```
"""
Author   : Abraham Flores
File     : p6311.py
Language : Python 3.6
Created  : 3/20/2018
Edited   : 3/21/2018

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

Strikwerda 6.3.11 : Parabolic Equations

Heat Equation:
    u_t = b*u_xx

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

             |   1 - |x|  for |x| < 1/2
    u_0(x) = |    1/4     for |x| = 1/2
             |     0      for |x| > 1/2

    Exact Solution:

        u(t,x) =
3/8 +
SUM[(cos(2pi*x*(2*i+1)))/(pi**2*(2*i+1)**2)*exp(-4t*pi**2(2*i+1)**2)]
(i=0,inf)
+
SUM[((-1)**j/(pi*(2j+1))+(2)/(pi**2*(2j+1)**2))*cos(pi*x*(2*j+1))*exp(-t*pi**2*(2*j+
1)**2)]
(j=0,inf)

    h = 1/10, 1/20, 1/40, 1/80

    a) Foward-Time Central-Space : mu = .4
    b) Crank-Nicolson(6.3.4)     : lambda = 1 / mu = 1/h
    c) Crank-Nicolson(6.3.4)     : mu = 5

    Boundaries:
        u(t,-1) = Exact
        u_x(t,1) = 0
        V(n,M+1) = V(n,M-1)

"""
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):
    sum1 = 0
    sum2 = 0
    for i in range(lim):
        #First Sum
        #################################
        numerator = np.cos(2*np.pi*x*(2*i+1))
        denominator = np.pi**2*(2*i+1)**2
        decay = np.exp(-4*t*np.pi**2*(2*i+1)**2)
        sum1 += numerator/denominator*decay
        #################################
        #Second Sum
        #################################
        term1 = (-1)**i/(np.pi*(2*i+1))
        term2 = 2.0/(np.pi**2*(2*i+1)**2)
        osc_decay = np.cos(np.pi*x*(2*i+1))*np.exp(-t*np.pi**2*(2*i+1)**2)
        sum2 += (term1 + term2)*osc_decay
        #################################
    return 3.0/8 + sum1 + sum2

#Generates intial value function
```

```
def intial_foo(x):
    if abs(x) < 0.5:
        return 1 - abs(x)
    if abs(x) == 0.5:
        return 0.25
    if abs(x) > 0.5:
        return 0
#Plot
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 plot_error(x,U,labels,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)
    colors = ["r","b","c"]
    for i in range(len(U)):
        plt.plot(x,U[i],linewidth=3.0,label=labels[i],color=colors[i])
    x_c = bounds[0] + (bounds[1]-bounds[0])/35
    y_c = bounds[-1] - (bounds[-1]-bounds[-2])/10
    ax.annotate("t = "+ str(round(time,3)),xy=(0,0) ,xytext=(x_c,y_c))
    plt.axis(bounds)
    plt.xlabel('x (Spatial)')
    plt.ylabel('U(t,x) or |Error|')
    plt.title(title)

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

"""
Makes a gif given a name and delay for each image in ms

--Assumes the images are in the figures directory
"""
def makeGif(gifName,delay):
    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 ' + str(delay) + ' @FileList.txt ' + gifName + '.gif')
    os.system('del FileList.txt')
    os.system('del *.png')
    os.chdir('..')

