```
"""
Author   : Abraham Flores
File     : HW3.py
Language : Python 3.6
Created  : 3/14/2018
Edited   : 3/16/2018

San Digeo State University
MTH 693b : Computational Partial Differential Equations

Strikwerda 6.3.10 : Parabolic Equations

Heat Equation:
    u_t = b*u_xx

    x = [-1,1]
    t = [0,1/2]

             |   1  for |x| < 1/2
    u_0(x) = |  1/2 for |x| = 1/2
             |   0  for |x| > 1/2

    Exact Solution and Boundaries given by:

        u(t,x) =
1/2 +2*
SUM[(-1)**i*(cos(pi*x*(2*i+1)))/(pi*(2*i+1))*exp(-t*pi**2(2*i+1)**2)]
(i=0,inf)

    Crank-Nicolson(6.3.4) :
    h = 1/10, 1/20, 1/40

    Compare lambda = 1 and mu = 10

    Demonstrate by the computations that when lambda is constant,
    the error in the solution does not decrease when measured
    in the supremum norm, but it does decrease in the L2 norm.

"""
import os,glob
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from scipy.sparse import diags

#Generators Exact Solution
def Exact(t,x,lim):
    value = 0
    for i in range(lim):
        numerator = np.cos(np.pi*x*(2*i+1))
        denominator = np.pi*(2*i+1)
        decay = np.exp(-t*np.pi**2*(2*i+1)**2)
        sign = (-1)**i
        value += sign*(numerator/denominator)*decay
    return 0.5 + 2*value

#Generates intial value function
def intial_foo(x):
    if abs(x) < 0.5:
        return 1
    if abs(x) == 0.5:
        return 0.5
    if abs(x) > 0.5:
        return 0

def plot(x,U,bounds,time,title,fileLoc):
    sns.set(font_scale = 2)
    sns.set_style("darkgrid", {"axes.facecolor": ".9"})
    fig,ax = plt.subplots()
    fig.set_size_inches(8,8)
    plt.plot(x,U,linewidth=3.0,label="t = "+ str(round(time,3)),color="r")
    plt.axis(bounds)
    plt.xlabel('x (Spatial)')
```

```
    plt.ylabel('U(t,x)')
    plt.title(title)

    plt.legend()
    plt.savefig(fileLoc+".png")
    plt.close()

def makeGif(gifName):
    os.chdir('Figures')
    #Create txt file for gif command
    fileList = glob.glob('*.png') #star grabs everything,
    fileList.sort()
    #writes txt file
    file = open('FileList.txt', 'w')
    for item in fileList:
        file.write("%s\n" % item)
    file.close()

    os.system('convert -delay 10 @FileList.txt ' + gifName + '.gif')
    os.system('del FileList.txt')
    os.system('del *.png')
    os.chdir('..')

def Crank_Nicolson(h,Lamb):
    b = 1
    mu = Lamb/h
    #generate array of intial values at t = 0
    X = np.arange(0-1,1+h,h)
    #dimension of our matrix
    dim = len(X)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    current_ = np.array(temp)
    #Factored out -b*mu/2
    NEXT = np.array([np.ones(dim-1),-2*(1+1/(b*mu))*np.ones(dim),np.ones(dim-1)])
    CURRENT = np.array([-1*np.ones(dim-1),2*(1-1/(b*mu))*np.ones(dim),-1*np.ones(dim
-1)])

    offset = [-1,0,1]#Location of each diagonal
    LEFT = diags(NEXT,offset).toarray()#Generate Matrix (n+1)
    RIGHT = diags(CURRENT,offset).toarray()#Generate Matrix (n)
    #Embed boundary conditions on matrix
    LEFT[0] *= 0
    LEFT[-1] *= 0
    LEFT[0][0] = 1
    LEFT[-1][-1] = 1
    RIGHT[0] *= 0
    RIGHT[-1] *= 0
    RIGHT[0][0] = 1
    RIGHT[-1][-1] = 1


    steps = int(0.5/(Lamb*h)) + 1
    for time in range(1,steps):
        #plot
