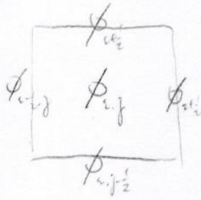
NE 577 Hwk 5

~~Stu~~ Name : Cheng-Kai Tai
StuID : 200328541.

5.1.



for $\phi_{i,j+\frac{1}{2}}$

$$\begin{cases} \phi_{i,j} + \frac{1}{2} M(D_y^+ \phi_{i,j}, D_y^- \phi_{i,j}), & \frac{1}{2}(v_{i,j+1} + v_{i,j}) > 0 \\ \phi_{i,j+1} - \frac{1}{2} M(D_y^+ \phi_{i,j+1}, D_y^- \phi_{i,j+1}), & \frac{1}{2}(v_{i,j+1} + v_{i,j}) < 0. \end{cases}$$

where $D_y^+ \phi_{i,j} \equiv \phi_{i,j+1} - \phi_{i,j}$ $\quad D_y^+ \phi_{i,j+1} \equiv \phi_{i,j+2} - \phi_{i,j+1}$

$D_y^- \phi_{i,j} \equiv \phi_{i,j} - \phi_{i,j-1}$. , $\quad D_y^- \phi_{i,j+1} \equiv \phi_{i,j+1} - \phi_{i,j}$

for $\frac{1}{2}(v_{i,j+1} + v_{i,j}) > 0$.

① $|D_y^+ \phi_{i,j}| < |D_y^- \phi_{i,j}| \rightarrow \phi_{i,j+\frac{1}{2}} = \phi_{i,j} + \frac{1}{2} D_y^+ \phi_{i,j} = \phi_{i,j} + \frac{1}{2}(\phi_{i,j+1} - \phi_{i,j}) = \dfrac{\phi_{i,j+1} + \phi_{i,j}}{2}$.

② $|D_y^- \phi_{i,j}| < |D_y^+ \phi_{i,j}| \rightarrow \phi_{i,j+\frac{1}{2}} = \phi_{i,j} + \frac{1}{2} D_y^- \phi_{i,j} = \phi_{i,j} + \frac{1}{2}(\phi_{i,j} - \phi_{i,j-1}) = \dfrac{3\phi_{i,j} - \phi_{i,j-1}}{2}$

for $\frac{1}{2}(v_{i,j+1} + v_{i,j}) < 0$.

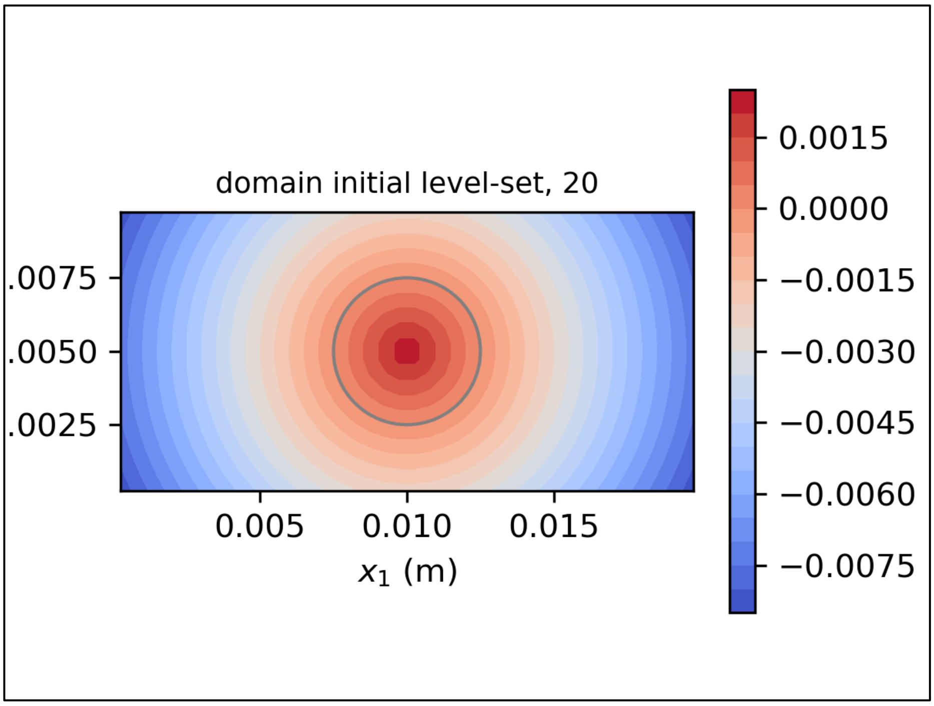
① $|D_y^+ \phi_{i,j+1}| < |D_y^- \phi_{i,j+1}| \rightarrow \phi_{i,j+\frac{1}{2}} = \phi_{i,j+1} - \frac{1}{2} D_y^+ \phi_{i,j+1} = \phi_{i,j+1} - \frac{1}{2}(\phi_{i,j+2} - \phi_{i,j+1}) = \dfrac{3\phi_{i,j+1} - \phi_{i,j+2}}{2}$

② $|D_y^- \phi_{i,j+1}| < |D_y^+ \phi_{i,j+1}| \rightarrow \phi_{i,j+\frac{1}{2}} = \phi_{i,j+1} - \frac{1}{2} D_y^- \phi_{i,j+1} = \phi_{i,j+1} - \frac{1}{2}(\phi_{i,j+1} - \phi_{i,j}) = \dfrac{\phi_{i,j+1} + \phi_{i,j}}{2}$

