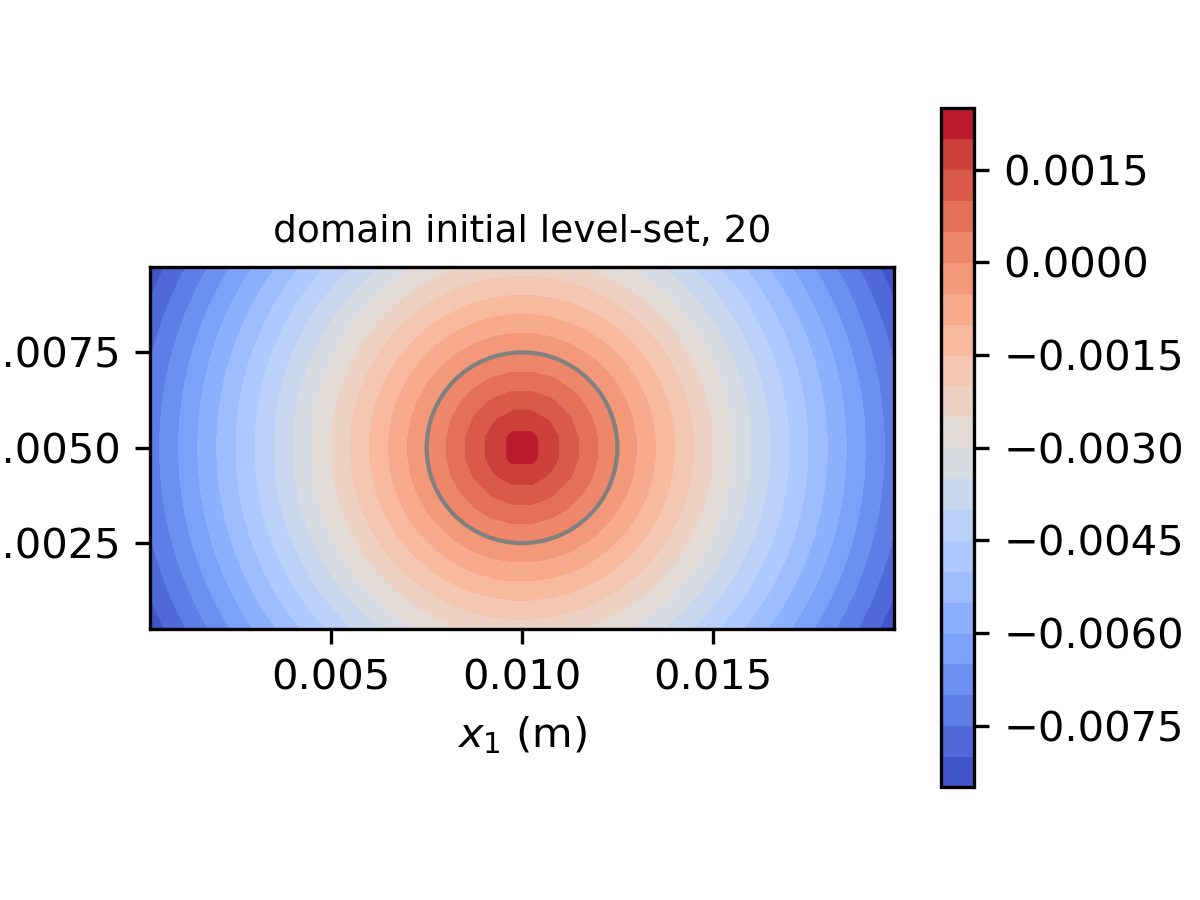
一張含有 文字 的圖片

自動產生的描述

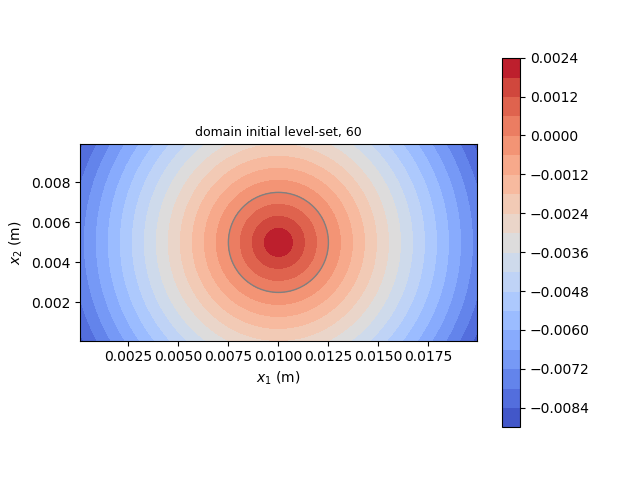
**#5.2**

(a)

