```
"""
Author   : Abraham Flores
File     : p1451.py
Language : Python 3.6
Created  : 5/4/2018
Edited   : 5/7/2018

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

Strikwerda 14.5.1 : Preconditioned Conjugate Gradiant Method

Possion's Equation:
    u_xx + u_yy = -4cos(x+y)sin(x-y)

    x = [0,1]
    y = [0,1]

    Exact Solution:

        u(x,y) = cos(x+y)sin(x-y)

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

    Boundaries: Exact Solution
    Intial Interior = 0
    tol = 10^(-6)

    SSOR Preconditioner:
    D = (1/omega)*Diagonal of A
    L = Lower triangular part of A
    M = 1.0/(2-omega)*(D+L)(D^[-1])(D+L)^[T]

http://www.netlib.org/linalg/html_templates/node58.html

    ***Useful Tips:
        Embed Boundary Conditions on M
        PCG with SSOR is extremly sensitive between h and omega

"""
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.sparse import diags
import numpy as np

def Preconditioned_Conugate_Gradiant(A,x,b,M,tol):
    #Define Preconditioner Inverse
    M_inv = np.linalg.inv(M)

    #Intialize Variables
    residual = b - np.matmul(A,x)
    rho = np.matmul(M_inv,residual)
    delta_new = np.dot(residual,rho)

    tol_r = np.linalg.norm(residual)*tol
    converged = False
    iters = 0
    while(not converged):

        #Useful Bookeeping
        gamma = np.matmul(A,rho)

        alpha = delta_new/(np.dot(rho,gamma))   #Update Alpha
        x += alpha*rho                          #Update Grid
        residual -= alpha*gamma                 #Update Residual
        pre = np.matmul(M_inv,residual)         #Update Preconditioned Residual

        delta_old = delta_new
        delta_new = np.dot(residual,pre)

        beta = delta_new/delta_old              #Update beta
        rho = pre + beta*rho                    #Update rho
```

```
        iters += 1
        converged = np.linalg.norm(residual) < tol_r or iters == 2*len(A)

    return iters

def Conugate_Gradiant(A,x,b,tol):
    #Intialize Variables
    residual = b - np.matmul(A,x)
    rho = np.empty_like (residual)
    rho[:] = residual
    r_norm_new = np.dot(residual,residual)
    r_norm_prev = 0

    iters = 0
    converged = False
    tol_r = np.sqrt(r_norm_new)*tol

    while(not converged):

        #Useful coefficents
        gamma = np.matmul(A,rho)
        kappa = np.dot(rho,gamma)

        alpha = r_norm_new/kappa                #Update alpha
        x += alpha*rho                          #Update X
        residual -= alpha*gamma                 #Update residual

        r_norm_prev = r_norm_new
        r_norm_new = np.dot(residual,residual)

        beta = r_norm_new/r_norm_prev           #Update Beta
        rho = beta*rho + residual               #Update rho

        iters+=1
        converged = np.sqrt(r_norm_new) < tol_r or iters == 2*len(A)

    return iters

def cont_plot(x,y,U,title,fileLoc):
    sns.set(font_scale = 2.0)
    sns.set_style("darkgrid", {"axes.facecolor": ".9"})

    fig,ax = plt.subplots()
    fig.set_size_inches(14.4,9)
    xlocs = []
    xlabel = []
    for i in range(0,len(x),int(len(x)/5)):
        xlocs.append(i+1)
        xlabel.append(x[i])

    ylocs = []
    ylabel = []
    for i in range(0,len(y),int(len(y)/5)):
        ylocs.append(i+1)
        ylabel.append(y[i])


    plt.xticks(xlocs,xlabel,rotation=45)
    plt.yticks(ylocs,ylabel,rotation=45)
    # Plot the contour
    plt.pcolor(U,vmin=U.min(),vmax=U.max())
    #legend
    clb = plt.colorbar()
    clb.set_label(r'$U(t,X,Y)$', labelpad=40, rotation=270)
    plt.xlabel('X (spatial)')
    plt.ylabel('Y (spatial)')
    plt.title(title)

    plt.savefig(fileLoc+'.png')
    plt.close()

if __name__=="__main__":