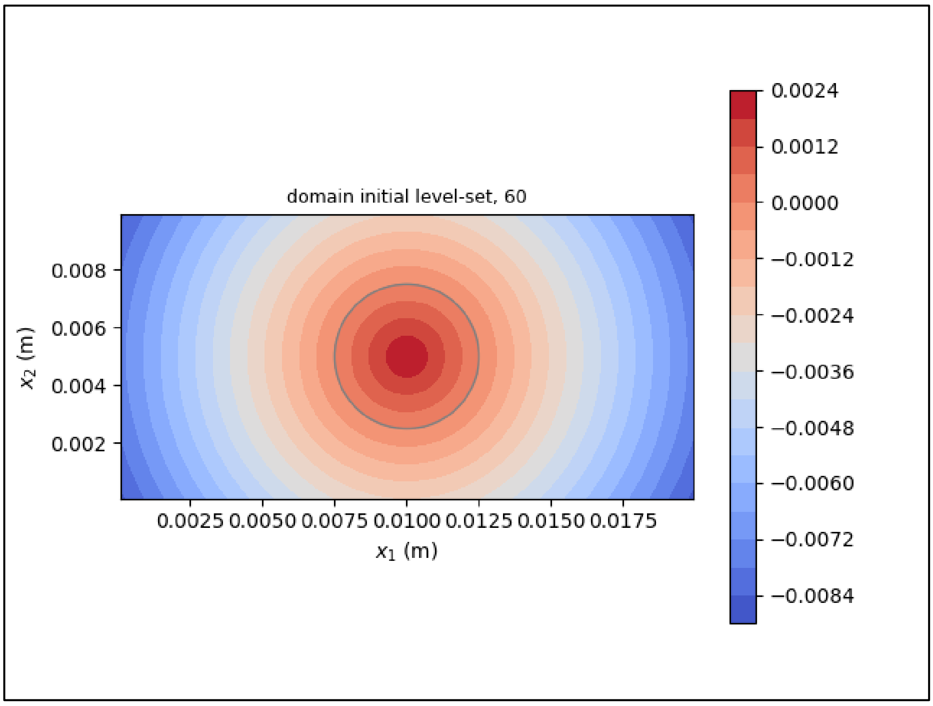
#5.2

(a)

Initial level-set as required, 20 cells across domain height



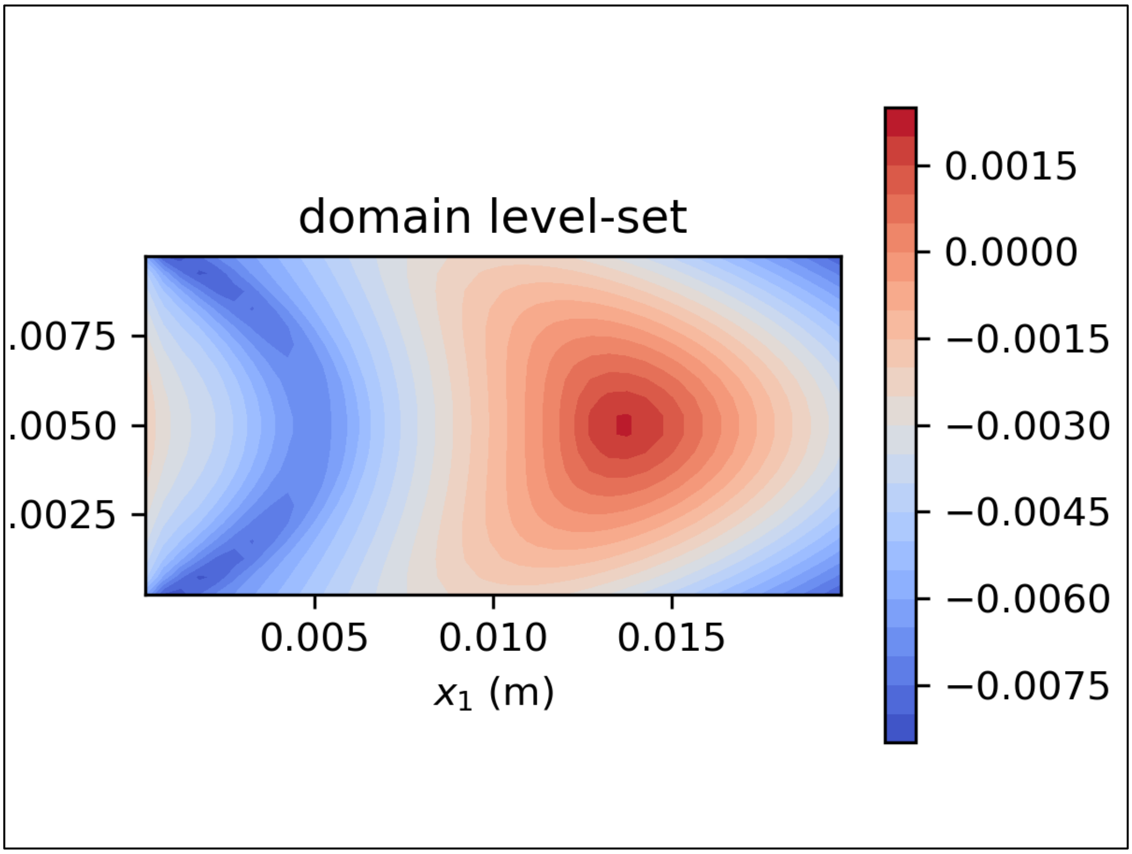
Initial level-set as required, 60 cells across domain height



