Simulating Two-Phase Incompressible Flow using Level Set Approach

MAE510 - Computational Moving Interfaces - Course Project
Instructor - Prof. David Salac

University at Buffalo

Pranav Ladkat (5009-4671)

1 Abstract

A level set approach for computing solutions to incompressible two-phase flow is presented. The interface between two fluids is described as zero level set of a smooth function. The incompressible flow field is solved using second order Projection Method by Kim and Moin[1]. A Continuum Surface Force method[2] is used for modeling Surface Tension. The case of low density fluid bubble rising in denser fluid is considered.

Keywords: Level Sets, Two-Phase Incompressible Flow, Projection Method.

2 Introduction

3 Description of Algorithm

3.1 Governing Equations

In the present study, we consider the motion of low density fluid rising in denser fluid. As the velocity field developed is much less than the velocity of sound, the incompressible form of Navier-Stokes equation is modeled with the incompressibility constraint. The equations of motion are:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}.\nabla)\,\mathbf{u} = \frac{1}{\rho}\left(-\nabla p + \nabla.\left(2\mu\mathcal{D}\right) - \sigma k \delta(d)\mathbf{n}\right) + \mathcal{F} \tag{1}$$

$$\nabla.\mathbf{u} = 0 \tag{2}$$

where $\mathbf{u} = (u, v)$ is the fluid velocity, ρ is the density, μ is viscosity, $\mathcal{D} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$ is the rate of deformation tensor and $\mathcal{F}$ is the body force due to gravity. The surface tension term is considered to be a force concentrated on the interface. The derivation of this form can be found in [2]. Surface tension is denoted by σ, curvature of the front by k, Dirac delta function by δ, and the normal to interface by $\mathbf{n}$. Boundary of the domain is assumed as solid wall and the no slip boundary condition is applied.

The non-dimensional form of (1) is

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial t} &= -\left(\mathbf{u}.\nabla\right)\mathbf{u} + \mathbf{g}_u \\ &+ \frac{1}{\rho}\left(-\nabla p + \frac{1}{Re}\nabla.\left(2\mu\mathcal{D}\right) - \frac{1}{B}k\delta(d)\mathbf{n}\right)\end{aligned} \tag{3}$$

The key parameters are ρ_b/ρ_c and μ_b/μ_c, which are dimensionless density and viscosity inside the bubble respectively. ρ_b and μ_b represents density and viscosity of fluid inside bubble respectively, and ρ_c and μ_c denotes the same for outer continuum fluid; The Reynolds Number is $Re = (2R)^{3/2}\sqrt{\mathbf{g}}\rho_c/\mu_c$; and the Bond Number, $B = 4\rho_c\mathbf{g}R^2/\sigma$. Where characteristic length $L = R$ and characteristic velocity $U = \sqrt{\mathbf{g}R}$ is used. R denotes the initial radius of bubble. The dimensionless density and viscosity outside the bubble are equal to 1. $\mathbf{g}_u$ denotes the unit gravitational force.

3.2 Level Set Description

In the present study, the interface between gas and liquid is tracked with the level set function ϕ and the interface Γ, is the zero level set of ϕ:

$$\Gamma = \{\mathbf{x}|\phi(\mathbf{x}, t) = 0\}$$

The level set function ϕ is positive in the liquid and negative in gas. Hence we have

$$\phi(x,t) = \begin{cases} > 0 & \text{if } x \in \text{the liquid} \\ = 0 & \text{if } x \in \Gamma \\ < 0 & \text{if } x \in \text{the gas} \end{cases} \tag{4}$$

The following equation will move the zero level set of ϕ exactly as the actual bubble interface moves.

$$\frac{\partial \phi}{\partial t} + \mathbf{u}.\nabla\phi = 0 \tag{5}$$

Since ϕ is a smooth function, unlike ρ and μ, the above equation can be easily solved numerically than solving for ρ and μ separately which are discontinuous at the interface. Hence, ρ and μ in terms of ϕ can be written as

$$\rho(\phi) = H(\phi) + \frac{\rho_b}{\rho_c}(1 - H(\phi)) \tag{6}$$

$$\mu(\phi) = H(\phi) + \frac{\mu_b}{\mu_c}(1 - H(\phi)) \tag{7}$$

Where, $H(\phi)$ is a smoothed Heaviside function to avoid numerical instabilities. The smoothed Heaviside function is defined as

$$H(\phi) = \begin{cases} 0 & \text{if } \phi < -\epsilon \\ \frac{1}{2}\left[1 + \frac{\phi}{\epsilon} + \frac{1}{\pi} sin\left(\frac{\pi\phi}{\epsilon}\right)\right] & \text{if } |\phi| \leq \epsilon \\ 1 & \text{if } \phi > \epsilon \end{cases} \tag{8}$$

The parameter ϵ can be used to define the interface thickness and its value is set to $\alpha\Delta x$, where $\alpha = \frac{3}{2}$ is used in the presents study which ensures that $H(\phi)$ varies smoothly over three grid spacings.

The surface tension force represented by $\dfrac{1}{B}k\delta(d)\mathbf{n}$ in (3), can be cast in the level set formulation as pointed out in [2]

$$\frac{1}{B}k\delta(d)\mathbf{n} = \frac{1}{B}k(\phi)\delta(\phi)\nabla\phi \tag{9}$$

where the curvature is

$$k(\phi) = \nabla.\left(\frac{\nabla\phi}{|\nabla\phi|}\right) = \frac{\phi_{xx}\phi_y^2 + \phi_{yy}\phi_x^2 - 2\phi_{xy}\phi_x\phi_y}{(\phi_x^2 + \phi_y^2)^{3/2}} \tag{10}$$

If ϕ is maintained as signed distance function, as is done here, then we may numerically approximate $\delta(\phi)$ by a mollified delta function as

$$\delta(\phi) = \begin{cases} \frac{1}{2\epsilon}\left[1 + cos\left(\frac{\pi\phi}{\epsilon}\right)\right] & \text{if } |\phi| < \epsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

where ϵ is same as that used in Heaviside function (8).

3.3 Reinitialization

While equation (5) will move the level set $\phi = 0$ at the correct velocity, ϕ will not longer be a distance function and ϕ can become irregular after some period of time. Hence frequent reinitialization is necessary to keep ϕ a distance function. In order to do so, we must be able to solve the following problem/; given a level set function $\phi(x, t)$, reinitialize it so that it is a distance function for $|\phi| < \epsilon$ without changing its zero level set. This is achieved if we solve

$$\frac{\partial\phi}{\partial\tau} = sign(\phi_0)\left(1 - \sqrt{\phi_x^2 + \phi_y^2}\right) \tag{12}$$

$$\phi(x, 0) = \phi_0(x) \tag{13}$$

Where, τ is pseudo time. For numerical purposes it is useful to smooth the sign function. Here, smoothed sign function used is

$$sign(\phi) = \frac{\phi}{\sqrt{\phi^2 + |\nabla\phi| * h^2}} \tag{14}$$

The equation (12) is a hyperbolic equation with the characteristic velocities pointing outwards from the interface in the direction of the normal. This means that ϕ will be reinitialized to $|\nabla\phi| = 1$ near the interface first. Since we need ϕ to be distance function near interface, it is only necessary to solve equation (12)for few iterations.

3.4 Projection Method

Projection Method is a way of solving Incompressible Navier-Stokes equation which has been a most preferred choice for transient flow simulations. In its most basic form, the projection method requires the solution of advection-diffusion equations, which are then projected onto the space of divergence-free vector fields. The projection exploits the orthogonality of the pressure gradient with divergence-free velocity fields that is manifest in the Hodge decomposition. In the present study, pressure-free projection method by Kim and Moin [1] is used. In projection methods, the momentum equation is split in to two equations. The first equation predicts an intermediate velocity which is in general not divergence free, and in second step the incompressibility condition is enforced by solving an elliptic equation.

For the diffusion-convection step we solve for the intermediate velocity using

$$\frac{\partial\mathbf{u}}{\partial t} = -\left[(\mathbf{u}.\nabla)\,\mathbf{u}\right]^{n+1/2} + \frac{1}{\rho}\left(\frac{1}{2}\mathcal{L}^n - \mathcal{T}^n\right) + \mathbf{g}_u \tag{15}$$

Where, $\partial\mathbf{u}/\partial t$ represents temporal discretization as discussed in section (4.1), $\mathcal{L}$ represents second order finite difference approximation to $(1/Re)\nabla.(2\mu(\phi)\mathcal{D})$, $\mathcal{T}$ is a finite difference approximation to $(1/B)k(\phi)\delta(\phi)\nabla\phi$ and $\mathbf{g}_u$ represents unit gravitational force in downward direction. Solution of the equation (15) will result in the intermediate velocity field $\mathbf{u}^*$.