"""
Computes intercept and slope for an unweighted linear best fit
"""
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 plot_norm(scheme,h,mu,inf_norm,L2_norm):
    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,inf_norm,linewidth=3.0,color="r",label=r'$-Log_{10}$[INFINITY NORM
]')
    plt.scatter(h,L2_norm,linewidth=3.0,color="b",label=r'$-Log_{10}$[L2 NORM]')
    #plt.xlim(1, 2)
    plt.xlabel(r'$-Log_{10}$[dx]')
    plt.ylabel(r'$-Log_{10}$|ERROR|')
    plt.title(scheme +" mu: "+str(mu)+" -- TIME: 0.5")

    a_inf, b_inf = best_fit(h, inf_norm)
    yfit = [a_inf + b_inf * xi for xi in h]
    plt.plot(h, yfit,color="k",label="(INF) SLOPE: "+str(round(b_inf,5)))

    a_L2, b_L2 = best_fit(h, L2_norm)
    yfit = [a_L2 + b_L2 * xi for xi in h]
    plt.plot(h, yfit,color="k",label="(L2)  SLOPE: "+str(round(b_L2,5)))
    plt.legend()

    plt.savefig("Figures/Error/"+scheme+"_norm_err_mu_"+str(mu)+".png")
    plt.close()

"""
Uses the Crank-Nicolson scheme to solve the given parabolic equation
"""
def Crank_Nicolson(h,mu):
    b = 1
    k = b*mu

    #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))
    #intialize array v{n,m}
    next_ = np.array(temp)

    #Generate Left and right matrices
    NEXT = np.array([(-k/2)*np.ones(dim-1),(1+k)*np.ones(dim),(-k/2)*np.ones(dim-1)]
)
    CURRENT = np.array([(k/2)*np.ones(dim-1),(1-k)*np.ones(dim),(k/2)*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 + k
    LEFT[-1][-2] = -k
    RIGHT[0] *= 0
    RIGHT[-1] *= 0
    RIGHT[0][0] = 1
    RIGHT[-1][-1] = 1 - k
    RIGHT[-1][-2] = k

    title = "6.3.11: Crank-Nicolson: h: " +str(round(h,4)) + ", mu: " +str(mu)
```

```
    bounds = [-1,1,0,1]
    #plot
    outFile = "Figures\CN00000"
    plot(X,next_,bounds,0,title,outFile)

    steps = int(0.5/(mu*h**2)) + 2
    for t in range(1,steps):
        time = t*mu*h**2

        #implement Scheme
        next_ = np.linalg.tensorsolve(LEFT,np.matmul(RIGHT,next_))

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

        #plot
        str_time = '0'*(5-len(str(t)))+str(t)
        outFile = "Figures\CN" + str_time
        plot(X,next_,bounds,time,title,outFile)

    #makeGif
    makeGif("Crank_Nicolson_h_"+str(h)+"_mu_"+str(mu),10)

def FTCS(h,mu):
    b = 1
    k = b*mu
    #generate array of intial values at t = 0
    X = np.arange(-1,1+h,h)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    next_ = np.array(temp)

    title = "6.3.11: FTCS mu: " +str(round(mu,3))
    bounds = [-1,1,0,1]
    outFile = "Figures\FTCS00000"
    plot(X,next_,bounds,0,title,outFile)

    steps = int(0.5/(mu*h**2)) + 2
    for t in range(1,steps):
        time = t*mu*h**2

        #implement Scheme
        prev_ = next_
        #np.roll: postive shift => terms to the left, negative => terms to the right

        next_ = k*(np.roll(next_,1)+np.roll(next_,-1)) + (1-2*k)*next_
        #Boundary Conditions
        next_[-1] = 2*k*prev_[-2] + (1-2*k)*prev_[-1]
        next_[0]  = Exact(time,-1,15)

        #plot
        str_time = '0'*(5-len(str(t)))+str(t)
        outFile = "Figures\FTCS" + str_time
        plot(X,next_,bounds,time,title,outFile)

    #makeGif
    makeGif("FTCS_h_"+str(h)+"_mu_"+str(mu),10)

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 = "6.3.11: Exact Solution"
    str_time = '00000'
    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)) + 2
    for t in range(1,steps):
        time = t*Lamb*h
        sol_t = Exact(time,X,25)

        #plot
        str_time = '0'*(5-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),10)

def CN_error(h,mu,gif,n_img):
    b = 1
    #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))
    next_ = np.array(temp)

    k = b*mu
    NEXT = np.array([(-k/2)*np.ones(dim-1),(1+k)*np.ones(dim),(-k/2)*np.ones(dim-1)]
)
    CURRENT = np.array([(k/2)*np.ones(dim-1),(1-k)*np.ones(dim),(k/2)*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 + k