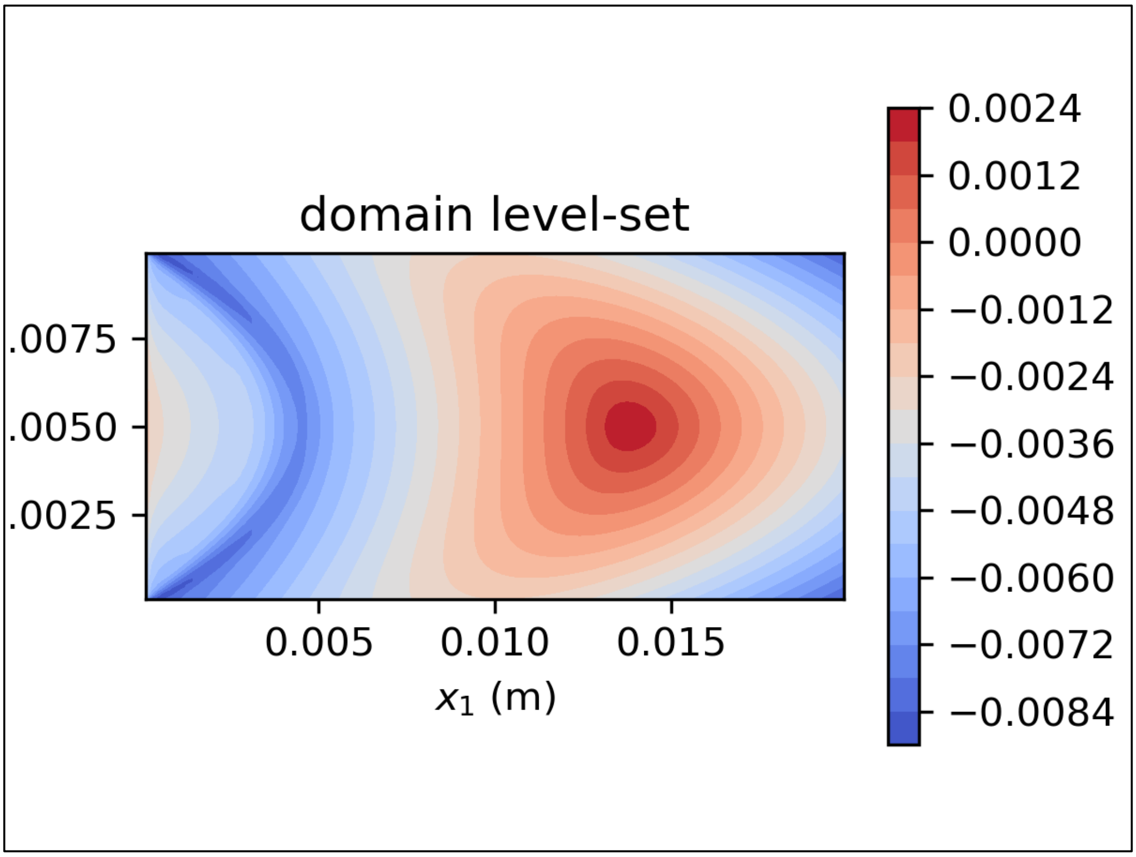
The mesh resolution mainly affects the maximum resolvable level-set despite smoothness in contour. The finer mesh case has higher droplet-center level-set, which is closer to reference (radius of the droplet).

(b)

Advected level-set field, 20 cells across domain height



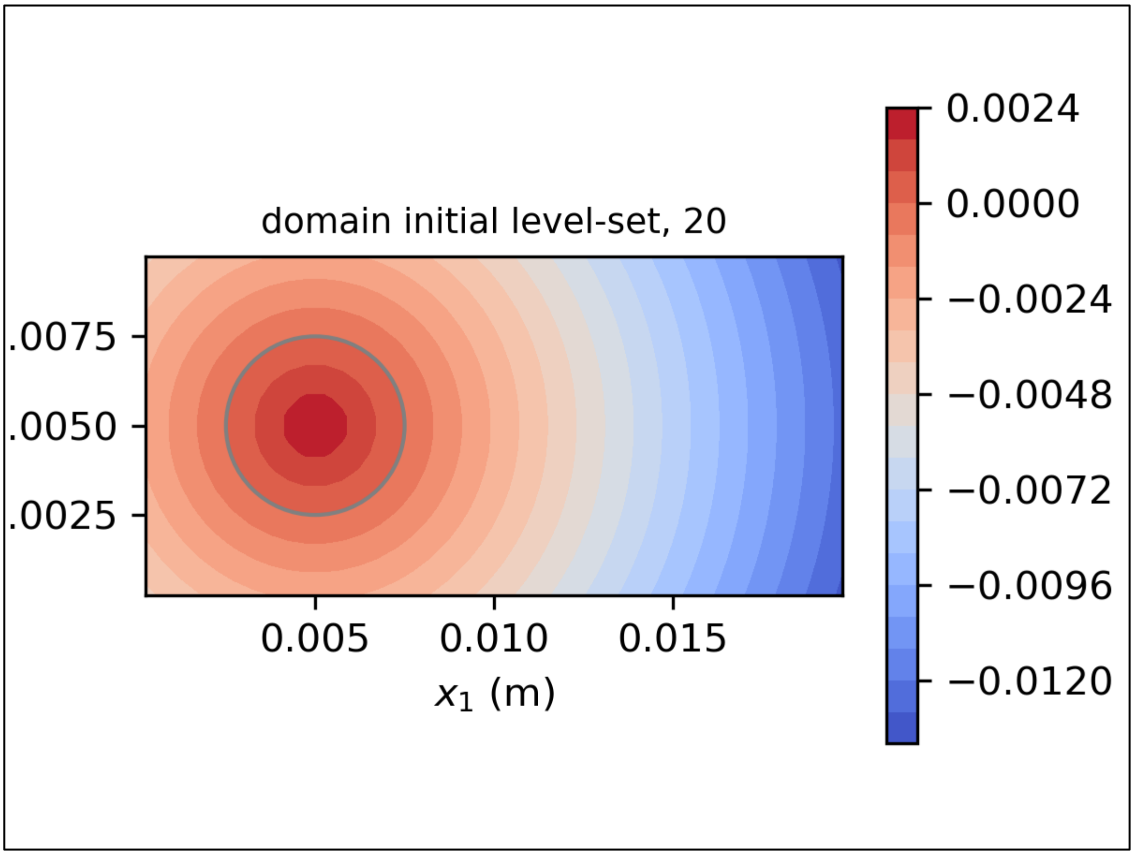
Advected level-set field,60 cells across domain height



