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

def adrs(n, x):
#u,t = -V u,x + k u,xx  -lamda u + f
    u=np.ones(n)
    return u

def metric(n, u):
#calcul metric hloc
    hloc=np.ones(n)
    return hloc

def mesh(n, hloc):
#calcul metric hloc
    x=np.ones(n)
    return x

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=0.15
err=0.01

# NUMERICAL PARAMETERS
NX = 3    #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
```

```
while( np.abs(NX0-NX) > 2 and iter<niter_refinement-1):

    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]
        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] = 2*np.exp(-100*(x[j]-(xmax+xmin)*0.25)**2)+np.exp(-200*(x[j]-(xmax+xmin)*0.65)**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]    #ux a droite = 0

    #metric[NX-1]=2*metric[NX-2]-metric[NX-3]   #uxx a droite =0

    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

    errorL2[iter]=errL2h
    errorH1[iter]=errL2h+errH1h


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

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

plt.show()


plt.plot(errorL2[:niter_refinement-1],1/itertab[:niter_refinement-1],label="NB pt vs Error")
plt.legend()
plt.show()
```