The incompressibility condition is enforced by solving the following elliptic equation

$$\frac{\nabla.\mathbf{u}^*}{\Delta t} = \nabla.\left(\frac{1}{\rho}(\nabla Q)\right) \tag{16}$$

where Q is the scalar field which behaves similar to the pressure. After computing Q, the velocities can be updated to time level $n+1$ using

$$\mathbf{n}^{n+1} = \mathbf{u}^* - \Delta t \left(\frac{1}{\rho} \nabla Q \right) \tag{17}$$

which will be divergence-free.

3.5 Algorithm Summary

The algorithm is summarized in the following steps:
Step 1. Initialize $\phi(x,t)$ to be signed normal distance to the front.

Step 2. Define variables $\rho(\phi)$ and $\mu(\phi)$ using equations (6) and (7), respectively.

Step 3. Solve for intermediate velocity field by equation (15), projection step by equation (16) and then update velocity to time level $n+1$ using equation (17).

Step 4. Advect the level set function using equation (5)

Step 5. Reinitialize level set function by solving equation (12) to steady state.

Step 6. We have now advanced one time step, repeat *Step 2* to *Step 5* till solution reaches final time.

4 Discretization

A collocated grid is used where velocity, pressure and level set functions are described at the grid points. For the uniform grid spacing, we have $\Delta x = \Delta y = h$. With h as mesh size, we define

$$\begin{aligned} \mathbf{x}_{i,j} &= (i * h, j * h) \\ \mathbf{u}_{i,j} &= \mathbf{u}(\mathbf{x}_{i,j}) \\ \phi_{i,j} &= \phi(\mathbf{x}_{i,j}) \\ i = 0...M&, j = 0...N \end{aligned}$$

The domain used in the present simulations is a square box with $Mh = Nh = 7R$. The arrangement of the discrete variables $\mathbf{u}$, p and ϕ on the grid is shown in figure (1). All variables are located at cell centroid.

4.1 Temporal Discretization

1. Adam Bashforth Time stepping for intermediate velocity U^*:
The time derivative in equation (15) are approximated using second order accurate Adams-Bashforth method to compute intermediate velocity

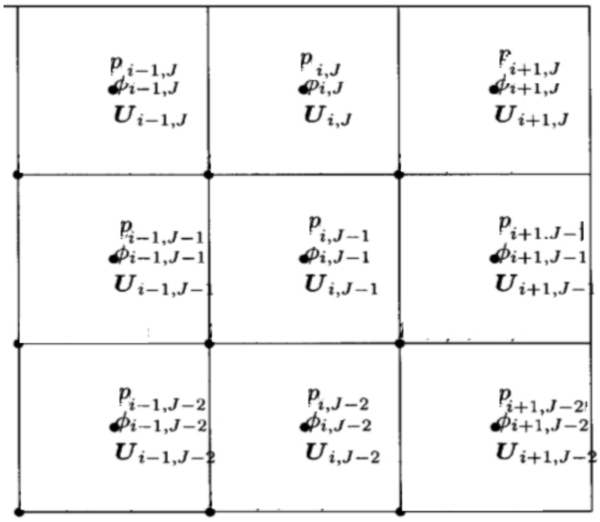

Figure 1: Collocated grid arrangement where discrete variables $\mathbf{u}$, p and ϕ are located

field. Hence the equation (15) is written as

$$\mathbf{u}^* = \mathbf{u}^n + \frac{\Delta t^n}{2} \left(\frac{\Delta t^n + 2\Delta t^{n-1}}{\Delta t^{n-1}} RHS^n - \frac{\Delta t^n}{\Delta t^{n-1}} RHS^{n-1} \right) \tag{18}$$

where Δt^n and Δt^{n-1} denotes the time step size at time level n and $n-1$ respectively. RHS^n is $\left[-[(\mathbf{u}.\nabla)\,\mathbf{u}]^{n+1/2} + \left(\frac{1}{2}\mathcal{L}^n - \mathcal{T}^n\right)/\rho + \mathbf{g}_u\right]^n$ and RHS^{n-1} is defined similarly.

2. Level Set update for ϕ^{n+1}:
For updating the level set function to a new time step, an explicit Euler time stepping is used for approximating time derivative in equation (5), which can then be written as

$$\frac{\phi^{n+1} - \phi}{\Delta t} + \mathbf{u}.\nabla(\phi) = 0 \tag{19}$$

4.2 Spatial Discretization

This section describes all the spatial discretization used for the governing equations. As noted earlier, all discretization uses collocated grid.

4.2.1 Convective Term

The discretization of the convective term in this algorithm is very similar to the discretization used by [1]. It is a second order Adam-Bashforth discretization in conjunction with centered difference approximation.

$$[(\mathbf{u}.\nabla)\,\mathbf{u}]^{n+1/2} = \frac{\Delta t^n}{2}\left(\frac{\Delta t^n + 2\Delta t^{n-1}}{\Delta t^{n-1}}\,[(\mathbf{u}.\nabla)\,\mathbf{u}]^n - \frac{\Delta t^n}{\Delta t^{n-1}}\,[(\mathbf{u}.\nabla)\,\mathbf{u}]^{n-1}\right) \tag{20}$$

where,

$$[(\mathbf{u}.\nabla)\,\mathbf{u}] = \begin{cases} u_{i,j}\dfrac{u_{i+1,j}-u_{i-1,j}}{2\Delta x} + v_{i,j}\dfrac{u_{i,j+1}-u_{i,j-1}}{2\Delta y} \\ u_{i,j}\dfrac{v_{i+1,j}-v_{i-1,j}}{2\Delta x} + v_{i,j}\dfrac{v_{i,j+1}-v_{i,j-1}}{2\Delta y} \end{cases} \tag{21}$$

4.2.2 Viscous Term

The viscous terms are discretized in straight forward central difference scheme. The viscous term $\nabla\cdot[\mu(\nabla\mathbf{u}+\nabla\mathbf{u}^T)]$ is written in x and y component as:

$$\text{x-direction}: \frac{\partial}{\partial x}\left(2\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right], \tag{22}$$

$$\text{y-direction}: \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)\right] + \frac{\partial}{\partial y}\left(2\mu\frac{\partial v}{\partial y}\right). \tag{23}$$

And the derivatives are discretized as:

$$\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) = \frac{\mu_{i+1/2,j}(\frac{\partial u}{\partial x})_{i+1/2,j} - \mu_{i-1/2,j}(\frac{\partial u}{\partial x})_{i-1/2,j}}{\Delta x} \tag{24}$$

Similar formulation for $\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)$ and other terms is used.

$$\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial y}\right) = \mu_{i+1/2,j}\frac{u_{i+1,j+1}-u_{i+1,j-1}+u_{i,j+1}-u_{i,j-1}}{4\Delta x^2} - \mu_{i-1/2,j}\frac{u_{i,j+1}-u_{i,j-1}+u_{i-1,j+1}-u_{i-1,j-1}}{4\Delta x^2} \tag{25}$$

Similar formulation is used for $\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial x}\right)$ and other similar cross derivative terms.

Where,

$$\mu_{i+1/2,j} = \frac{\mu_{i+1,j}+\mu_{i,j}}{2} \tag{26}$$

$$\left(\frac{\partial u}{\partial x}\right)_{i+1/2,j} = \frac{u_{i+1,j}-u_{i-1,j}}{\Delta x} \tag{27}$$

4.2.3 Curvature Terms

The surface tension equation used is (9), where $k(\phi)$ and $\delta(\phi)$ is given by equation (10) and (11), respectively. The curvature term is discretized making use of equation (24) and (25). The value for curvature $k(\phi)$ is restricted between $\pm\frac{1}{\Delta x}$.

4.2.4 Discretization of projection step

The projection step is given by equation (16). The correct discretization of the projection step operators is very essential on collocated grid to avoid possible checkerboard effect.

1. Discretization of $\nabla\cdot\mathbf{u}^*$:

$$\nabla\cdot\mathbf{u}^* = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

where,

$$\frac{\partial u}{\partial x} = \begin{cases} \dfrac{u_{i+1,j}-u_{i-1,j}}{2\Delta x} & \text{when not at bounday} \\ \dfrac{-3u_{i,j}+4u_{i+1,j}-u_{i+2,j}}{2\Delta x} & \text{at left boundary} \\ \dfrac{3u_{i,j}-4u_{i-1,j}+u_{i-2,j}}{2\Delta x} & \text{at right boundary} \end{cases} \tag{28}$$

Similar formulation is used for $\dfrac{\partial v}{\partial y}$.