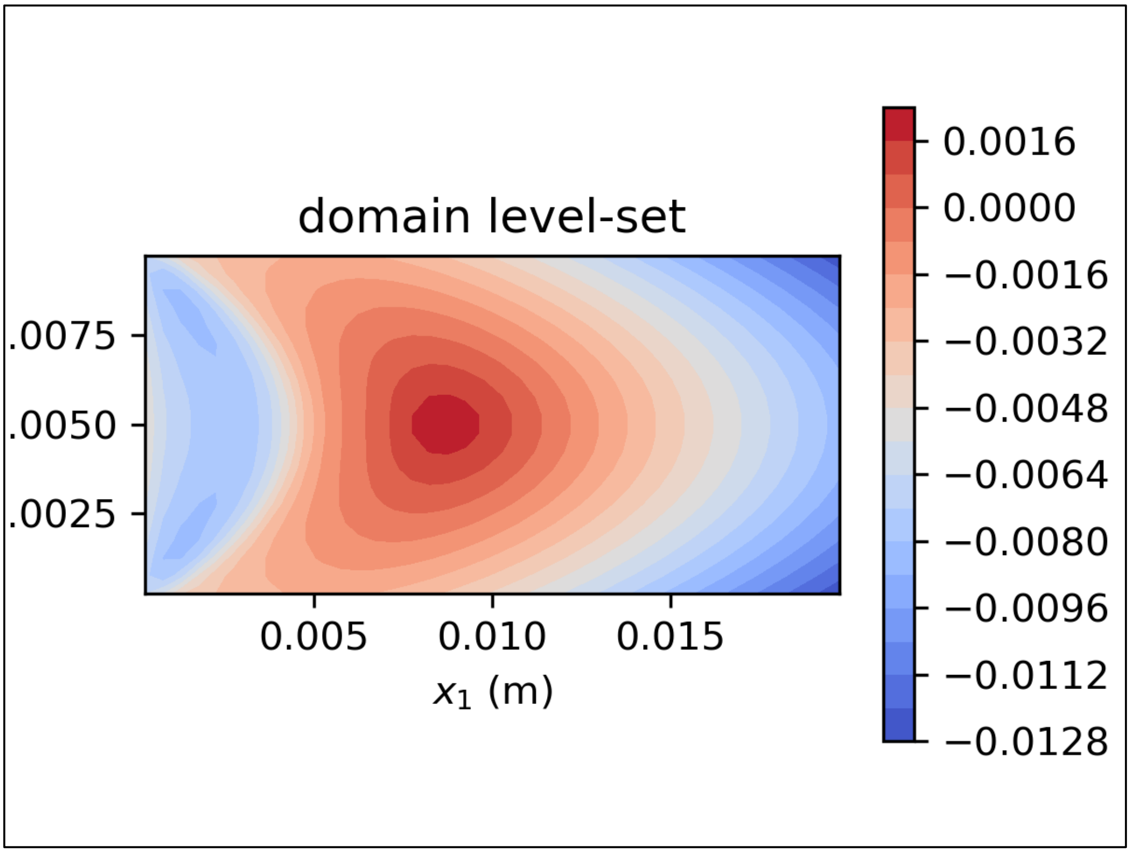
Due to level-set re-distancing is not enabled in the scope of this problem, the level-set field is advected along with the parabolic velocity profile across the channel. The advected droplet was elongated on the x_1 direction and the surrounding level-set field becomes bell-shaped. The distortion is more obvious at the near-wall region.