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



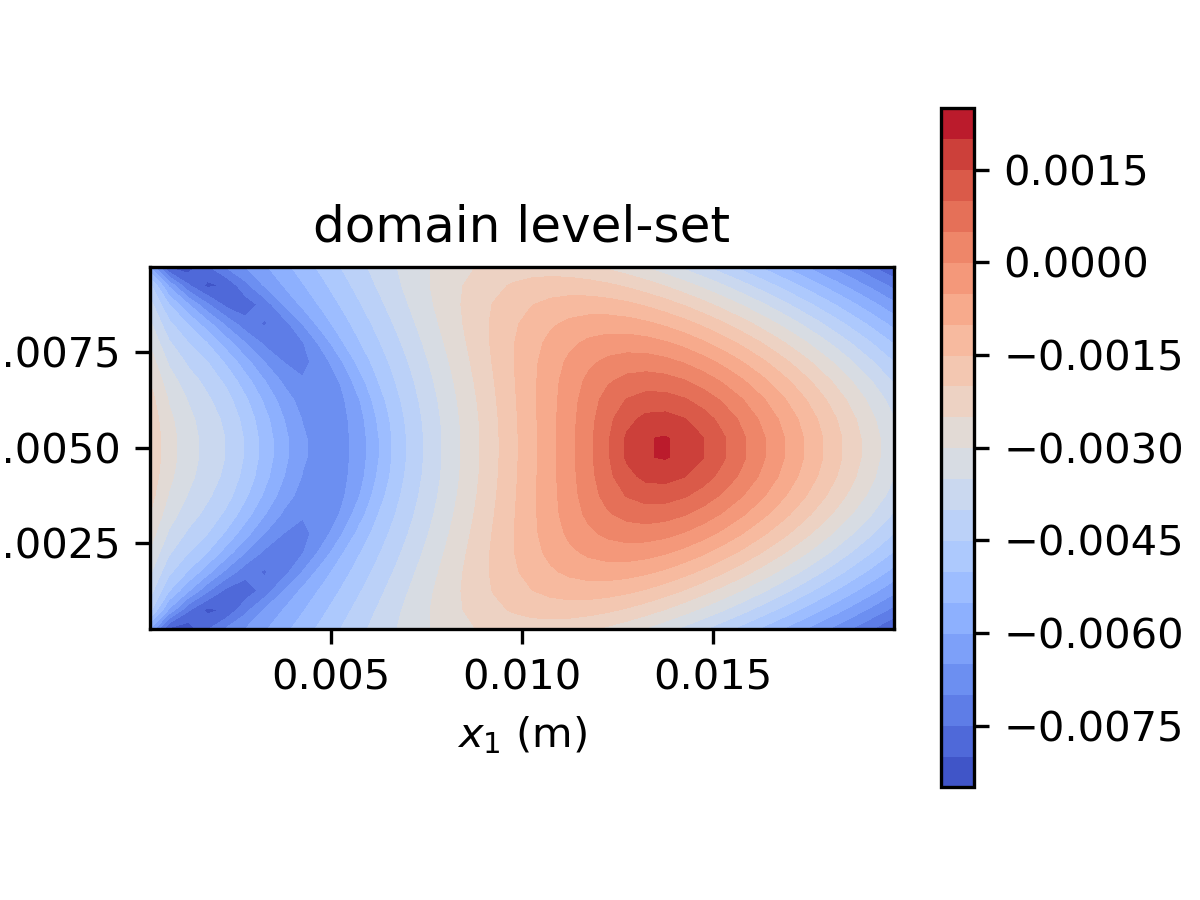
Initial level-set as required, 60 cells across domain height