```

```
    grid_spacing = [1/10.0,1/20.0,1/40.0]
    tol = 10**(-8)

    for h in grid_spacing:
        x = np.arange(0,1+h,h)
        y = np.arange(0,1+h,h)
        n = len(x)

        X,Y = np.meshgrid(x,y)

        N = n**2 #Length of one Side
        scheme = np.array([np.ones(N-n),np.ones(N-1),-4*np.ones((N)),np.ones(N-1),np
.ones(N-n)])
        offset = [-n,-1,0,1,n]#Location of each diagonal

        scheme = diags(scheme,offset).toarray()#Generate Matrix
        for i in range(n):
            scheme[i] *= 0
            scheme[-(i+1)] *= 0
            scheme[i][i] = 1
            scheme[-(i+1)][-(i+1)] = 1

            scheme[i*n] *= 0
            scheme[i*n][i*n] = 1
            scheme[(i+1)*n-1] *= 0
            scheme[(i+1)*n-1][(i+1)*n-1] = 1

        #intialize Grid
        grid = np.zeros((n,n))
        grid_forcing = -4*np.cos(X+Y)*np.sin(X-Y)*h**2

        grid[0] = np.cos(x)*np.sin(x)
        grid[-1] = np.cos(x+1.0)*np.sin(x-1.0)

        grid_forcing[0] = np.cos(x)*np.sin(x)
        grid_forcing[-1] = np.cos(x+1.0)*np.sin(x-1.0)

        for i in range(1,n-1):
            grid[i][0] = np.cos(y[i])*np.sin(-y[i])
            grid[i][-1] = np.cos(1.0+y[i])*np.sin(1.0-y[i])
            grid_forcing[i][0] = np.cos(y[i])*np.sin(-y[i])
            grid_forcing[i][-1] = np.cos(1.0+y[i])*np.sin(1.0-y[i])

        init = np.ndarray.flatten(grid)
        grid = np.ndarray.flatten(grid)
        grid_forcing = np.ndarray.flatten(grid_forcing)

        #Preconditioner -- SSOR
        omega = .90 #Tested omega = .1 -> 1.9 with steps of .1
        D = (1.0/omega)*np.diag(np.diag(scheme))
        L = np.tril(scheme,-1)
        D_inv = np.linalg.inv(D)

        C = D + L
        C_T = np.transpose(C)

        SSOR = 1.0/(2-omega)*np.matmul(C,np.matmul(D_inv,C_T))

        #Embed Boundary Condtions on Preconditioner
        for i in range(n):
            SSOR[i] *= 0
            SSOR[-(i+1)] *= 0
            SSOR[i][i] = 1
            SSOR[-(i+1)][-(i+1)] = 1

            SSOR[i*n] *= 0
            SSOR[i*n][i*n] = 1
            SSOR[(i+1)*n-1] *= 0
            SSOR[(i+1)*n-1][(i+1)*n-1] = 1


        #Precondtioned Conjugate Gradiant
        iters = Preconditioned_Conugate_Gradiant(scheme,grid,grid_forcing,SSOR,tol)
```

```
        print("PCG | h: " +str(h) + " | N: "+str(n))
        print("Iterations: "+str(iters))

        grid = np.reshape(grid,(n,n))

        #Exact Solution
        exact = np.cos(X+Y)*np.sin(X-Y)
        #Reshape
        grid = np.reshape(grid,(n,n))

        err = abs(grid-exact)
        INFNORM = np.max(np.max(err))
        L2 = np.sqrt(sum(sum(err*err)))
        print("L2 NORM of Error: "+str(L2))
        print("INFNORM of Error: "+str(INFNORM))
        print("#"*25)

        path = "D:/SDSU/MTH693b/Strikwerda-Problems/Chapter-14/Section-5/Problem-1/F
igures/"
        #plot
        cont_plot(x,y,exact,"EXACT h: "+str(h),path+"EXACT_h_"+str(h))
        cont_plot(x,y,grid,"Preconditioned Conjugate Gradiant h: "+str(h),path+"PCG_
h_"+str(h))
        cont_plot(x,y,err,"ERROR h: "+str(h),path+"ERROR_PCG_h_"+str(h))

        grid = init
        iters = Conugate_Gradiant(scheme,grid,grid_forcing,tol)
        print("CG | h: " +str(h) + " | N: "+str(n))
        print("Iterations: "+str(iters))

        grid = np.reshape(grid,(n,n))

        #Exact Solution
        exact = np.cos(X+Y)*np.sin(X-Y)
        #Reshape
        grid = np.reshape(grid,(n,n))

        err = abs(grid-exact)
        INFNORM = np.max(np.max(err))
        L2 = np.sqrt(sum(sum(err*err)))
        print("L2 NORM of Error: "+str(L2))
        print("INFNORM of Error: "+str(INFNORM))
        print("#"*25)
```