2. Discretization of $\nabla.\left(\frac{1}{\rho}(\nabla Q)\right)$:

$$\nabla.\left(\frac{1}{\rho}(\nabla Q)\right) = \frac{\partial}{\partial x}\left(\frac{1}{\rho}(\nabla Q)\right) + \frac{\partial}{\partial y}\left(\frac{1}{\rho}(\nabla Q)\right) \tag{29}$$

$$\frac{\partial}{\partial x}\left(\frac{1}{\rho}(\nabla Q)\right) = \frac{1}{\rho_{i+1/2,j}}\left(\frac{Q_{i+1,j}-Qi,j}{\Delta x}\right) - \frac{1}{\rho_{i-1/2,j}}\left(\frac{Q_{i,j}-Q_{i-1,j}}{\Delta x}\right) \tag{30}$$

Similar discretization is used for $\dfrac{\partial}{\partial y}\left(\frac{1}{\rho}(\nabla Q)\right)$. Also,

$$\frac{1}{\rho_{i+1/2,j}} = \frac{2}{\rho_{i+1,j}+\rho_{i,j}} \tag{31}$$

4.2.5 Reinitialization Equation

The reinitialization equation is given by (12) and due to its hyperbolic nature, it needs to be discretized using conservative schemes. Here first order Godunov scheme is used, also known as automatic upwind scheme. The discrete form of the equation (12) can be written as

$$\phi^{n+1} = \phi^n - \Delta t * \hat{H}(\phi_{i,j}) \tag{32}$$

where $\hat{H}(\phi_x, \phi_y)$ denotes discretized form of $sign(\phi_0)\left(1 - \sqrt{\phi_x^2 + \phi_y^2}\right)$.

We define,

$$\begin{aligned} a &= D_x^- \phi_{i,j} = (\phi_{i,j} - \phi_{i-1,j})/\Delta x \\ b &= D_x^+ \phi_{i,j} = (\phi_{i+1,j} - \phi_{i,j})/\Delta x \\ c &= D_y^- \phi_{i,j} = (\phi_{i,j} - \phi_{i,j-1})/\Delta y \\ d &= D_y^+ \phi_{i,j} = (\phi_{i,j+1} - \phi_{i,j})/\Delta y \\ S &= sign(\phi_0) \end{aligned} \tag{33}$$

and

$$\begin{aligned} &\hat{H}(\phi_{i,j}) \\ &= S^+(\sqrt{max((a^+)^2, (b^-)^2) + max((c^+)^2, (d^-)^2)} - 1) \\ &+ S^-(\sqrt{max((a^-)^2, (b^+)^2) + max((c^-)^2, (d^+)^2)} - 1) \end{aligned} \tag{34}$$

where the notation,

$$\begin{aligned} F^+ &= max(F, 0) \\ F^- &= min(F, 0) \end{aligned} \tag{35}$$

A Sub-cell fix by Russo and Smereka [4] is used near the interface to achieve 2nd order accuracy near interface. We update ϕ near the interface using

$$\phi^{n+1} = \phi^n - \frac{\Delta T}{\Delta x}\left(S|\phi_{i,j}^0| - D_{i,j}\right) \tag{36}$$

where,

$$D_{i,j} = \frac{2\Delta x \phi_{i,j}^0}{\sqrt{(\phi_{i+1,j}^0 - \phi_{i-1,j}^0)^2 + (\phi_{i,j+1}^0 - \phi_{i,j-1}^0)^2}} \tag{37}$$

And this subcell fix is used when any of the following condition is satisfied.

$$\begin{aligned} &(\phi_{i+1,j}^0 * \phi_{i,j}^0 < 0), or \\ &(\phi_{i-1,j}^0 * \phi_{i,j}^0 < 0), or \\ &(\phi_{i,j+1}^0 * \phi_{i,j}^0 < 0), or \\ &(\phi_{i,j-1}^0 * \phi_{i,j}^0 < 0) \end{aligned} \tag{38}$$

4.2.6 Level Set Equation

The level set equation is given by (5). A simple explicit euler scheme is used for approximating time derivative and can be written in discretized form as

$$\phi_{i,j}^{n+1} = \phi_{i,j}^n - \Delta t(u\phi_x + v\phi_y)_{i,j} \tag{39}$$

A simple first order upwind scheme is used for spatial discretization,

$$(\phi_x)_{i,j} = \begin{cases} (\phi_{i,j} - \phi_{i-1,j})/\Delta x \text{ , when } u_{i,j} > 0 \\ (\phi_{i+1,j} - \phi_{i,j})/\Delta x \text{ , when } u_{i,j} < 0 \\ 0 \text{ , otherwise} \end{cases} \tag{40}$$

Similar discretization is used for ϕ_y.

5 Computation of Δt

An adaptive time step is used in the current work. The time step is computed at each time steps which obeys CFD condition due to convective terms, restriction due to source terms (gravity and surface tension) and due to viscous terms. It is computed using following criteria:

$$\begin{aligned} \Delta t_s &= \sqrt{\frac{B(\rho_b + \rho_c)}{8\pi}}\Delta x^{3/2} \\ \Delta t_v &= min\left(\frac{3}{14}(\rho Re \Delta x^2/\mu\right) \\ \Delta t_c &= min\left(\frac{\Delta x}{|\mathbf{u}|}\right) \\ \Delta t_F &= \sqrt{\frac{2\Delta x}{\mathcal{F}^n}} \text{ where, } \mathcal{F}^n = |\frac{\mathcal{L}^n}{\rho} - \frac{\mathcal{T}^n}{\rho} + \mathbf{g}_u| \end{aligned} \tag{41}$$

$$\Delta t = 0.5 min(\Delta t_s, \Delta t_v, \Delta t_c, \Delta t_F)$$

6 Volume Correction

When Level set Method is used in multiphse flow problems, there is a significant volume loss (or mass loss) observed. The attempt to overcome this drawback is studied by various researchers and many different methods has been proposed. This error usually comes from the low order accurate discretization of reinitialization equation and can be reduced significantly by using higher order schemes such as $3^{rd}/5^{th}$ order accurate ENO or WENO. For the present work, a simple volume correction is added to the ϕ after the interface is advanced in time by

$$\phi^{n+1} = \phi^{n+1} + \Delta\phi \tag{42}$$

where $\Delta\phi$ is a volume correction term given by,

$$\Delta\phi = \frac{V^0 - V^n}{L^n} \tag{43}$$

where, V^0 is the volume of the initial bubble at time $t = 0$ and V^n is the volume of the bubble at time $t = n$. And L^n denotes the length of the interface at time $t = n$. The volume and the length of the bubble can be computed by,

$$V = \int_\Omega (1 - H(\phi))d\Omega = \sum_{i,j}(1 - H(\phi_{i,j}))\Delta x \Delta y \tag{44}$$

$$L = \int_\Omega \delta(\phi)d\Omega = \sum_{i,j}(\delta(\phi_{i,j}))\Delta x \Delta y \tag{45}$$

H and δ are smoothed Heaviside and delta functions and are given by equations (8) and (11) respectively.

7 Analysis of results

The above code is tested with the classical problem of multiphase flow which simulates a flow of lower density bubble rising in denser fluid. The key parameters are density ratio (ρ_c/ρ_b), viscosity ratio (μ_c/μ_b), Bond number $B = 4\rho_c \mathbf{g} R^2/\sigma$, and Reynolds number $Re = (2R)^{3/2}\sqrt{\mathbf{g}}\rho_c/\mu_c$. The radius of the bubble, R is taken as 1 unit, computational domain is $7R \times 7R$. The grid of 80×80 nodes is used for the results shown in subsequent sections. No slip boundary condition is applied to all 4 boundaries of the domain hence the flow is generated solely due to density difference. The result is shown for two different cases.

7.1 Grid Independent study

The grid independent study is performed to obtain a grid independent solution which does not change much when the grid is refined further. The figures below shows the solution of interface and flow field at time $t = 3$ sec for Re = 100, B = 200, density and viscosity ratio of 40. The figure (2) shows the interface at 3 sec with the velocity field for the different grids. It can be seen that the solution obtained from 80×80 grid matched closely with 100×100 grid.