(c)

Initial level-set, initialized at $(0.005, 0.005)$, 20 cells across domain height



Advected level-set, initialized at $(0.005, 0.005)$, 20 cells across domain height

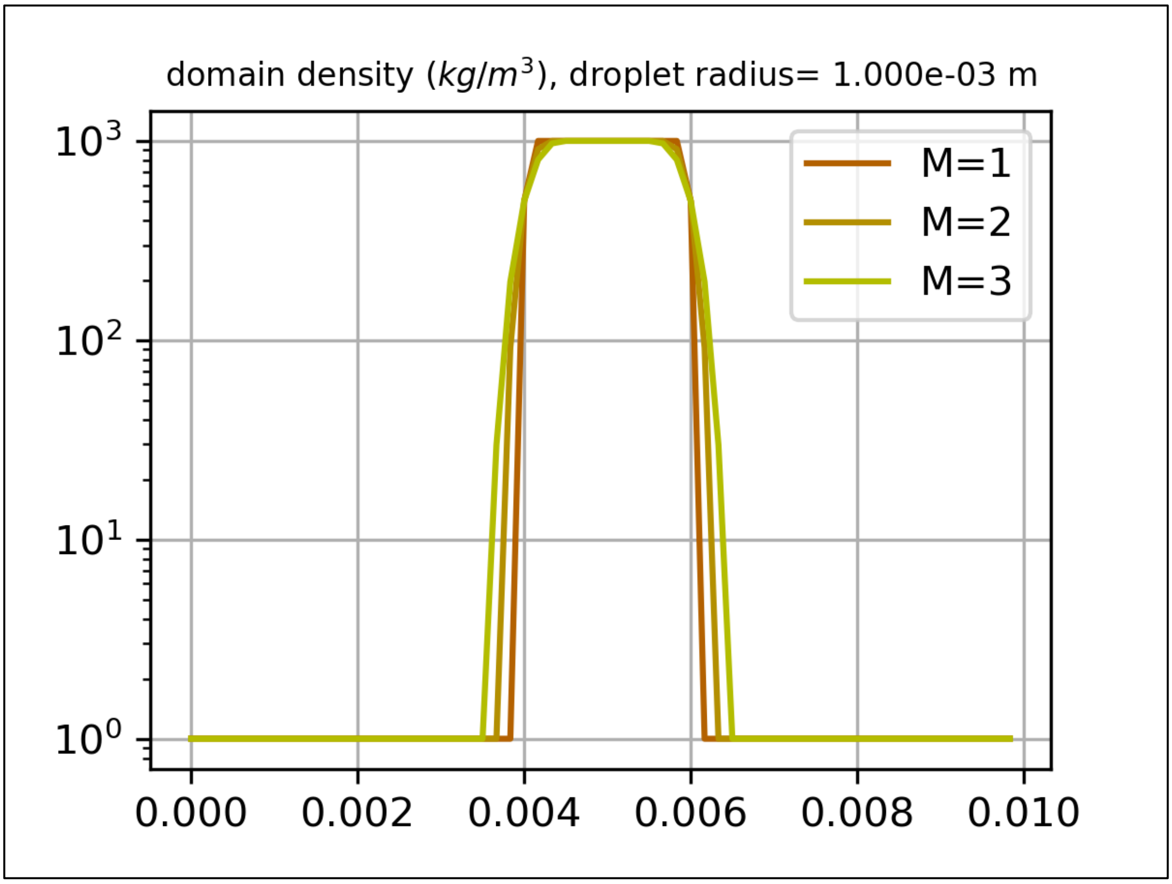


Compare results from different initial drop position, the (c) option is better since the less distortion of level-set at the wake region of the droplet.

(code is attached)

#5.3

The density distribution corresponding to $M = 1,2,3$ is plotted as below:



Mass of droplet corresponding to listed options:

	Option 1	Option 2	Option 3
$M = 1$	1.833	1.833	2.00
$M = 2$	1.833	1.803	2.00
$M = 3$	1.833	1.759	2.00

(Reference mass of the 1-D droplet $2.00\ kg/m^2$)

From observation, option 3 seems to be the best option to estimate mass of droplet since the loss of mass across smoothed level-set is insignificant compared to option 1 and 2. The option 2 is the most undesired option since the mass loss increases as M increases.
(code is attached)

Code for #5.2 (a), (b), (c)

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



#problem constants
nu=1e-6
mu=1e-3
rho=1e+3
dt=0.00001
gradP=-2.4
n_iter=0
'''
node generation section
'''
#domain length

Lx1=0.02
Lx2=0.01

r_dpl=0.25*Lx2
#number of cells on each direction
Nx1=40
Nx2=20

cell_vol=(Lx1/Nx1)*(Lx2/Nx2)

#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

def lvlset_init(x,y):
	def ls_dpl(x,y):
		return-1*(math.sqrt((x-0.005)**2+(y-0.005)**2)-r_dpl)
	return ls_dpl(x,y)

def D_x_p_n(i,j):
	return cell_cent_phin[i+1,j]-cell_cent_phin[i,j]

def D_x_m_n(i,j):
	return cell_cent_phin[i,j]-cell_cent_phin[i-1,j]

def D_y_p_n(i,j):
	return cell_cent_phin[i,j+1]-cell_cent_phin[i,j]

def D_y_m_n(i,j):
	return cell_cent_phin[i,j]-cell_cent_phin[i-1,j]

def D_x_p_s(i,j):
	return cell_cent_phis[i+1,j]-cell_cent_phis[i,j]

def D_x_m_s(i,j):
	return cell_cent_phis[i,j]-cell_cent_phis[i-1,j]

def D_y_p_s(i,j):
	return cell_cent_phis[i,j+1]-cell_cent_phis[i,j]

def D_y_m_s(i,j):
```

```
        return cell_cent_phis[i,j]-cell_cent_phis[i-1,j]

