```python
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 21 11:34:56 2020

@author: Wazowskai
"""

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



#problem constants
nu=1e-6
mu=1e-3
rho=1e+3
dt=0.0001
gradP=-2.4
n_iter=0
'''
node generation section
'''
#domain length

Lx1=0.02
Lx2=0.01

#number of cells on each direction
Nx1=60
Nx2=30

#mesh spacing
h=Lx1/Nx1

#uave
u1_ave=0.02

def Adv_x_n(i,j):
    return (1/h)*((0.5*(cell_S_x_un[i,j]+cell_S_x_un[i+1,j]))**2-(0.5*(cell_S_x_un[i,j]+cell_S_x_un[i-1,j]))**2+(0.5*(cell_S_x_un[i,j+1]+cell_S_x_un[i,j]))*(cell_S_y_vn[i,j+1]+cell_S_y_vn[i+1,j+1])-(0.5*(cell_S_x_un[i,j]+cell_S_x_un[i,j-1]))*(0.5*(cell_S_y_vn[i,j]+cell_S_y_vn[i+1,j])))
def Dif_x_n(i,j):
    return (1/(h**2))*(cell_S_x_un[i+1,j]+cell_S_x_un[i-1,j]+cell_S_x_un[i,j+1]+cell_S_x_un[i,j-1]-4*cell_S_x_un[i,j])
#def Adv_y_n(i,j):
#    return (1/h)*((0.5*())**2-(0.5*())**2+(0.5*())*(0.5*())-(0.5*())*(0.5*()))
def Dif_y_n(i,j):
    return (1/(h**2))*(cell_S_y_vn[i+1,j]+cell_S_y_vn[i-1,j]+cell_S_y_vn[i,j+1]+cell_S_y_vn[i,j-1]-4*cell_S_y_vn[i,j])

def ref_vel_prof(x2):
    '''
    function returning reference analytic sol
    '''
    return -1200*((x2-0.005)**2)+0.03


#cell centroid coor
#the +2 stands for ghost cells on each direction
cell_cent_x=np.zeros([Nx1+2,Nx2+2])
cell_cent_y=np.zeros([Nx1+2,Nx2+2])

```

```
cell_cent_un=np.zeros([Nx1+2,Nx2+2])
cell_cent_us=np.zeros([Nx1+2,Nx2+2])
cell_cent_unn=np.zeros([Nx1+2,Nx2+2])

#cell corner coor
cell_cor_x=np.zeros([Nx1+3,Nx2+3])
cell_cor_y=np.zeros([Nx1+3,Nx2+3])

#surf area of the cell
cell_S_x=np.zeros([Nx1+2,Nx2+2])
cell_S_y=np.zeros([Nx1+2,Nx2+2])

cell_S_x_coor_x=np.zeros([Nx1+2,Nx2+2])
cell_S_x_coor_y=np.zeros([Nx1+2,Nx2+2])
cell_S_y_coor_x=np.zeros([Nx1+2,Nx2+2])
cell_S_y_coor_y=np.zeros([Nx1+2,Nx2+2])

#normal vector of cell surfaces
cell_S_x_nx=np.zeros([Nx1+2,Nx2+2])
cell_S_x_ny=np.zeros([Nx1+2,Nx2+2])
cell_S_y_nx=np.zeros([Nx1+2,Nx2+2])
cell_S_y_ny=np.zeros([Nx1+2,Nx2+2])
#surface velocities
cell_S_x_un=np.zeros([Nx1+2,Nx2+2])
cell_S_x_us=np.zeros([Nx1+2,Nx2+2])
cell_S_x_unn=np.zeros([Nx1+2,Nx2+2])

cell_S_x_vn=np.zeros([Nx1+2,Nx2+2])
cell_S_x_vs=np.zeros([Nx1+2,Nx2+2])
cell_S_x_vnn=np.zeros([Nx1+2,Nx2+2])
#cell_S_x_v=np.zeros([Nx1+2,Nx2+2])
cell_S_y_un=np.zeros([Nx1+2,Nx2+2])
cell_S_y_us=np.zeros([Nx1+2,Nx2+2])
cell_S_y_unn=np.zeros([Nx1+2,Nx2+2])

cell_S_y_vn=np.zeros([Nx1+2,Nx2+2])
cell_S_y_vs=np.zeros([Nx1+2,Nx2+2])
cell_S_y_vnn=np.zeros([Nx1+2,Nx2+2])
#cell_S_y_v=np.zeros([Nx1+2,Nx2+2])
#reference velocity profile
ref_S_u=np.zeros([Nx2+2])
L_sq=np.array([1.0,1.0])

#corner coor initialization
for j in range(0,Nx2+3):
    for i in range(0, Nx1+3):
        cell_cor_x[i,j]=(Lx1/Nx1)*(i-1)
        cell_cor_y[i,j]=(Lx2/Nx2)*(j-1)

#cell cent coor storage
for j in range(0, Nx2+2):
    for i in range(0, Nx1+2):

        cell_cent_x[i,j]='{:10.6e}'.format(0.25*(cell_cor_x[i,j]+cell_cor_x[i+1,j]+cell_cor_x[i,j+1]+cell_cor_x[i+1,j+1]))
        cell_cent_y[i,j]='{:10.6e}'.format(0.25*(cell_cor_y[i,j]+cell_cor_y[i+1,j]+cell_cor_y[i,j+1]+cell_cor_y[i+1,j+1]))
        cell_S_x_coor_x[i,j]=(cell_cor_x[i,j]+cell_cor_x[i,j+1])/2
        cell_S_x_coor_y[i,j]=(cell_cor_y[i,j]+cell_cor_y[i,j+1])/2
        cell_S_y_coor_x[i,j]=(cell_cor_x[i,j]+cell_cor_x[i+1,j])/2
        cell_S_y_coor_y[i,j]=(cell_cor_y[i,j]+cell_cor_y[i+1,j])/2
        #initial conditions
        cell_S_x_un[i,j]=0.03
        cell_S_y_un[i,j]=0.00

        cell_S_x[i,j]=abs(cell_cor_y[i,j]-cell_cor_y[i,j+1])
```

```