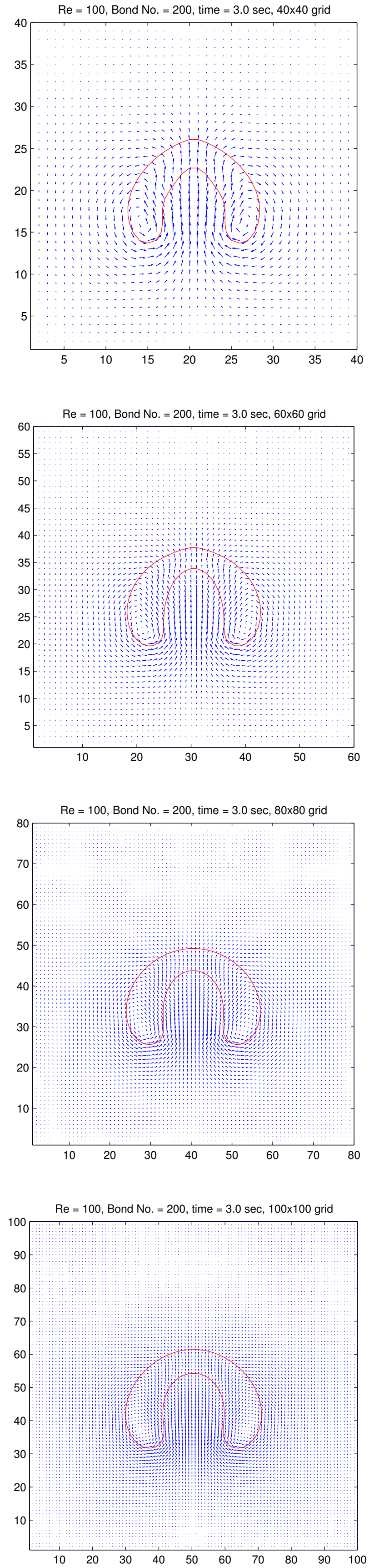

Figure 2: Grid dependence test for grid of 40×40, 60×60, 80×80, 100×100

7.2 Case 1: Flow for Re = 5, Bo = 1

This case simulates flow with very low Reynolds number (Re = 5) and low bond number (B = 1) i.e. high surface tension. The density and viscosity ratio of 40 is used and the grid of 80×80 is used. It can be seen that at low Reynolds number, the bubble does not deform much and obtains a steady state ellipsoidal shape and continues to rise up. The solution at time 1, 2, 3 and 4.2 seconds is shown in figure (3).

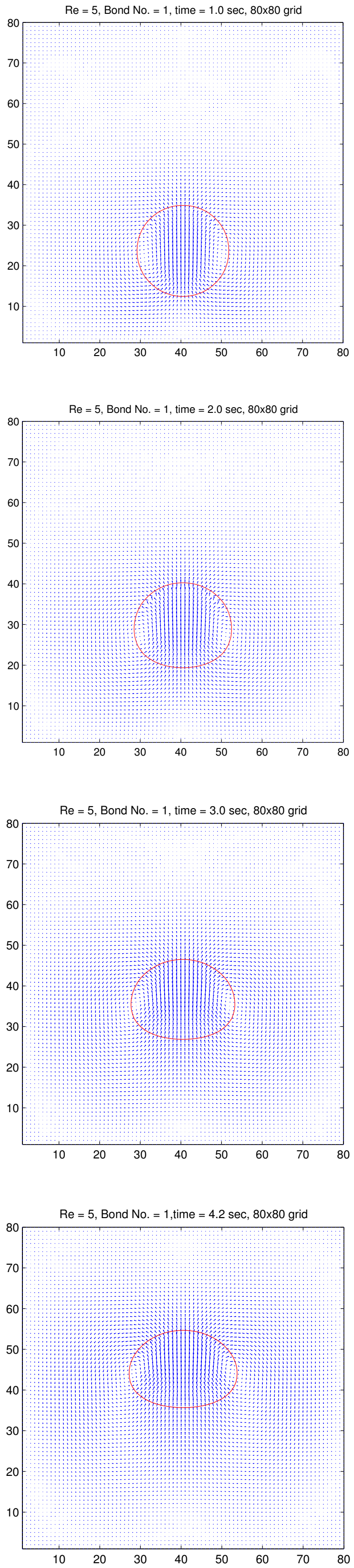

Figure 3: Solution of flow for Reynolds number 5 and Bond Number 1 at different times

7.3 Case 2: Flow for Re = 100, Bo = 200

The flow with Reynolds number 100 and Bond number 200 i.e. low surface tension is simulated with density and viscosity ratio of 40. The grid of 80×80 is used. The behavior of the bubble is very different than that of Re = 1 and Bo = 5 case. The bubbles deforms significantly due to very low surface tension and relatively higher Reynolds number.

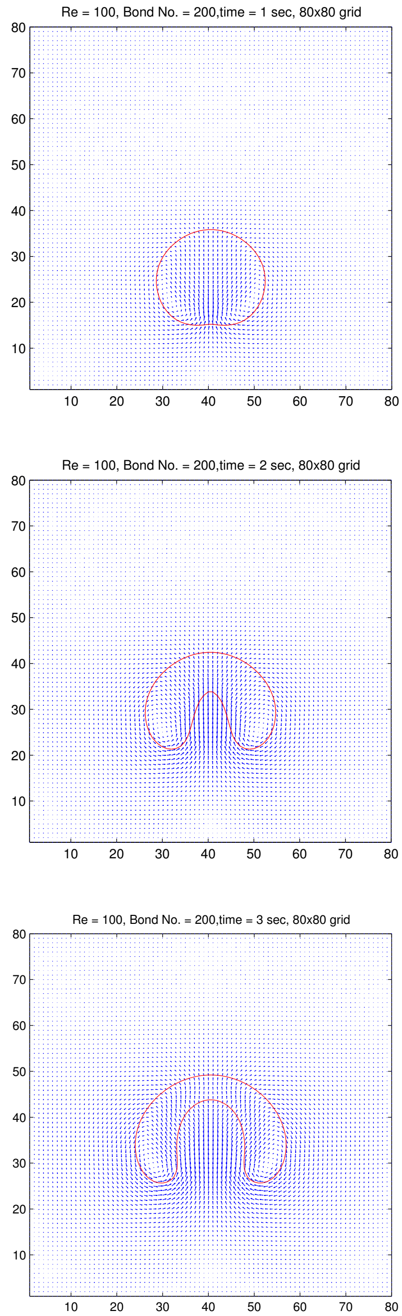

Figure 4: Solution of flow for Reynolds number 100 and Bond Number 200 at different times

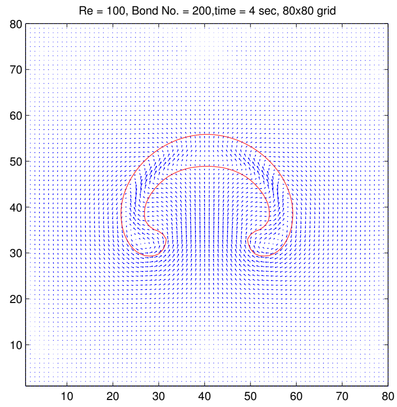

Figure 5: Solution of flow for Reynolds number 100 and Bond Number 200 at different times

7.4 Effect of Volume Correction

The volume correction employed in this code is mentioned in detailed in section (6). The effect of volume correction is shown in the figure (6) where the volume of bubble at each iterations is plotted for the simulation of case 2 with correction term and without. The volume loss can be clearly seen when no volume correction is considered. For the case where volume correction is considered, the volume is very close to the initial volume but is oscillating slightly.

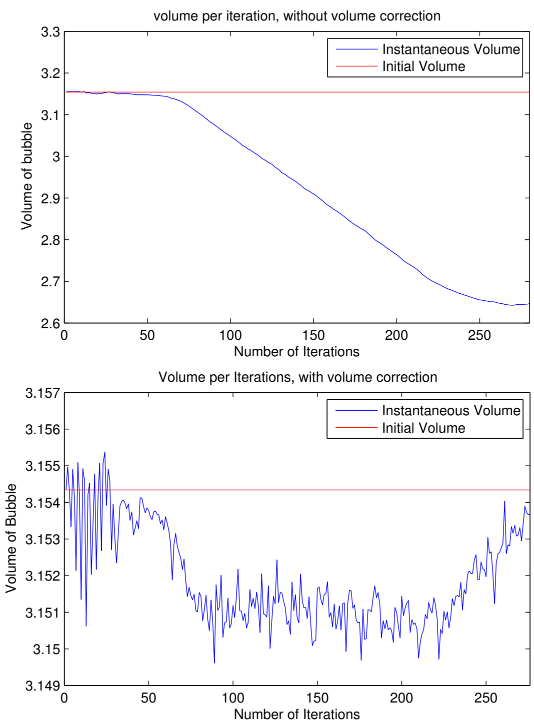

Figure 6: instantaneous Volume at each iteration, with and without volume correction

8 Conclusion

A two-phase incompressible Navier-Stokes flow solver with Level Set method to track the interface is developed and tested with the classical problem on rising bubble. The resulting flow field by solving Navier-Stokes equation is second order accurate in space and time however the discretization of level set is first order accurate. Moreover the effects of surface tension are modeled by using Continuum Surface Force method which is first order near interface. Due to low order discretization used in level set, there is significant volume loss which is corrected by a volume correction method mentioned in section 6. The projection step is solved using GMRES solver built in PETSc library. The density and viscosity ratio used in the simulation are moderate and could get solution for density ratio higher than 100. I think with the use of proper pre-conditioner this limit can be extended to allow higher density ratios.