def L_phi_n(i,j):
        def phi_xh_n(i,j):
                if 0.5*(cell_S_x_unn[i+1,j]+cell_S_x_unn[i,j])>0:
                        return cell_cent_phin[i,j]+0.5*min(D_x_p_n(i,j), D_x_m_n(i,j))
                elif 0.5*(cell_S_x_unn[i+1,j]+cell_S_x_unn[i,j])<0:
                        return cell_cent_phin[i+1,j]-0.5*min(D_x_p_n(i+1,j), D_x_p_n(i+1,j))

        def phi_hx_n(i,j):
                if 0.5*(cell_S_x_unn[i,j]+cell_S_x_unn[i-1,j])<0:
                        return cell_cent_phin[i,j]-0.5*min(D_x_p_n(i,j), D_x_m_n(i,j))
                elif 0.5*(cell_S_x_unn[i,j]+cell_S_x_unn[i-1,j])>0:
                        return cell_cent_phin[i-1,j]+0.5*min(D_x_p_n(i-1,j), D_x_m_n(i-1,j))
        def phi_yh_n(i,j):
                if 0.5*(cell_S_y_vnn[i,j+1]+cell_S_y_vnn[i,j])>0:
                        return cell_cent_phin[i,j]+0.5*min(D_y_p_n(i.j), D_y_m_n(i,j))
                elif 0.5*(cell_S_y_vnn[i,j+1]+cell_S_y_vnn[i,j])<0:
                        return cell_cent_phin[i,j+1]+0.5*min(D_y_p_n(i,j+1), D_y_m_n(i,j+1))
        def phi_hy_n(i,j):
                if 0.5*(cell_S_y_vnn[i,j]+cell_S_y_vnn[i,j-1])<0:
                        return cell_cent_phin[i,j]-0.5*min(D_y_p_n(i,j), D_y_m_n(i,j))
                elif 0.5*(cell_S_y_vnn[i,j]+cell_S_y_vnn[i,j-1])>0:
                        return cell_cent_phin[i,j-1]+0.5*min(D_y_p_n(i,j-1), D_y_m_n(i,j-1))

        #return-1*(cell_S_x_unn[i,j])*(phi_xh_n(i,j)-phi_hx_n(i,j))/h-
(cell_S_y_vnn[i,j])*(phi_yh_n(i,j)-phi_hy_n(i,j))/h
        return-1*(cell_S_x_unn[i,j])*(phi_xh_n(i,j)-phi_hx_n(i,j))/h

def L_phi_s(i,j):
        def phi_xh_s(i,j):
                if 0.5*(cell_S_x_unn[i+1,j]+cell_S_x_unn[i,j])>0:
                        return cell_cent_phis[i,j]+0.5*min(D_x_p_s(i,j), D_x_m_s(i,j))
                elif 0.5*(cell_S_x_unn[i+1,j]+cell_S_x_unn[i,j])<0:
                        return cell_cent_phis[i+1,j]-0.5*min(D_x_p_s(i+1,j), D_x_p_s(i+1,j))

        def phi_hx_s(i,j):
                if 0.5*(cell_S_x_unn[i,j]+cell_S_x_unn[i-1,j])<0:
                        return cell_cent_phis[i,j]-0.5*min(D_x_p_s(i,j), D_x_m_s(i,j))
                elif 0.5*(cell_S_x_unn[i,j]+cell_S_x_unn[i-1,j])>0:
                        return cell_cent_phis[i-1,j]+0.5*min(D_x_p_s(i-1,j), D_x_m_s(i-1,j))
        def phi_yh_s(i,j):
```

```
            if 0.5*(cell_S_y_vnn[i,j+1]+cell_S_y_vnn[i,j])>0:
                return cell_cent_phis[i,j]+0.5*min(D_y_p_s(i.j), D_y_m_s(i,j))
            elif 0.5*(cell_S_y_vnn[i,j+1]+cell_S_y_vnn[i,j])<0:
                return cell_cent_phis[i,j+1]+0.5*min(D_y_p_s(i,j+1), D_y_m_s(i,j+1))
        def phi_hy_s(i,j):
            if 0.5*(cell_S_y_vnn[i,j]+cell_S_y_vnn[i,j-1])<0:
                return cell_cent_phis[i,j]-0.5*min(D_y_p_s(i,j), D_y_m_s(i,j))
            elif 0.5*(cell_S_y_vnn[i,j]+cell_S_y_vnn[i,j-1])>0:
                return cell_cent_phis[i,j-1]+0.5*min(D_y_p_s(i,j-1), D_y_m_s(i,j-1))
        #return-1*(cell_S_x_unn[i,j])*(phi_xh_s(i,j)-phi_hx_s(i,j))/h-
(cell_S_y_vnn[i,j])*(phi_yh_s(i,j)-phi_hy_s(i,j))/h
        return-1*(cell_S_x_unn[i,j])*(phi_xh_s(i,j)-phi_hx_s(i,j))/h


epstot=100.0
p_iter=0
#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_cent_pn=np.zeros([Nx1+2,Nx2+2])
cell_cent_pnn=np.zeros([Nx1+2,Nx2+2])

cell_cent_phin=np.zeros([Nx1+2,Nx2+2])
cell_cent_phis=np.zeros([Nx1+2,Nx2+2])
cell_cent_phinn=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]))
            #lvlset init
            cell_cent_phin[i,j]=lvlset_init(cell_cent_x[i,j], cell_cent_y[i,j])
            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]=ref_vel_prof(cell_cent_y[i,j])
            #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]