        cell_S_y[i,j]=abs(cell_cor_x[i,j]-cell_cor_x[i+1,j])

        cell_S_x_nx[i,j]=(cell_cor_y[i,j+1]-cell_cor_y[i,j])/cell_S_x[i,j]
        cell_S_x_ny[i,j]=(cell_cor_x[i,j+1]-cell_cor_x[i,j])/cell_S_x[i,j]
        cell_S_y_nx[i,j]=(cell_cor_y[i+1,j]-cell_cor_y[i,j])/cell_S_y[i,j]
        cell_S_y_ny[i,j]=(cell_cor_x[i+1,j]-cell_cor_x[i,j])/cell_S_y[i,j]

plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1], cell_S_x_un[1:Nx1+1, 1:Nx2+1], 20, cmap='inferno')
plt.colorbar()
plt.show()

L_sq_r=L_sq[1]/L_sq[0]
for i in range(1, Nx2+1):
    ref_S_u[i]=ref_vel_prof(cell_S_x_coor_y[0,i])

while L_sq_r<1.01:
    L_sq[0]=L_sq[1]
    #predictor step:
    for j in range(1, Nx2+1):
        for i in range(1, Nx1+1):
            #cell_S_x_us[i,j]=cell_S_x_un[i,j]+dt*(-Adv_x_n(i,j)+nu*Dif_x_n(i,j))
            cell_S_x_us[i,j]=cell_S_x_un[i,j]+dt*(nu*Dif_x_n(i,j))
#    #B.C. update
    for j in range(0, Nx2+2):
        cell_S_x_us[0,j]=cell_S_x_us[-2,j]
        cell_S_x_us[-1,j]=cell_S_x_us[1,j]
    for i in range(0, Nx1+2):
        cell_S_x_us[i,0]=-cell_S_y_us[i,1]
        cell_S_x_us[i,-1]=-cell_S_y_us[i,-2]

    #corrector step:
    for j in range(1, Nx2+1):
        for i in range(1, Nx1+1):
            cell_S_x_unn[i,j]=cell_S_x_us[i,j]-(1/rho)*(dt)*(gradP)

    #B.C. update
    for j in range(0, Nx2+2):
        cell_S_x_unn[0,j]=cell_S_x_unn[-2,j]
        cell_S_x_unn[-1,j]=cell_S_x_unn[1,j]
    for i in range(0, Nx1+2):
        cell_S_x_unn[i,0]=-cell_S_y_unn[i,1]
        cell_S_x_unn[i,-1]=-cell_S_y_unn[i,-2]

    for j in range(1, Nx2+1):
        for i in range(1, Nx1+1):
            cell_S_x_un[i,j]=cell_S_x_unn[i,j]
    for j in range(0, Nx2+2):
        cell_S_x_un[0,j]=cell_S_x_un[-2,j]
        cell_S_x_un[-1,j]=cell_S_x_un[1,j]
    for i in range(0, Nx1+2):
        cell_S_x_un[i,0]=-cell_S_y_un[i,1]
        cell_S_x_un[i,-1]=-cell_S_y_un[i,-2]

    sq_sum_error=0

    for i in range(1,Nx2+1):
        sq_sum_error+=(ref_S_u[i]-cell_S_x_un[50,i])**2
    L_sq[1]=math.sqrt(sq_sum_error/(Nx2+1))

    if n_iter%10000==0:
        print('iter= '+str(n_iter)+', L_sq= {:.4e}'.format(L_sq[0]))
        plt.plot(cell_S_x_un[50,1:Nx2+1],cell_S_x_coor_y[50,1:Nx2+1], color='navy', label='numerical sol, $L^2$= {:10.4e}'.format(L_sq[0]))
        plt.plot(ref_S_u[1:Nx2+1] ,cell_S_x_coor_y[50,1:Nx2+1], color='red', label='reference')
        plt.xlabel('$u_1$ ($m/s$)')
```

```
            plt.ylabel('$x_2$ (m)')
            plt.legend()

            plt.grid()
            plt.show()

            plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1],
            cell_S_x_un[1:Nx1+1, 1:Nx2+1], 20, cmap='inferno')
            plt.colorbar()
            plt.xlabel('$x_1$ (m)')
            plt.ylabel('$x_2$ (m)')
            plt.title('domain $u_1$ contour ($m/s$)')
            plt.show()
        L_sq_r=L_sq[1]/L_sq[0]

        n_iter+=1


    print('iter= '+str(n_iter)+', L_sq= {:.4e}'.format(L_sq[0]))
    plt.plot(cell_S_x_un[50,1:Nx2+1],cell_S_x_coor_y[50,1:Nx2+1], color='navy',
    label='numerical sol, $L^2$= {:10.4e}'.format(L_sq[0]))
    plt.plot(ref_S_u[1:Nx2+1] ,cell_S_x_coor_y[50,1:Nx2+1], color='red', label='reference')
    plt.xlabel('$u_1$ ($m/s$)')
    plt.ylabel('$x_2$ (m)')
    plt.legend()
    plt.grid()
    plt.show()

    plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1],
    cell_S_x_un[1:Nx1+1, 1:Nx2+1], 20, cmap='inferno')
    plt.colorbar()
    plt.xlabel('$x_1$ (m)')
    plt.ylabel('$x_2$ (m)')
    plt.title('domain $u_1$ contour ($m/s$)')
    plt.show()
```