References

[1] J. Kim and P. Moin, ***Application of a Fractional-Step Method to Incompressible Navier-Stokes equations***, J. Computational Physics, **59**, 308-323 (1985)

[2] J. U. Brackbill, ***A Continuum Method for Modeling Surface Tension***, J. Computational Physics, **100**, 335-354 (1992)

[3] David L. Brown, Ricardo Cortez and Michael L. Minion, ***Accurate Projection Methods for the Incompressible NavierStokes Equations***, J. Computational Physics, **168**, 464-499 (2001)

[4] G. Russo and P. Smereka, ***A Remark on Computing Distance Functions***, J. Computational Physics, **163**, 51-67 (2000)

Appendix: C Program

```c
static char help[] = "Solves 2D multiphase flow problem using level set formulation.\n\n";

#include <petscdm.h>
#include <petscdmda.h>
#include <petscksp.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <engine.h>   //matlab engine


typedef struct{
    Vec U, V, P, rho, mu;
}Solution;

typedef struct{

    Solution soln;
    Solution solstar;
    Solution solnp1;
    Solution solnm1;
    Vec RHSu, RHSv;

    PetscReal h, dt, dt_old, Re, Bond, time;
    PetscReal smallval;

    Vec RHS;
    Mat Jac;
    PetscInt counter, M, N;

    Vec phi_0, phi, phi_new, H;
    Vec normalX, normalY;
    PetscReal Volume0, VolumeNew, Length, dphi;

    double rho_ratio, mu_ratio;
    double gravity, sigma, R;

}UserContext;



/* User defined functions*/
extern PetscErrorCode CreateStructures(DM, UserContext *);
extern PetscErrorCode DestroyStructures(UserContext *);
extern PetscErrorCode PredictVelocity(DM, UserContext *);
extern PetscErrorCode WriteVec(Vec, char const *);
extern PetscErrorCode ComputeExactSol(DM, UserContext *);
extern PetscErrorCode ComputeRHS(DM, UserContext *);
extern PetscErrorCode ComputeMatrix(DM, UserContext *);
extern PetscErrorCode WriteMat(Mat,char const *);
extern PetscErrorCode Projection_Step(KSP, UserContext *);
extern PetscErrorCode UpdateVelocity(DM, UserContext *);
extern PetscErrorCode DefineLevelSet(DM, UserContext *);
extern PetscErrorCode Reinitialize(DM, UserContext *);
extern PetscReal sign(PetscReal);
extern PetscErrorCode DefineVariables(DM, UserContext *);
extern PetscErrorCode AdvectInterface(DM, UserContext *);
extern PetscErrorCode WriteOutput(UserContext *);
extern PetscErrorCode ComputeTimeStep(UserContext *);
extern PetscErrorCode ConserveMass(DM, UserContext*);
extern PetscErrorCode Screenshot(UserContext *, Engine *);



#undef __FUNCT__
#define __FUNCT__ "main"
int main(int argc,char **args){

    KSP ksp;
    DM da;
    UserContext user;
    PetscErrorCode ierr;

    /* start matlab engine */
    //Engine *ep = engOpen(NULL);

    /* Define parameters */
    user.R = 1.0;                    // Radius of bubble
    user.M = 80; user.N = user.M;    // grid size
    user.h = 7*user.R/(user.M-1);
    user.rho_ratio = 1.0/40.0;       // viscosity ratio
    user.mu_ratio = 1.0/40.0;        // density ratio
    user.Re = 100.0;                 // reynolds number
    user.Bond = 200.0;               // bond number
    user.smallval = user.h*user.h;
    int itmax = 2000;                // max iterations
    user.counter = 0;
    user.time = 0.0;

    ierr = PetscInitialize(&argc,&args,(char*)0,help);CHKERRQ(ierr);
    ierr = DMDACreate2d(PETSC_COMM_WORLD,DM_BOUNDARY_NONE,DM_BOUNDARY_NONE, DMDA_STENCIL_BOX, user.M,user.M,
                        PETSC_DECIDE,PETSC_DECIDE,1,1,NULL,NULL,&da); CHKERRQ(ierr);

    ierr = KSPCreate(PETSC_COMM_WORLD,&ksp); CHKERRQ(ierr);
    ierr = CreateStructures(da,&user); CHKERRQ(ierr);

    PetscPrintf(PETSC_COMM_WORLD,"Reynolds no.= %g Bond no.= %g\n",
                user.Re,user.Bond);

    ierr = DefineLevelSet(da,&user); CHKERRQ(ierr);
```

```
    ierr = Reinitialize(da,&user); CHKERRQ(ierr);

    for(user.counter = 0; user.counter < itmax; user.counter++){

        PetscPrintf(PETSC_COMM_WORLD,"\n\nIteration: %d\n",user.counter+1);

        ierr = ComputeTimeStep(&user); CHKERRQ(ierr);
        ierr = AdvectInterface(da,&user); CHKERRQ(ierr);
        ierr = Reinitialize(da,&user); CHKERRQ(ierr);
        ierr = DefineVariables(da,&user); CHKERRQ(ierr);
        ierr = ConserveMass(da,&user); CHKERRQ(ierr);
        ierr = PredictVelocity(da,&user); CHKERRQ(ierr);
        ierr = ComputeRHS(da,&user); CHKERRQ(ierr);
        ierr = ComputeMatrix(da,&user); CHKERRQ(ierr);
        ierr = Projection_Step(ksp,&user); CHKERRQ(ierr);
        ierr = UpdateVelocity(da,&user); CHKERRQ(ierr);
        //ierr = Screenshot(&user,ep); CHKERRQ(ierr);

        if(user.time >= 4.0){
            break;
        }

    }

    ierr = WriteOutput(&user); CHKERRQ(ierr);

    ierr = DestroyStructures(&user); CHKERRQ(ierr);
    ierr = KSPDestroy(&ksp); CHKERRQ(ierr);
    //ierr = DMDestroy(&da); CHKERRQ(ierr);

    ierr = PetscFinalize();

    return(0);
}




/* Create vectors */
#undef __FUNCT__
#define __FUNCT__ "CreateStructures"
PetscErrorCode CreateStructures(DM da, UserContext *user){

    PetscErrorCode ierr;

    ierr = DMCreateGlobalVector(da,&user->soln.U); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->soln.V); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->soln.P); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->soln.rho); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->soln.mu); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solstar.U); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solstar.V); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solstar.P); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnp1.U); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnp1.V); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnp1.P); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnp1.rho); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnp1.mu); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->RHS); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnm1.U); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->solnm1.V); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->phi); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->phi_0); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->phi_new); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->H); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->normalX); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->normalY); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->RHSu); CHKERRQ(ierr);
    ierr = DMCreateGlobalVector(da,&user->RHSv); CHKERRQ(ierr);

    ierr = DMCreateMatrix(da,&user->Jac); CHKERRQ(ierr);
    return(0);
}



/* Destroy vectors */
#undef __FUNCT__
#define __FUNCT__ "DestroyStructures"
PetscErrorCode DestroyStructures(UserContext *user){

    PetscErrorCode ierr;

    ierr = VecDestroy(&user->soln.U); CHKERRQ(ierr);
    ierr = VecDestroy(&user->soln.V); CHKERRQ(ierr);
    ierr = VecDestroy(&user->soln.P); CHKERRQ(ierr);
    ierr = VecDestroy(&user->soln.rho); CHKERRQ(ierr);
    ierr = VecDestroy(&user->soln.mu); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solstar.U); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solstar.V); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solstar.P); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnp1.U); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnp1.V); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnp1.P); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnp1.rho); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnp1.mu); CHKERRQ(ierr);
    ierr = VecDestroy(&user->RHS); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnm1.U); CHKERRQ(ierr);
    ierr = VecDestroy(&user->solnm1.V); CHKERRQ(ierr);
    ierr = VecDestroy(&user->phi); CHKERRQ(ierr);
    ierr = VecDestroy(&user->phi_0); CHKERRQ(ierr);