bub=plt.Circle((0.005, 0.005), r_dpl, color='grey', fill=False)
fig, ax=plt.subplots()
plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1],
cell_cent_phin[1:Nx1+1, 1:Nx2+1], 20, cmap='coolwarm')
plt.colorbar()
ax.add_artist(bub)
plt.xlabel('$x_1$ (m)')
plt.ylabel('$x_2$ (m)')
plt.title('domain initial level-set, '+str(Nx2), fontsize=9)
plt.gca().set_aspect('equal')
plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_ls_init.png')
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[1]>=1e-5:
while n_iter<=13000:
    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)-gradP/rho)
    epstot=100.0
    while epstot>1e-4:
        epstot=0.0


        for j in range(2, Nx2):
            U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_us[2,j]-
cell_S_x_unn[1,j]+cell_S_y_vs[1,j+1]-cell_S_y_vs[1,j])
            cell_cent_pnn[1,j]=(cell_vol*U_s-
(cell_cent_pn[2,j]+cell_cent_pn[1,j+1]+cell_cent_pn[1,j-1]))/(-3)
            epstot+=(cell_cent_pnn[1,j]-cell_cent_pn[1,j])**2
            cell_cent_pn[1,j]=cell_cent_pnn[1,j]


        for i in range(2, Nx1):
            for j in range(1,Nx2+1):
                U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_us[i+1,j]-
cell_S_x_us[i,j]+cell_S_y_vs[i,j+1]-cell_S_y_vs[i,j])
                cell_cent_pnn[i,j]=(cell_vol*U_s-(cell_cent_pn[i+1,j]+cell_cent_pn[i-
1,j]+cell_cent_pn[i,j+1]+cell_cent_pn[i,j-1]))/(-4)
                #print('{:4.4e}, {:4.4e}'.format(cell_cent_pnn[i,j], cell_cent_pn[i,j]))
                epstot+=(cell_cent_pnn[i,j]-cell_cent_pn[i,j])**2
                cell_cent_pn[i,j]=cell_cent_pnn[i,j]
        for j in range(2, Nx2):
            U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_unn[-1,j]-cell_S_x_us[-2,j]+cell_S_y_vs[-
2,j+1]-cell_S_y_vs[-2,j])
            cell_cent_pnn[-2,j]=(cell_vol*U_s-(cell_cent_pn[-3,j]+cell_cent_pn[-
2,j+1]+cell_cent_pn[-2,j-1]))/(-3)
            epstot+=(cell_cent_pnn[-2,j]-cell_cent_pn[-2,j])**2
            cell_cent_pn[-2,j]=cell_cent_pnn[-2,j]
        #coroner update
        U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_us[2,1]-cell_S_x_unn[1,1]+cell_S_y_vs[1,2]-
cell_S_y_vnn[1,1])
        cell_cent_pnn[1,1]=(cell_vol*U_s-(cell_cent_pn[2,1]+cell_cent_pn[1,2]))/(-2)
        epstot+=(cell_cent_pnn[1,1]-cell_cent_pn[1,1])**2
        cell_cent_pn[1,1]=cell_cent_pnn[1,1]

        U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_us[2,-2]-cell_S_x_unn[1,-2]+cell_S_y_vnn[1,-
2]-cell_S_y_vnn[1,-3])
        cell_cent_pnn[1,-2]=(cell_vol*U_s-(cell_cent_pn[2,-2]+cell_cent_pn[1,-3]))/(-2)
```

```
            epstot+=(cell_cent_pnn[1,-2]-cell_cent_pn[1,-2])**2
            cell_cent_pn[1,-2]=cell_cent_pnn[1,-2]

            U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_unn[-2+1,1]-cell_S_x_us[-2,1]+cell_S_y_vs[-
2,2]-cell_S_y_vnn[-2,1])
            cell_cent_pnn[-2,1]=(cell_vol*U_s-(cell_cent_pn[-2-1,1]+cell_cent_pn[-2,2]))/(-2)
            epstot+=(cell_cent_pnn[-2,1]-cell_cent_pn[-2,1])**2
            cell_cent_pn[-2,1]=cell_cent_pnn[-2,1]

            U_s=(rho/(dt*(Lx1/Nx1)))*(cell_S_x_unn[-2+1,-2]-cell_S_x_us[-2,-
2]+cell_S_y_vnn[-2,-2+1]-cell_S_y_vs[-2,-2])
            cell_cent_pnn[-2,-2]=(cell_vol*U_s-(cell_cent_pn[-2-1,-2]+cell_cent_pn[-2,-2-
1]))/(-2)
            epstot+=(cell_cent_pnn[-2,-2]-cell_cent_pn[-2,-2])**2
            cell_cent_pn[-2,-2]=cell_cent_pnn[-2,-2]



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

            if p_iter%500==0:
                plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1],
cell_cent_pnn[1:Nx1+1, 1:Nx2+1], 20, cmap='jet')
                plt.colorbar()
                plt.xlabel('$x_1$ (m)')
                plt.ylabel('$x_2$ (m)')
                plt.title('domain $p^{n+1}$ contour ($Pa$)')
                plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_p_contour.png')
                plt.gca().set_aspect('equal')
                plt.show()
                print('eps_tot= {:5.4e}'.format(epstot))
            p_iter+=1
        #print('eps_tot= {:5.4e}'.format(epstot))
        #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)*(cell_cent_pnn[i,j]-
cell_cent_pnn[i-1,j])
```