        title = "6.3.10: Parabolic Equations"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\CN" + str_time
        bounds = [-1,1,0,1]
        plot(X,current_,bounds,time*Lamb*h,title,outFile)

        #implement Scheme
        next_ = \
   np.linalg.tensorsolve((-b*mu/2)*LEFT,(-b*mu/2)*np.matmul(RIGHT,current_))

        #Boundary Conditions
        next_[-1] = Exact(time*Lamb*h,1,15)
        next_[0]  = Exact(time*Lamb*h,-1,15)

        current_ = next_
```

```
    #makeGif
    makeGif("Crank_Nicolson_h_"+str(h)+"_Lambda_"+str(Lamb))

def ExactGIF(h,Lamb):
    #generate array of intial values at t = 0
    X = np.arange(0-1,1+h,h)

    temp = []
    for dx in X:
        temp.append(intial_foo(dx))
    #plot
    title = "6.3.10: Parabolic Equations"
    str_time = '0000'
    outFile = "Figures\exact" + str_time
    bounds = [-1,1,0,1]
    plot(X,np.asarray(temp),bounds,0,title,outFile)

    steps = int(0.5/(Lamb*h)) + 1
    for time in range(1,steps):
        t = time*Lamb*h
        sol_t = Exact(t,X,25)

        #plot
        title = "6.3.10: Parabolic Equations"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\exact" + str_time
        plot(X,sol_t,bounds,t,title,outFile)

    #makeGif
    makeGif("Exact_Solution_h_"+str(h)+"_Lambda_"+str(Lamb))

def ErrorGIF(h,Lamb):
    b = 1
    mu = Lamb/h
    #generate array of intial values at t = 0
    X = np.arange(0-1,1+h,h)
    #dimension of our matrix
    dim = len(X)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    current_ = np.array(temp)
    #Factored out -b*mu/2
    NEXT = np.array([np.ones(dim-1),-2*(1+1/(b*mu))*np.ones(dim),np.ones(dim-1)])
    CURRENT = np.array([-1*np.ones(dim-1),2*(1-1/(b*mu))*np.ones(dim),-1*np.ones(dim
-1)])

    offset = [-1,0,1]#Location of each diagonal
    LEFT = diags(NEXT,offset).toarray()#Generate Matrix (n+1)
    RIGHT = diags(CURRENT,offset).toarray()#Generate Matrix (n)
    #Embed boundary conditions on matrix
    LEFT[0] *= 0
    LEFT[-1] *= 0
    LEFT[0][0] = 1
    LEFT[-1][-1] = 1
    RIGHT[0] *= 0
    RIGHT[-1] *= 0
    RIGHT[0][0] = 1
    RIGHT[-1][-1] = 1


    steps = int(0.5/(Lamb*h)) + 1
    for time in range(1,steps):
        t = time*Lamb*h

        sol_t = Exact(t,X,15)
        #implement Scheme
        next_ = \
   np.linalg.tensorsolve((-b*mu/2)*LEFT,(-b*mu/2)*np.matmul(RIGHT,current_))

        #Boundary Conditions
        next_[-1] = Exact(t,1,15)
```

```
        next_[0]  = Exact(t,-1,15)

        err = abs(sol_t - next_)
        current_ = next_

        #plot
        title = "6.3.10: Parabolic Equations"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\err" + str_time
        bounds = [-1,1,0,1]
        plot(X,err,bounds,t,title,outFile)

    #makeGif
    makeGif("ERROR_h_"+str(h)+"_Lambda_"+str(Lamb))

def best_fit(X, Y):

    xbar = sum(X)/len(X)
    ybar = sum(Y)/len(Y)
    n = len(X) # or len(Y)

    numer = sum([xi*yi for xi,yi in zip(X, Y)]) - n * xbar * ybar
    denum = sum([xi**2 for xi in X]) - n * xbar**2

    b = numer / denum
    a = ybar - b * xbar

    return a, b

def INFNORM_plot(h,infNORM,LAMBDA):
    sns.set(font_scale = 2)
    sns.set_style("darkgrid", {"axes.facecolor": ".9"})
    fig,ax = plt.subplots()
