```
"""
Author   : Abraham Flores
File     : p1374.py
Language : Python 3.6
Created  : 4/22/2018
Edited   : 5/5/2018 -- Fixed SOR on Boundaries... Halved iterations

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

Strikwerda 13.7.4 : Neumann Boundary Conditions

Posson's Equation:
    u_xx + u_yy = -2pi^(2)*cos(pi*x)cos(pi*y)

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

    Exact Solution:

        u(x,y) = cos(pi*x)cos(pi*y)

    h = 1/10, 1/20, 1/40
    omega = 2/(1+pi*h/(sqrt(2))

    Boundaries:
      du/dn = 0

      13.7.3: Second Order Forward/Reverse Difference
      13.7.4: First Order Foward/Reverse Difference

    Stop iterations at Tolerance of order 7 : (10^(-7))
"""

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

"""
Successive Over Relaxation Method:
    Solves Ax = b
    an iteravive method with a given tolerance to achieve
    First Order Neumann Boundary Conditions

    Parameters:
        grid: grid points (x)
        grid_forcing: Related forcing term in possion's equations (b)
        n : Length and Width of A
        omega : relaxation factor ([0,2])
        tol : tolerance to achieve

    Returns:
        Number of Iterations Ran
"""
def SOR_FO(grid,grid_forcing,n,omega,tol):
#Assume grid is intialized
#repeat until convergence
  iters = 0
  converged = False
  while(not converged):
      change = 0
      #loop over inner grid
      change += sum([x**2 for x in omega*(grid[1] - grid[0])])
      grid[0] += omega*(grid[1] - grid[0])

      for i in range(1,n-1):
          point_change = omega*(grid[i][1] - grid[i][0])
          grid[i][0] += point_change
          change += (point_change)**2

          for j in range(1,n-1):
              #SOR METHOD
              sigma = grid[i+1][j] + grid[i-1][j] + grid[i][j+1] +grid[i][j-1]
```

```
                point_change = omega*((grid_forcing[i][j]-sigma)/(-4)-grid[i][j])
                grid[i][j] += point_change

                #ADD to L2 change Norm
                change += (point_change)**2

            point_change = omega*(grid[i][-2] - grid[i][-1])
            grid[i][-1] += point_change
            change += (point_change)**2

        change += sum([x**2 for x in omega*(grid[-2] - grid[-1])])
        grid[-1] += omega*(grid[-2] - grid[-1])

        iters+=1
        #check if convergence is reached
        if (np.sqrt(change) < tol):
            converged = True

    return iters

"""
Successive Over Relaxation Method:
    Solves Ax = b
    an iteravive method with a given tolerance to achieve
    Second Order Neumann Boundary Conditions

    Parameters:
        grid: grid points (x)
        grid_forcing: Related forcing term in possion's equations (b)
        n : Length and Width of A
        omega : relaxation factor ([0,2])
        tol : tolerance to achieve

    Returns:
        Number of Iterations Ran
"""
def SOR_SO(grid,grid_forcing,n,omega,tol):
#Assume grid is intialized
#repeat until convergence
    iters = 0
    converged = False
    while(not converged):
        change = 0
        #loop over inner grid
        point_change = omega*((-4*grid[1] + grid[2])/(-3) - grid[0])
        change += sum(point_change*point_change)
        grid[0] += point_change

        for i in range(1,n-1):
            point_change = omega*((-4*grid[i][1] + grid[i][2])/(-3)-grid[i][0])
            grid[i][0] += point_change
            change += point_change*point_change

            for j in range(1,n-1):
                #SOR METHOD
                sigma = grid[i+1][j] + grid[i-1][j] + grid[i][j+1] +grid[i][j-1]
                point_change = omega*((grid_forcing[i][j]-sigma)/(-4)-grid[i][j])
                grid[i][j] += point_change

                #ADD to L2 change Norm
                change += (point_change)**2

            point_change = omega*((-4*grid[i][-2] + grid[i][-3])/(-3)-grid[i][-1])
            grid[i][-1] += point_change
            change += point_change*point_change

        point_change = omega*((-4*grid[-2] + grid[-3])/(-3) - grid[-1])
        grid[-1] += point_change
        change += sum(point_change*point_change)


        iters+=1
        #check if convergence is reached
```

```
        if (np.sqrt(change) < tol):
            converged = True

    return iters

def surf_plot(x,y,U,bounds,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)

    X,Y = np.meshgrid(x,y)
    plt.xlim(0,1)
    plt.ylim(0,1)
    plt.xticks(rotation=45)
    plt.yticks(rotation=45)
    # Plot the contour
    plt.pcolor(X, Y, U,vmin=bounds[0],vmax=bounds[1])
    #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**(-7)

    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)

        grid_forcing = -2*np.pi**2*np.cos(np.pi*X)*np.cos(np.pi*Y)*h**2
        omega = 2/(1+np.pi*h/np.sqrt(2))

        #intialize Grid
        grid = np.zeros((n,n))

        #RUN SOR
        iters = SOR_FO(grid,grid_forcing,n,omega,tol)
        print("First Order Iterations: "+str(iters))

        #Exact Solution
        exact = np.cos(np.pi*X)*np.cos(np.pi*Y)

        #plot
        surf_plot(x,y,exact,[-1,1],"EXACT h: "+str(h),"Figures/EXACT_h_"+str(h))
        surf_plot(x,y,grid,[-1,1],"SOR First Order h: "+str(h),"Figures/SOR_FO_h_"+s
tr(h))
        surf_plot(x,y,abs(grid-exact),[0,np.max(np.max(abs(grid-exact)))],"ERROR (Fi
rst Order) h: "+str(h),"Figures/ERROR_FO_h_"+str(h))

        #intialize Grid
        grid = np.zeros((n,n))

        #Run SOR
        iters = SOR_SO(grid,grid_forcing,n,omega,tol)
        print("Second Order Iterations: "+str(iters))

        #plot
        surf_plot(x,y,grid,[-1,1],"SOR Second Order h: "+str(h),"Figures/SOR_SO_h_"+
str(h))
        surf_plot(x,y,abs(grid-exact),[0,np.max(np.max(abs(grid-exact)))],"ERROR (Se
cond Order) h: "+str(h),"Figures/ERROR_SO_h_"+str(h))
```