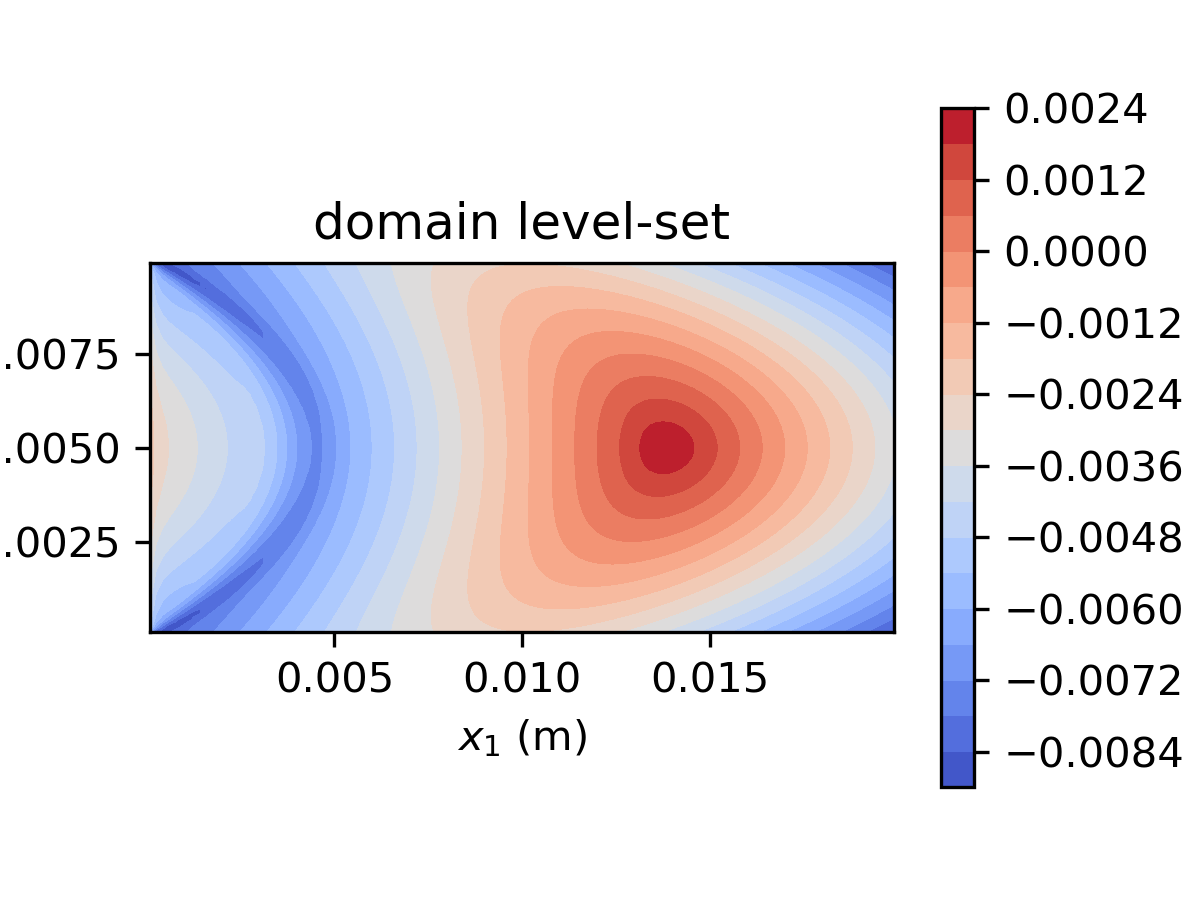
The mesh resolution mainly affects the maximum resolvable level-set despite smoothness in contour. The finer mesh case has higher droplet-center level-set, which is closer to reference (radius of the droplet).

