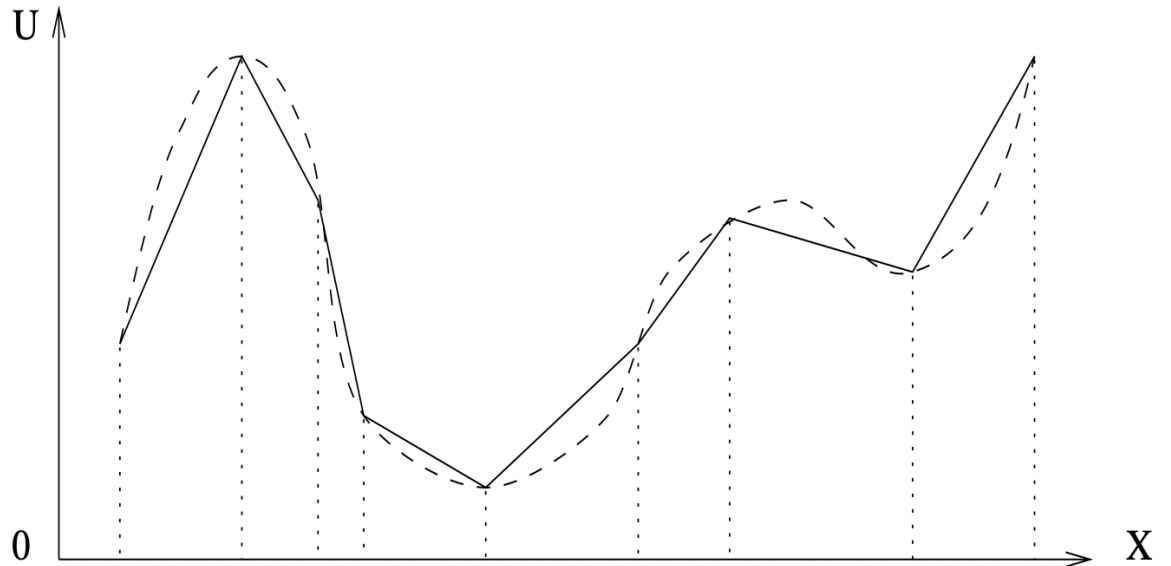


FIGURE 18.1 – Approximation $P1$ Lagrange de u .

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 .

```
1 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 :

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


```

1     for j in range (0, NX-1):
2         metric[j]=0.5*(metric[j]+metric[j+1])*(x[j+1]-x[j])**2
3         metric[NX-1]=metric[NX-2]
4
5     hloc[0:NX]=np.sqrt(1./metric[0:NX])

```

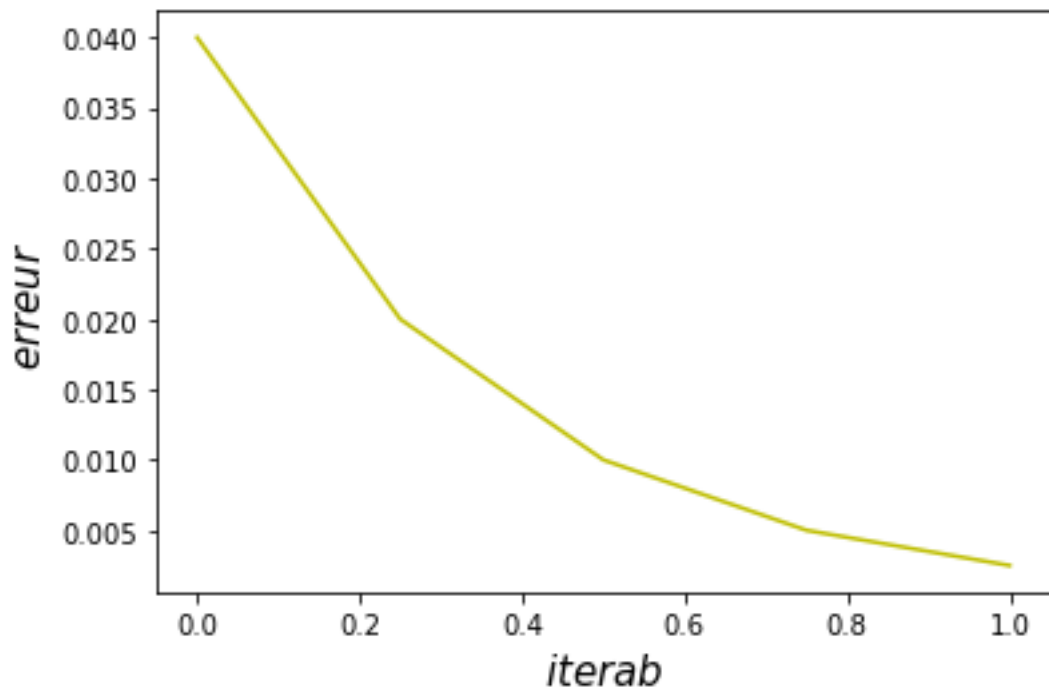
Traçons l'erreur pour $\text{err} = 0.04, 0.02, 0.01, 0.005, 0.0025$

```

1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 #u,t = -V u,x + k u,xx -lamda u + f
6
7 iplot=1
8
9 # PHYSICAL PARAMETERS
10 K = 0.01      #Diffusion coefficient
11 xmin = 0.0
12 xmax = 1.0
13 Time = 10.    #Integration time
14
15 V=1.
16 lamda=1
17
18 #mesh adaptation param
19
20 niter_refinement=5      #niter different calculations
21 hmin=0.01
22 hmax=0.1
23 err=0.01
24
25 # NUMERICAL PARAMETERS
26 NX = 4      #Number of grid points : initialization
27 NT = 10000   #Number of time steps max
28 ifre=1000000 #plot every ifre time iterations
29 eps=0.001    #relative convergence ratio
30
31 errorL2=np.zeros((niter_refinement))
32 errorH1=np.zeros((niter_refinement))
33 itertab=np.zeros((niter_refinement))
34 hloc = np.ones((NX))*hmax
35
36 iter=0
37 NX0=0
38
39 for i in range(1,5):
40     itertab[i]=i/NX
41 print(itertab)
42 erreur=[0.04, 0.02, 0.01, 0.005, 0.0025]

```

```
43  
44  
45 plt.xlabel(u'$iterab$', fontsize=15)  
46 plt.ylabel(u'$erreur$', fontsize=15)  
47 plt.plot(itertab, erreur, 'y')
```



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