    fig.set_size_inches(14.4,9)
    plt.scatter(h,infNORM,linewidth=3.0,color="r")
    plt.xlim(1, 2)
    plt.xlabel(r'$-Log_{10}$[dx]')
    plt.ylabel(r'$-Log_{10}$[INFINITY NORM]')
    plt.title("Lambda: "+str(LAMBDA)+" -- TIME: 0.5")

    a, b = best_fit(h, infNORM)
    yfit = [a + b * xi for xi in h]
    plt.plot(h, yfit,color="k",label="SLOPE: "+str(round(b,5)))
    plt.legend()

    plt.savefig("Figures/Err/INFNORMerr_LAMBDA_"+str(LAMBDA)+".png")
    plt.close()

def L2NORM_plot(h,L2norm,LAMBDA):
    sns.set(font_scale = 2)
    sns.set_style("darkgrid", {"axes.facecolor": ".9"})
    fig,ax = plt.subplots()
    fig.set_size_inches(14.4,9)
    plt.scatter(h,L2norm,linewidth=3.0,color="r")
    plt.xlim(1, 2)
    plt.xlabel(r'$-Log_{10}$[dx]')
    plt.ylabel(r'$-Log_{10}$[L2 NORM]')
    plt.title("Lambda: "+str(LAMBDA)+" -- TIME: 0.5")

    a, b = best_fit(h, L2norm)
    yfit = [a + b * xi for xi in h]
    plt.plot(h, yfit,color="k",label="SLOPE: "+str(round(b,5)))
    plt.legend()

    plt.savefig("Figures/Err/L2NORMerr_LAMBDA_"+str(LAMBDA)+".png")
    plt.close()

def ErrNorms(h,Lamb):
    b = 1
    mu = Lamb/h
    #generate array of intial values at t = 0
    X = np.arange(0-1,1+h,h)
    #dimension of our matrix
```

```
    dim = len(X)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    current_ = np.array(temp)
    #Factored out -b*mu/2
    NEXT = np.array([np.ones(dim-1),-2*(1+1/(b*mu))*np.ones(dim),np.ones(dim-1)])
    CURRENT = np.array([-1*np.ones(dim-1),2*(1-1/(b*mu))*np.ones(dim),-1*np.ones(dim
-1)])

    offset = [-1,0,1]#Location of each diagonal
    LEFT = diags(NEXT,offset).toarray()#Generate Matrix (n+1)
    RIGHT = diags(CURRENT,offset).toarray()#Generate Matrix (n)
    #Embed boundary conditions on matrix
    LEFT[0] *= 0
    LEFT[-1] *= 0
    LEFT[0][0] = 1
    LEFT[-1][-1] = 1
    RIGHT[0] *= 0
    RIGHT[-1] *= 0
    RIGHT[0][0] = 1
    RIGHT[-1][-1] = 1

    steps = int(0.5/(Lamb*h)) + 1
    L2_last = []
    infNORM = []
    for time in range(1,steps):
        t = time*Lamb*h

        sol_t = Exact(t,X,15)
        #implement Scheme
        next_ = \
   np.linalg.tensorsolve((-b*mu/2)*LEFT,(-b*mu/2)*np.matmul(RIGHT,current_))

        #Boundary Conditions
        next_[-1] = Exact(t,1,15)
        next_[0]  = Exact(t,-1,15)

        err = sol_t - next_
        infNORM.append(-1*np.log10(max(abs(err))))
        L2_last.append(-1*np.log10(np.sqrt(sum(err*err))))
        current_ = next_

    return infNORM[-1],L2_last[-1]


if __name__ == "__main__":
    mu = 10
    infi = []
    L2_data= []
    h = []
    for i in range(10,110,10):
        print(i)
        inf,L2 = ErrNorms(1.0/i,1.0/50)
        infi.append(inf)
        L2_data.append(L2)
        h.append(-1*np.log10(1.0/i))

    INFNORM_plot(h,infi,1.0/50)
    L2NORM_plot(h,L2_data,1.0/50)

    dx = [1/10,1/20,1/40]
    LAMBDA = 1.0
    mu = 10
    for h in dx:
        Crank_Nicolson(h,LAMBDA)
        ExactGIF(h,LAMBDA)
        ErrorGIF(h,LAMBDA)

        if h != 1/10:
            Crank_Nicolson(h,mu*h)
            ExactGIF(h,mu*h)
```

```
            ErrorGIF(h,mu*h)

'''
Report.

We see that the inf norm of the error decreases faster than the
L2 Norm of the error. From the figure plots.
'''
```