(b)

Advected level-set field, 20 cells across domain height



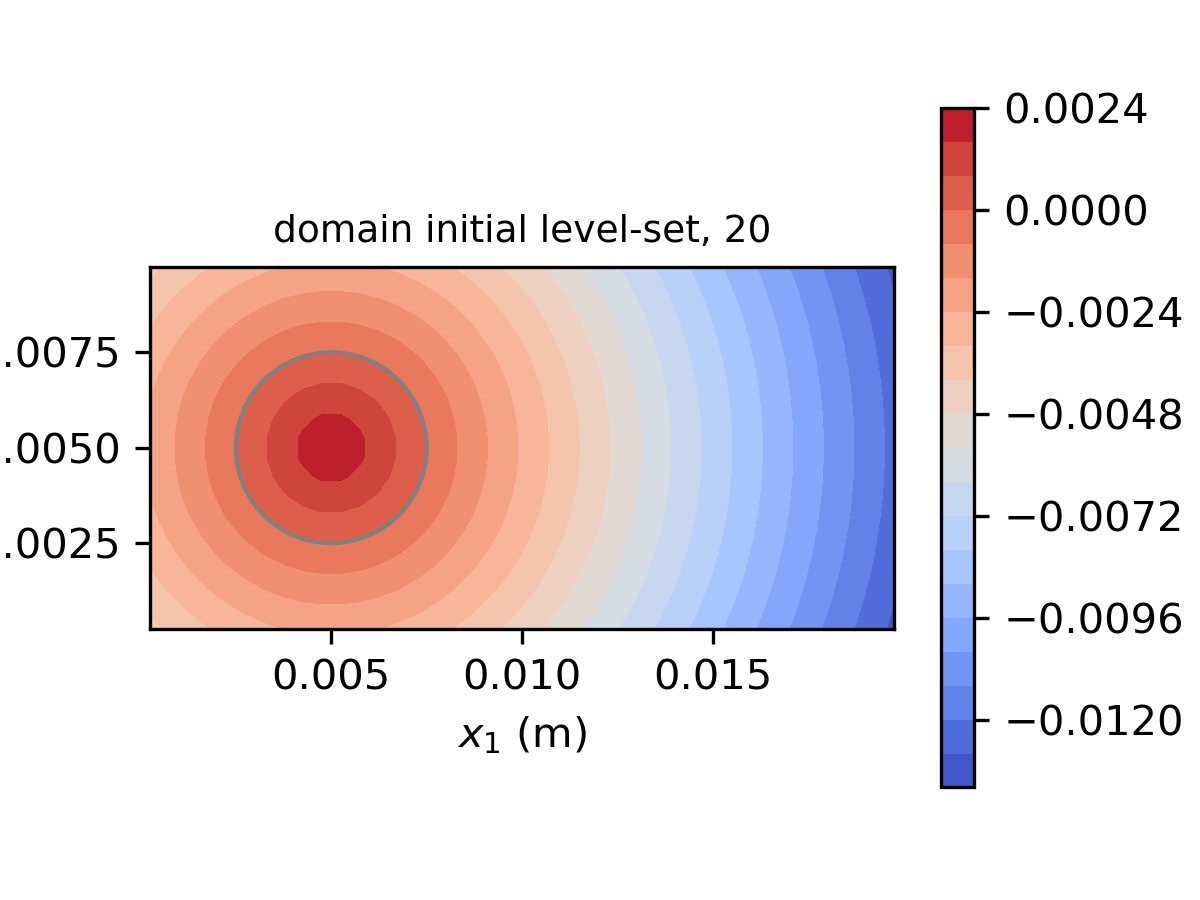
Advected level-set field,60 cells across domain height