    ierr = VecDestroy(&user->phi_new); CHKERRQ(ierr);
    ierr = VecDestroy(&user->H); CHKERRQ(ierr);
```

```
    ierr = VecDestroy(&user->normalX); CHKERRQ(ierr);
    ierr = VecDestroy(&user->normalY); CHKERRQ(ierr);
    ierr = VecDestroy(&user->RHSu); CHKERRQ(ierr);
    ierr = VecDestroy(&user->RHSv); CHKERRQ(ierr);

    ierr = MatDestroy(&user->Jac); CHKERRQ(ierr);
    return(0);
}




#undef __FUNCT__
#define __FUNCT__ "WriteVec"
PetscErrorCode WriteVec(Vec vec, char const *name){
    PetscViewer viewer;
    PetscErrorCode ierr;

    /* Create "Output" directory */
    struct stat st = {0};
    if(stat("Output",&st) == -1)
        mkdir("Output",0777);

    char filename[64] = "Output/Vec_"; char pfix[12] = ".m";
    strcat(filename,name); strcat(filename,pfix);
    ierr = PetscObjectSetName((PetscObject)vec,name); CHKERRQ(ierr);
    ierr = PetscViewerASCIIOpen(PETSC_COMM_WORLD, filename, &viewer); CHKERRQ(ierr);
    ierr = PetscViewerSetFormat(viewer, PETSC_VIEWER_ASCII_MATLAB); CHKERRQ(ierr);
    ierr = VecView(vec,viewer);CHKERRQ(ierr);
    ierr = PetscViewerDestroy(&viewer); CHKERRQ(ierr);
    PetscFunctionReturn(0);
}




#undef __FUNCT__
#define __FUNCT__ "PredictVelocity"
extern PetscErrorCode PredictVelocity(DM da, UserContext *user){

  PetscErrorCode ierr;
  PetscReal **_Un, **_Vn, **_Pn, **_Ustr, **_Vstr;
  PetscReal **_Unm1, **_Vnm1, **_Pstr, **_mu, **_rho;
  PetscReal **_N1, **_N2, **_phi, **_H, **_RHSu, **_RHSv;
  PetscReal h,dt, dt_old;
  PetscInt i,j,xs,ys,xm,ym;
  PetscReal ux, uy, vx, vy;
  PetscReal convect, viscous, curvature, delta, surftension;
  double pi = 3.1415926535, eps = 1.5*user->h;
  PetscReal GV1,GV2;
  PetscReal phi_x, phi_y, phi_xx, phi_yy, phi_xy;
  PetscReal mu1, mu2,mu3,mu4;

  h = user->h; dt = user->dt; dt_old = user->dt_old;

  ierr = DMDAVecGetArray(da,user->soln.U,&_Un); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->soln.V,&_Vn); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->soln.P,&_Pn); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->solstar.U,&_Ustr); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->solstar.V,&_Vstr); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->solnm1.U,&_Unm1); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->solnm1.V,&_Vnm1); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->solstar.P,&_Pstr); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->soln.mu,&_mu); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->normalX,&_N1); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->normalY,&_N2); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->phi,&_phi); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->H,&_H); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->RHSu,&_RHSu); CHKERRQ(ierr);
  ierr = DMDAVecGetArray(da,user->RHSv,&_RHSv); CHKERRQ(ierr);

  ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

  for (j=ys; j<ys+ym; j++) {
    for (i=xs; i<xs+xm; i++){
      if(i > 0 && i < user->M-1 && j > 0 && j < user->M-1){

    // u direction

        /* convective term */
        if(user->counter == 0){
            /* central difference */
            ux = 0.5*(_Un[j][i+1] - _Un[j][i-1])/h;
            uy = 0.5*(_Un[j+1][i] - _Un[j-1][i])/h;
            convect = _Un[j][i]*ux + _Vn[j][i]*uy;
        }else if(user->counter > 0){
          /* Adam Bashford */

          convect = 0.5*(((dt + 2*dt_old)/dt_old)*(_Un[j][i]*(0.5*(_Un[j][i+1]-_Un[j][i-1])/h)
                                                + _Vn[j][i]*(0.5*(_Un[j+1][i] - _Un[j-1][i])/h))
                       -(dt/dt_old)*(_Unm1[j][i]*(0.5*(_Unm1[j][i+1] - _Unm1[j][i-1])/h) + _Vnm1[j][i]
                       *(0.5*(_Unm1[j+1][i] - _Unm1[j-1][i])/h)));
        }
        mu1 = 0.5*(_mu[j][i+1] + _mu[j][i]);
        mu2 = 0.5*(_mu[j][i-1] + _mu[j][i]);
        mu3 = 0.5*(_mu[j+1][i] + _mu[j][i]);
        mu4 = 0.5*(_mu[j-1][i] + _mu[j][i]);

        viscous = (2*mu1*((_Un[j][i+1] - _Un[j][i])/h) - 2*mu2*((_Un[j][i] - _Un[j][i-1])/h))/h
               + (mu3*((_Un[j+1][i] - _Un[j][i])/h) - mu4*((_Un[j][i] - _Un[j-1][i])/h))/h
               + (mu3*((_Vn[j+1][i+1] - _Vn[j+1][i-1] + _Vn[j][i+1] - _Vn[j][i-1])/(4*h))
               - mu4*((_Vn[j][i+1] - _Vn[j][i-1] + _Vn[j-1][i+1] - _Vn[j-1][i-1])/(4*h)))/h;
```

```
        viscous = viscous/(user->Re*_rho[j][i]);

        /* surface tension term */
        phi_x = (_phi[j][i+1] - _phi[j][i-1])/(2*h);
        phi_y = (_phi[j+1][i] - _phi[j-1][i])/(2*h);
        phi_xx = (_phi[j][i+1] + _phi[j][i-1] - 2*_phi[j][i])/(h*h);
        phi_yy = (_phi[j+1][i] + _phi[j-1][i] - 2*_phi[j][i])/(h*h);
        phi_xy = (_phi[j+1][i+1] - _phi[j-1][i+1] - _phi[j+1][i-1] + _phi[j-1][i-1])/(4*h*h);

        GV1 = (_phi[j][i+1] - _phi[j][i-1])/(2*h);
        GV2 = (_phi[j+1][i] - _phi[j-1][i])/(2*h);

        curvature = (phi_xx*(phi_y*phi_y) + phi_yy*(phi_x*phi_x) - 2*phi_xy*phi_x*phi_y)
                    / pow((phi_x*phi_x + phi_y*phi_y + user->smallval),1.5);

         // limit curvature
        if(curvature < -1.0/h){
            curvature = -1.0/h;
        }else if(curvature > 1.0/h){
            curvature = 1.0/h;
        }

        // delta function
        if( fabs(_phi[j][i]) <= 1.5*h){
            delta = 0.5*(1 + cos(pi*_phi[j][i]/eps))/eps;
        }else if(fabs(_phi[j][i]) > 1.5*h){
            delta = 0.0;
        }

        surftension = curvature*delta*GV1/(user->Bond*_rho[j][i]);


        /* compute U star */

        if(user->counter == 0){
            /* Backward Euler */
            _Ustr[j][i] = _Un[j][i] + dt*(-convect + viscous - surftension);
            _RHSu[j][i] = -convect + viscous - surftension;
        }
        else if(user->counter > 0){
            /* Adam Bashford */
            _Ustr[j][i] = _Un[j][i] + dt*(0.5*(((dt + 2*dt_old)/dt_old)*(-convect + viscous - surftension)
                          - (dt/dt_old)*_RHSu[j][i]));
            _RHSu[j][i] = -convect + viscous - surftension;
        }

    // v direction

        /* convective term */
        if(user->counter == 0){
           // central difference
            vx = 0.5*(_Vn[j][i+1] - _Vn[j][i-1])/h;
            vy = 0.5*(_Vn[j+1][i] - _Vn[j-1][i])/h;
            convect = _Un[j][i]*vx + _Vn[j][i]*vy;
        }
        else if(user->counter > 0){
            /* Adam Bashford */
                convect = 0.5*(((dt + 2*dt_old)/dt_old)*(_Un[j][i]*(0.5*(_Vn[j][i+1] - _Vn[j][i-1])/h)
                + _Vn[j][i]*(0.5*(_Vn[j+1][i] - _Vn[j-1][i])/h))
                -(dt/dt_old)*(_Unm1[j][i]*(0.5*(_Vnm1[j][i+1] - _Vnm1[j][i-1])/h)
                + _Vnm1[j][i]*(0.5*(_Vnm1[j+1][i] - _Vnm1[j-1][i])/h)));
        }

        viscous = (mu1*((_Vn[j][i+1] - _Vn[j][i])/h) - mu2*((_Vn[j][i] - _Vn[j][i-1])/h))/h
                + (mu1*((_Un[j+1][i+1] - _Un[j-1][i+1] + _Un[j+1][i] - _Un[j-1][i])/(4*h))
                - mu2*((_Un[j+1][i] - _Un[j-1][i] + _Un[j+1][i-1] - _Un[j-1][i-1])/(4*h)))/h
                + (2*mu3*((_Vn[j+1][i] - _Vn[j][i])/h) - 2*mu4*((_Vn[j][i] - _Vn[j-1][i])/h))/h;


        viscous = viscous/(user->Re*_rho[j][i]);

        /* surface tension term */
        surftension = curvature*delta*GV2/(user->Bond*_rho[j][i]);

        /* compute V star */
        if(user->counter == 0){
            _Vstr[j][i] = _Vn[j][i] + dt*(-convect + viscous - surftension -1);
            _RHSv[j][i] = -convect + viscous - surftension -1;
        }
        else if(user->counter > 0){
            /* Adam Bashford */
            _Vstr[j][i] = _Vn[j][i]+dt*(0.5*(((dt + 2*dt_old)/dt_old)*(-convect + viscous - surftension -1)
                          - (dt/dt_old)*_RHSv[j][i]));
            _RHSv[j][i] = -convect + viscous - surftension - 1;
        }

       // boundary condition
      }else if(i == 0 || i == user->M-1 || j == 0 || j == user->M-1){