```
    #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_x_unn[i,1]
        cell_S_x_unn[i,-1]=cell_S_x_unn[i,-2]



    for j in range(1, Nx2+1):
        for i in range(1, Nx1+1):
            cell_cent_phis[i,j]=cell_cent_phin[i,j]+dt*L_phi_n(i, j)
            cell_cent_phinn[i,j]=cell_cent_phin[i,j]+0.5*dt*(L_phi_n(i,j)+L_phi_s(i,j))
            cell_cent_phin[i,j]=cell_cent_phinn[i,j]
            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]

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

        cell_cent_phin[i,0]=-cell_cent_phin[i,1]
        cell_cent_phin[i,-1]=-cell_cent_phin[i,-2]

    sq_sum_error=0

    for i in range(1,Nx2+1):
        sq_sum_error+=(ref_S_u[i]-cell_S_x_un[int(0.5*Nx1),i])**2
    L_sq[1]=math.sqrt(sq_sum_error/(Nx2+1))
    if n_iter%500==0:
        print('iter= '+str(n_iter)+', L_sq= {:.4e}'.format(L_sq[0]))
        plt.plot(cell_S_x_un[int(0.5*Nx1),1:Nx2+1],cell_S_x_coor_y[int(0.5*Nx1),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[int(0.5*Nx1),1:Nx2+1], color='red',
label='reference')
        plt.xlabel('$u_1$ ($m/s$)')
```

```
            plt.ylabel('$x_2$ (m)')
            plt.legend()
            plt.grid()
            plt.gca().set_aspect('equal')
            plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_v_profile.png')
            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.gca().set_aspect('equal')
            plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_v_contour.png')
            plt.show()



            plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1],
cell_cent_phin[1:Nx1+1, 1:Nx2+1], 20, cmap='coolwarm')
            plt.colorbar()
            plt.xlabel('$x_1$ (m)')
            plt.ylabel('$x_2$ (m)')
            plt.title('domain level-set')
            plt.gca().set_aspect('equal')
            plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_ls.png')
            plt.show()

            print('{:10d},     {:5.7e}'.format(n_iter, L_sq[1]))
      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[int(0.5*Nx1),1:Nx2+1],cell_S_x_coor_y[int(0.5*Nx1),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[int(0.5*Nx1),1:Nx2+1], color='red',
label='reference')
plt.xlabel('$u_1$ ($m/s$)')
plt.ylabel('$x_2$ (m)')
plt.legend()
plt.gca().set_aspect('equal')
```

```
plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_v_profile.png')
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 ($Pa$)')
plt.gca().set_aspect('equal')
plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_v_contour.png')
plt.show()


plt.contourf(cell_cent_x[1:Nx1+1, 1:Nx2+1], cell_cent_y[1:Nx1+1, 1:Nx2+1],
cell_cent_phin[1:Nx1+1, 1:Nx2+1], 20, cmap='coolwarm')
plt.colorbar()
plt.xlabel('$x_1$ (m)')
plt.ylabel('$x_2$ (m)')
plt.title('domain level-set')
plt.gca().set_aspect('equal')
plt.savefig('hw5_2_'+str(Nx2)+'_init_ref_ls.png')
plt.show()
```

Code for #5.3

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


L=0.01
N_n=60
r_dpl=0.001
x_dpl=0.005
global h
h=L/N_n

rho_l=1000
rho_g=1

m_1=0
m_2=0
m_3=0


cell_x=np.zeros(N_n)
cell_phi=np.zeros(N_n)
cell_rho=np.zeros(N_n)
def m_o1():
	return rho_l*h

def m_o2(rho):
	return rho*h

def m_o3(rho):
	w=(rho-rho_g)/(rho_l-rho_g)
	return rho_l*w*h
def lvlset(x):
	if x > (x_dpl-r_dpl) and x < (x_dpl+r_dpl):
		return min(abs(x-(x_dpl-r_dpl)), abs(x-(x_dpl+r_dpl)))
	else:
		return-1*min(abs(x-(x_dpl-r_dpl)), abs(x-(x_dpl+r_dpl)))
def hvsd(phi, M):
	if phi<-1*M*h:
```

```
            return 0
        elif abs(phi)<= M*h:
            return 0.5*(1+phi/(M*h)+math.sin(math.pi*phi/(M*h))/math.pi)
        elif phi> M*h:
            return 1
for M in range(1,4):
    m_1=0
    m_2=0
    m_3=0
    for i in range(0, len(cell_x)):
        cell_x[i]=0.0+i*h
        cell_phi[i]=lvlset(cell_x[i])
        cell_rho[i]=rho_l*hvsd(cell_phi[i], M)+rho_g*(1-hvsd(cell_phi[i], M))
        m_3+=m_o3(cell_rho[i])
        if cell_phi[i]>0:

            m_1+=m_o1()
            m_2+=m_o2(cell_rho[i])
    print('{:4.3e}, {:4.3e}, {:4.3e}, {:1d}'.format(m_1, m_2, m_3, M))
    #plt.plot(cell_x, cell_phi, color='red', label='domain level-set')
    plt.plot(cell_x, cell_rho, color=(0.7, 0.18*M+0.2, 0), label='M={:1d}'.format(M))
plt.yscale('log')
plt.legend()
plt.title('domain density $(kg/m^3)$, droplet radius= {:4.3e} m'.format(r_dpl), fontsize=8)
plt.grid()
plt.savefig('hw5_3_density_distr.png')
plt.show()
#plt.plot(cell_x, cell_rho, color='navy', label='domain density $(kg/m^3)$')
#plt.show()
```