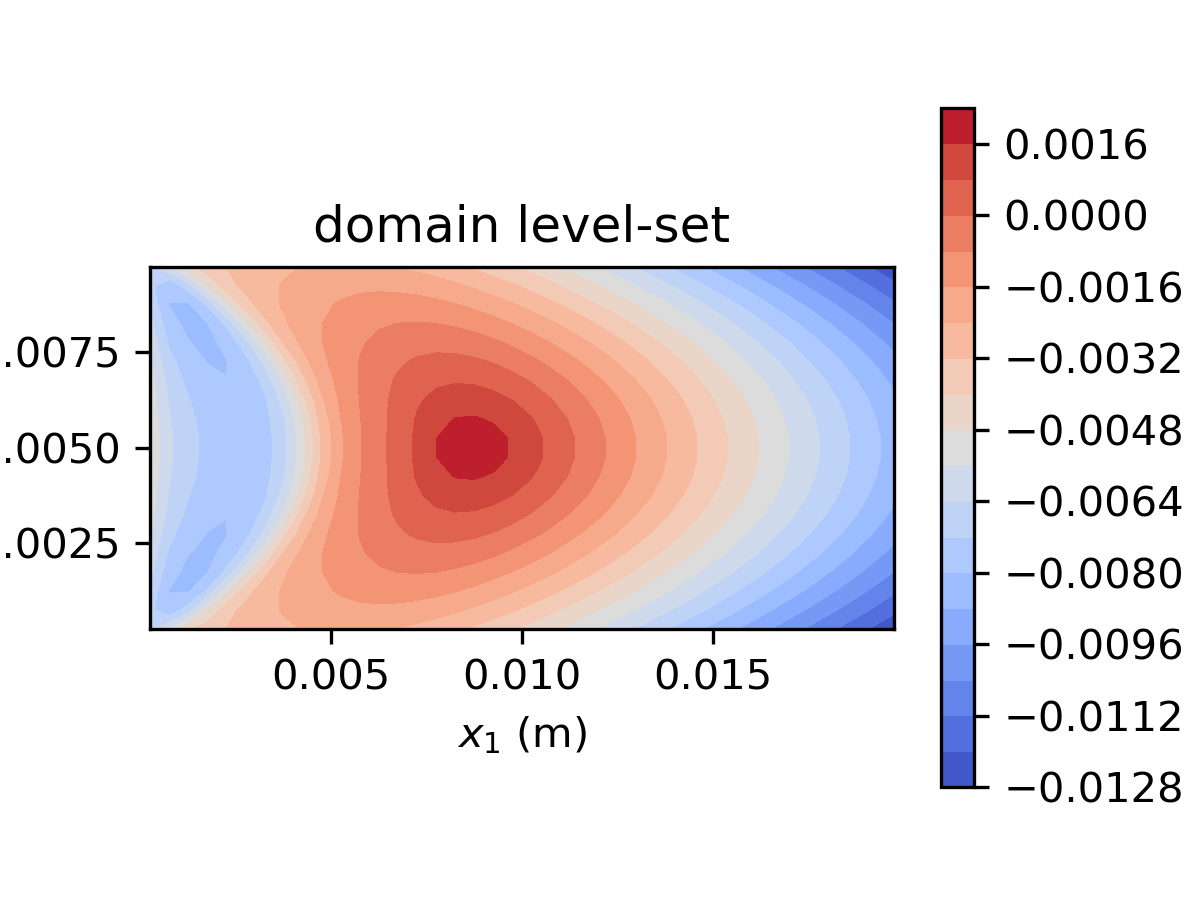
Due to level-set re-distancing is not enabled in the scope of this problem, the level-set field is advected along with the parabolic velocity profile across the channel. The advected droplet was elongated on the direction and the surrounding level-set field becomes bell-shaped. The distortion is more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

(c)