          /* 4 corners */
          if(i == 0 && j == 0){
              _Ustr[j][i] = _Un[j][i];
              _Vstr[j][i] = _Vn[j][i];
          }else if(i == 0 && j == user->M-1){
              _Ustr[j][i] = _Un[j][i];
              _Vstr[j][i] = _Vn[j][i];
          }else if(i == user->M-1 && j == 0){
              _Ustr[j][i] = _Un[j][i];
              _Vstr[j][i] = _Vn[j][i];
          }else if(i == user->M-1 && j == user->M-1){
              _Ustr[j][i] = _Un[j][i];
              _Vstr[j][i] = _Vn[j][i];
```

```c
            }

            /* normal B.C. */
            if(i == 0 || i == user->M-1){
                _Ustr[j][i] = _Un[j][i];
            }

            if(j == 0 || j == user->M-1){
                _Vstr[j][i] = _Vn[j][i];
            }

            /* tangent b.c */
            if(j == 0 || j == user->M-1){
                if(i != 0 && i != user->M-1){
                    _Ustr[j][i] = _Un[j][i] + dt*(_Pstr[j][i+1] - _Pstr[j][i-1])/(2*h);
                }
            }

            if(i == 0 && i == user->M-1){
                if(j != 0 && j != user->M-1){
                    _Vstr[j][i] = _Vn[j][i] + dt*(_Pstr[j+1][i] -  _Pstr[j-1][i])/(2*h);
                }
            }

        }

    }
  }

  ierr = DMDAVecRestoreArray(da,user->soln.U,&_Un); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->soln.V,&_Vn); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->soln.P,&_Pn); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->solstar.U,&_Ustr); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->solstar.V,&_Vstr); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->solnm1.U,&_Unm1); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->solnm1.V,&_Vnm1); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->solstar.P,&_Pstr); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->soln.mu,&_mu); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->normalX,&_N1); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->normalY,&_N2); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->phi,&_phi); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->H,&_H); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->RHSu,&_RHSu); CHKERRQ(ierr);
  ierr = DMDAVecRestoreArray(da,user->RHSv,&_RHSv); CHKERRQ(ierr);

  return(0);
}





#undef __FUNCT__
#define __FUNCT__ "ComputeRHS"
PetscErrorCode ComputeRHS(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscReal **_Ustr, **_Vstr, **_b;
    PetscInt i,j,xs,ys,xm,ym;
    PetscReal dt = user->dt, h = user->h, ux, vy;

    ierr = DMDAVecGetArray(da,user->solstar.U,&_Ustr); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->solstar.V,&_Vstr); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->RHS,&_b); CHKERRQ(ierr);
    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    for(i = xs; i < xs+xm; i++){
        for(j = ys; j < ys+ym; j++){

            if(i == 0){
                ux = 0.5*(-3*_Ustr[j][i] + 4*_Ustr[j][i+1] - _Ustr[j][i+2])/h;
            }else if(i == user->M-1){
                ux = 0.5*(3*_Ustr[j][i] - 4*_Ustr[j][i-1] + _Ustr[j][i-2])/h;
            }else if(i > 0 && i < user->M-1 ){
                ux = (0.5*(_Ustr[j][i+1] - _Ustr[j][i-1]))/h;
                //ux = (0.5*(_Ustr[j][i] + _Ustr[j][i+1]) - 0.5*(_Ustr[j][i] + _Ustr[j][i-1]))/h;
            }

            if(j == 0){
                vy = 0.5*(-3*_Vstr[j][i] + 4*_Vstr[j+1][i] - _Vstr[j+2][i])/h;
            }else if (j == user->M-1){
                vy = 0.5*(3*_Vstr[j][i] - 4*_Vstr[j-1][i] + _Vstr[j-2][i])/h;
            }else if(j > 0 && j < user->M-1){
                vy = (0.5*(_Vstr[j+1][i] - _Vstr[j-1][i]))/h;
                //vy = (0.5*(_Ustr[j][i] + _Ustr[j+1][i]) - 0.5*(_Ustr[j][i] + _Ustr[j-1][i]))/h;
            }

            _b[j][i] = (ux + vy)/dt;

        }
    }

    ierr = VecAssemblyBegin(user->RHS); CHKERRQ(ierr);
    ierr = VecAssemblyEnd(user->RHS); CHKERRQ(ierr);

    ierr = DMDAVecRestoreArray(da,user->solstar.U,&_Ustr); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->solstar.V,&_Vstr); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->RHS,&_b); CHKERRQ(ierr);

    MatNullSpace nullspace;
    ierr = MatNullSpaceCreate(PETSC_COMM_WORLD,PETSC_TRUE,0,0,&nullspace); CHKERRQ(ierr);
    ierr = MatNullSpaceRemove(nullspace,user->RHS); CHKERRQ(ierr);
```

```
    ierr = MatNullSpaceDestroy(&nullspace); CHKERRQ(ierr);


    return(0);
}



#undef __FUNCT__
#define __FUNCT__ "ComputeMatrix"
PetscErrorCode ComputeMatrix(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscInt i,j,xs,ys,xm,ym;
    PetscReal v[5];
    MatStencil row, col[5];
    PetscReal h = user->h, h2 = h*h;
    PetscReal a1, a2, a3, a4, a5;
    PetscReal **_rho;

    ierr = DMDAVecGetArray(da,user->soln.rho,&_rho);
    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    if(user->counter > 0){
        ierr = MatZeroEntries(user->Jac);
    }


    for (j=ys; j<ys+ym; j++) {
        for (i=xs; i<xs+xm; i++) {

            row.i = i; row.j = j;

            /* compute coefficients */
            if(i == 0){
                a2 = 0;
                a4 = (2.0/(_rho[j][i] + _rho[j][i+1]))/h2;
            }else if(i == user->M-1){
                a2 = (2.0/(_rho[j][i] + _rho[j][i-1]))/h2;
                a4 = 0;
            }else if(i != 0 && i != user->M-1){
                a2 = (2.0/(_rho[j][i] + _rho[j][i-1]))/h2;
                a4 = (2.0/(_rho[j][i] + _rho[j][i+1]))/h2;
            }

            if(j == 0){
                a1 = 0;
                a5 = (2.0/(_rho[j][i] + _rho[j+1][i]))/h2;
            }else if(j == user->N-1){
                a1 = (2.0/(_rho[j][i] + _rho[j-1][i]))/h2;
                a5 = 0;
            }else if(j != 0 && j != user->N-1){
                a1 = (2.0/(_rho[j][i] + _rho[j-1][i]))/h2;
                a5 = (2.0/(_rho[j][i] + _rho[j+1][i]))/h2;
            }

            a3 = -(a1 + a2 + a4 + a5);

            /* assemble matrix */

            if(i == 0 && j == 0){
                v[0] = a3;          col[0].i = i;   col[0].j = j;
                v[1] = a4;          col[1].i = i+1; col[1].j = j;
                v[2] = a5;          col[2].i = i;   col[2].j = j+1;
                ierr = MatSetValuesStencil(user->Jac,1,&row,3,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i == 0 && j != 0 && j != user->N-1){
                v[0] = a1;          col[0].i = i;   col[0].j = j-1;
                v[1] = a3;          col[1].i = i;   col[1].j = j;
                v[2] = a4;          col[2].i = i+1; col[2].j = j;
                v[3] = a5;          col[3].i = i;   col[3].j = j+1;
                ierr = MatSetValuesStencil(user->Jac,1,&row,4,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i == 0 && j == user->N-1){
                v[0] = a1;          col[0].i = i;   col[0].j = j-1;
                v[1] = a3;          col[1].i = i;   col[1].j = j;
                v[2] = a4;          col[2].i = i+1; col[2].j = j;
                ierr = MatSetValuesStencil(user->Jac,1,&row,3,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i != 0 && i != user->M-1 && j == 0){
                v[0] = a2;          col[0].i = i-1; col[0].j = j;
                v[1] = a3;          col[1].i = i;   col[1].j = j;
                v[2] = a4;          col[2].i = i+1; col[2].j = j;
                v[3] = a5;          col[3].i = i;   col[3].j = j+1;
                ierr = MatSetValuesStencil(user->Jac,1,&row,4,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i == user->M-1 && j == 0){
                v[0] = a2;          col[0].i = i-1; col[0].j = j;
                v[1] = a3;          col[1].i = i;   col[1].j = j;
                v[2] = a5;          col[2].i = i;   col[2].j = j+1;