    LEFT[-1][-2] = -k
    RIGHT[0] *= 0
    RIGHT[-1] *= 0
    RIGHT[0][0] = 1
    RIGHT[-1][-1] = 1 - k
    RIGHT[-1][-2] = k

    if gif:
        title = "6.3.11: Crank-Nicolson: h: " +str(round(h,4)) + ", mu: " +str(mu)
        bounds = [-1,1,0,1]

    inf_norm = []
    L2_norm = []
    steps = int(0.5/(mu*h**2)) + 2
    for t in range(1,steps):
        time = t*mu*h**2
        sol_t = Exact(time,X,15)
        #implement Scheme
        next_ = np.linalg.tensorsolve(LEFT,np.matmul(RIGHT,next_))

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

        err = abs(sol_t - next_)
        inf_norm.append(-1*np.log10(max(err)))
        L2_norm.append(-1*np.log10(np.sqrt(sum(err*err))))

        #plot
        if gif and (t%n_img==0):
            str_time = '0'*(5-len(str(t)))+str(t)
            outFile = "Figures\CN_err" + str_time
            plot_error\
    (X,[sol_t,next_,err],["Exact","CN","|Error|"],bounds,time,title,outFile)
```

```
    if gif:
        #makeGif
        makeGif("CN_ERROR_h_"+str(h)+"_mu_"+str(mu),10)

    return inf_norm[-1],L2_norm[-1]

def FTCS_error(h,mu,gif,n_img):
    b = 1
    k = b*mu
    #generate array of intial values at t = 0
    X = np.arange(0-1,1+h,h)
    #dimension of our matrix

    temp = []
    for dx in X:
        temp.append(intial_foo(dx))
    next_ = np.array(temp)


    if gif:
        title = "6.3.11: FTCS: h: " +str(round(h,4)) + " mu: " +str(mu)
        bounds = [-1,1,0,1]

    inf_norm = []
    L2_norm = []
    steps = int(0.5/(mu*h**2)) + 2
    for t in range(1,steps):
        time = t*mu*h**2
        sol_t = Exact(time,X,15)
        #implement Scheme
        prev_ = next_
        #np.roll: postive shift => terms to the left, negative => terms to the right

        next_ = k*(np.roll(next_,1)+np.roll(next_,-1)) + (1-2*k)*next_
        #Boundary Conditions
        next_[-1] = 2*k*prev_[-2] + (1-2*k)*prev_[-1]
        next_[0]  = Exact(time,-1,15)

        err = abs(sol_t - next_)
        inf_norm.append(-1*np.log10(max(err)))
        L2_norm.append(-1*np.log10(np.sqrt(sum(err*err))))

        #plot
        if gif and (t%n_img==0):
            str_time = '0'*(5-len(str(t)))+str(t)
            outFile = "Figures\FTCS_err" + str_time
            plot_error\
    (X,[sol_t,next_,err],["Exact","FTCS","|Error|"],bounds,time,title,outFile)

    if gif:
        #makeGif
        makeGif("FTCS_ERROR_h_"+str(h)+"_mu_"+str(mu),10)

    return inf_norm[-1],L2_norm[-1]

if __name__ == "__main__":
    inf_all = []
    L2_all = []
    h = []
    for i in range(10,110,10):
        inf,L2 = FTCS_error(1.0/i,0.4,False)
        inf_all.append(inf)
        L2_all.append(L2)
        h.append(-1*np.log10(1.0/i))

    plot_norm("FTCS",h,0.4,inf_all,L2_all)

    inf_all = []
    L2_all = []
    h = []
    for i in range(10,110,10):
        inf,L2 = CN_error(1.0/i,5,False)
        inf_all.append(inf)
```

```
        L2_all.append(L2)
        h.append(-1*np.log10(1.0/i))

    plot_norm("Crank-Nicolson",h,5,inf_all,L2_all)

    inf_all = []
    L2_all = []
    h = []
    for i in range(10,110,10):
        inf,L2 = CN_error(1.0/i,i,False)
        inf_all.append(inf)
        L2_all.append(L2)
        h.append(-1*np.log10(1.0/i))

    plot_norm("Crank-Nicolson",h,r"$h^{-1}$",inf_all,L2_all)
    #The plots are moved to error directory so they do not get deleted
    dx = [1/10,1/20]

    for h in dx:
        CN_error(h,5,True,1)
        CN_error(h,1/h,True,1)
        FTCS_error(h,0.4,True,1)

    dx = [1/40,1/80]

    for h in dx:
        CN_error(h,5,True,(1/(8*h)))
        CN_error(h,1/h,True,1)
        FTCS_error(h,0.4,True,(1/(8*h)))

'''
Report.

The accuracy of both schemes are reasonably accurate, however the FTCS scheme
is extremly fast due being explicit and without matrix multiplication.
Although it is essentialy matrix multiplication.
'''
```