Initial level-set, initialized at , 20 cells across domain height



Advected level-set, initialized at , 20 cells across domain height

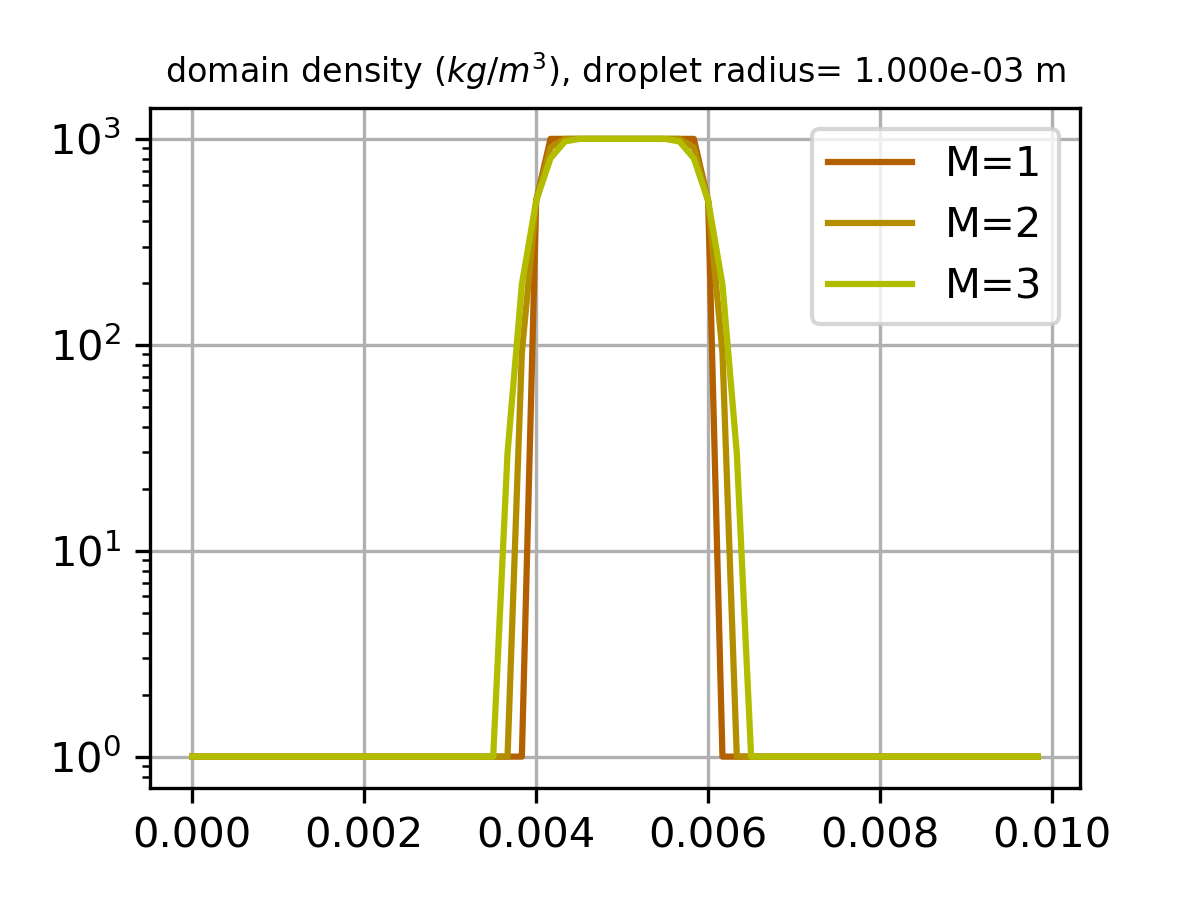


Compare results from different initial drop position, the (c) option is better since the less distortion of level-set at the wake region of the droplet.

(code is attached)

#5.3

The density distribution corresponding to is plotted as below:



Mass of droplet corresponding to listed options:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Option 1 | Option 2 | Option 3 |
|  | 1.833 | 1.833 | 2.00 |
|  | 1.833 | 1.803 | 2.00 |
|  | 1.833 | 1.759 | 2.00 |

(Reference mass of the 1-D droplet )

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()