                ierr = MatSetValuesStencil(user->Jac,1,&row,3,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i == user->M-1 && j != 0 && j != user->N-1){
                v[0] = a1;          col[0].i = i;   col[0].j = j-1;
                v[1] = a2;          col[1].i = i-1; col[1].j = j;
                v[2] = a3;          col[2].i = i;   col[2].j = j;
                v[3] = a5;          col[3].i = i;   col[3].j = j+1;
                ierr = MatSetValuesStencil(user->Jac,1,&row,4,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i == user->M-1 && j == user->N-1){
                v[0] = a1;          col[0].i = i;   col[0].j = j-1;
                v[1] = a2;          col[1].i = i-1; col[1].j = j;
                v[2] = a3;          col[2].i = i;   col[2].j = j;
```

```
                ierr = MatSetValuesStencil(user->Jac,1,&row,3,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i != 0 && i != user->M-1 && j == user->N-1){
                v[0] = a1;          col[0].i = i;   col[0].j = j-1;
                v[1] = a2;          col[1].i = i-1; col[1].j = j;
                v[2] = a3;          col[2].i = i;   col[2].j = j;
                v[3] = a4;          col[3].i = i+1; col[3].j = j;
                ierr = MatSetValuesStencil(user->Jac,1,&row,4,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }
            else if(i > 0 && i < user->M-1 && j > 0 && j < user->N-1){
                v[0] = a1;          col[0].i = i;   col[0].j = j-1;
                v[1] = a2;          col[1].i = i-1; col[1].j = j;
                v[2] = a3;          col[2].i = i;   col[2].j = j;
                v[3] = a4;          col[3].i = i+1; col[3].j = j;
                v[4] = a5;          col[4].i = i;   col[4].j = j+1;
                ierr = MatSetValuesStencil(user->Jac,1,&row,5,col,v,INSERT_VALUES); CHKERRQ(ierr);
            }

        }
    }

    ierr = MatAssemblyBegin(user->Jac,MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
    ierr = MatAssemblyEnd(user->Jac,MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);

    ierr = MatSetOption(user->Jac,MAT_NEW_NONZERO_LOCATIONS,PETSC_TRUE); CHKERRQ(ierr);

    ierr = DMDAVecRestoreArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);

    MatNullSpace nullspace;
    ierr = MatNullSpaceCreate(PETSC_COMM_WORLD,PETSC_TRUE,0,0,&nullspace); CHKERRQ(ierr);
    ierr = MatSetNullSpace(user->Jac,nullspace); CHKERRQ(ierr);
    ierr = MatNullSpaceDestroy(&nullspace); CHKERRQ(ierr);

    return(0);
}




#undef __FUNCT__
#define __FUNCT__ "WriteMat"
PetscErrorCode WriteMat(Mat mat,char const *name){

    PetscViewer viewer;
    PetscErrorCode ierr;

    /* Create "Output" directory */
    struct stat st = {0};
    if(stat("Output",&st) == -1)
        mkdir("Output",0777);

    char filename[64] = "Output/Mat_"; char pfix[12] = ".m";
    strcat(filename,name); strcat(filename,pfix);
    ierr = PetscObjectSetName((PetscObject)mat,name); CHKERRQ(ierr);
    ierr = PetscViewerASCIIOpen(PETSC_COMM_WORLD, filename, &viewer); CHKERRQ(ierr);
    ierr = PetscViewerSetFormat(viewer, PETSC_VIEWER_ASCII_MATLAB); CHKERRQ(ierr);
    ierr = MatView(mat,viewer);CHKERRQ(ierr);
    ierr = PetscViewerDestroy(&viewer); CHKERRQ(ierr);
    PetscFunctionReturn(0);
}





#undef __FUNCT__
#define __FUNCT__ "Projection_Step"
PetscErrorCode Projection_Step(KSP ksp, UserContext *user){

    PetscErrorCode ierr;
    MatNullSpace nullsp;
    PetscInt itn;

    ierr = KSPSetOperators(ksp,user->Jac,user->Jac); CHKERRQ(ierr);

    if (user->counter > 0){
        ierr = KSPSetInitialGuessNonzero(ksp,PETSC_TRUE);
    }


    ierr = MatNullSpaceCreate(PETSC_COMM_WORLD, PETSC_TRUE, 0, NULL, &nullsp); CHKERRQ(ierr);
    ierr = KSPSetNullSpace(ksp, nullsp); CHKERRQ(ierr);
    ierr = KSPSetTolerances(ksp,1.e-9,PETSC_DEFAULT,PETSC_DEFAULT,PETSC_DEFAULT); CHKERRQ(ierr);

    ierr = KSPSetFromOptions(ksp); CHKERRQ(ierr);
    ierr = KSPSetUp(ksp); CHKERRQ(ierr);
    ierr = KSPSolve(ksp,user->RHS,user->solstar.P); CHKERRQ(ierr);
    ierr = MatNullSpaceDestroy(&nullsp); CHKERRQ(ierr);
    ierr = KSPGetIterationNumber(ksp,&itn); CHKERRQ(ierr);
    ierr = PetscPrintf(PETSC_COMM_WORLD,"KSP: %d\n",itn); CHKERRQ(ierr);

    ierr = MatNullSpaceCreate(PETSC_COMM_WORLD,PETSC_TRUE,0,0,&nullsp); CHKERRQ(ierr);
    ierr = MatNullSpaceRemove(nullsp,user->solstar.P); CHKERRQ(ierr);
    ierr = MatNullSpaceDestroy(&nullsp); CHKERRQ(ierr);

    return(0);

}




#undef __FUNCT__
```

```c
#define __FUNCT__ "UpdateVelocity"
extern PetscErrorCode UpdateVelocity(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscReal **_Unp1, **_Vnp1, **_Ustr, **_Vstr, **_Pnp1, **_Pstr, **_Pn, **_Un, **_Vn, **_rho;
    PetscInt i,j,xs,ys,xm,ym;
    PetscReal dt = user->dt, h = user->h;

    ierr = DMDAVecGetArray(da,user->solnp1.U,&_Unp1); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->solnp1.V,&_Vnp1); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->solnp1.P,&_Pnp1); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->solstar.U,&_Ustr); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->solstar.V,&_Vstr); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->solstar.P,&_Pstr); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.P,&_Pn); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.U,&_Un); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.V,&_Vn); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);

    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    for (j=ys; j<ys+ym; j++) {
        for (i=xs; i<xs+xm; i++) {
            if(i > 0 && i < user->M-1 && j > 0 && j < user->M-1){

                _Unp1[j][i] = _Ustr[j][i] - dt*(1.0/_rho[j][i])*((_Pstr[j][i+1] - _Pstr[j][i-1])/(2*h));

                _Vnp1[j][i] = _Vstr[j][i] - dt*(1.0/_rho[j][i])*((_Pstr[j+1][i] - _Pstr[j-1][i])/(2*h));

                _Pnp1[j][i] = _Pstr[j][i];
            }
            else if(i ==0 || j == 0 || i == user->M-1 || j == user->M-1){

                _Unp1[j][i] = _Un[j][i];
                _Vnp1[j][i] = _Vn[j][i];
                _Pnp1[j][i] = _Pstr[j][i];
            }

        }
    }

    ierr = DMDAVecRestoreArray(da,user->solnp1.U,&_Unp1); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->solnp1.V,&_Vnp1); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->solnp1.P,&_Pnp1); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->solstar.U,&_Ustr); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->solstar.V,&_Vstr); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->solstar.P,&_Pstr); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.P,&_Pn); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.U,&_Un); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.V,&_Vn); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);

    ierr = VecCopy(user->soln.U,user->solnm1.U); CHKERRQ(ierr);
    ierr = VecCopy(user->soln.V,user->solnm1.V); CHKERRQ(ierr);

    ierr = VecCopy(user->solnp1.U,user->soln.U); CHKERRQ(ierr);
    ierr = VecCopy(user->solnp1.V,user->soln.V); CHKERRQ(ierr);
    ierr = VecCopy(user->solnp1.P,user->soln.P); CHKERRQ(ierr);

    return(0);
}




#undef __FUNCT__
#define __FUNCT__ "DefineLevelSet"
PetscErrorCode DefineLevelSet(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscInt i,j,xs,ys,xm,ym;
    DM cda;
    Vec xy;
    DMDACoor2d **_xy;
    PetscReal **_phi, R = user->R;
    PetscReal xmin, xmax, ymin,ymax, xbubble, ybubble;

    xmin = 0.0; xmax = 7*R; ymin = 0.0; ymax = 7*R;
    xbubble = xmin + fabs(xmax-xmin)/2; ybubble = ymin + fabs(xmax-xmin)/4;

    ierr = DMDASetUniformCoordinates(da,xmin,xmax,ymin,ymax,0.0,1.0); CHKERRQ(ierr);

    /* Get coordinates */
    ierr = DMGetCoordinateDM(da,&cda); CHKERRQ(ierr);
    ierr = DMGetCoordinatesLocal(da,&xy); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(cda,xy,&_xy); CHKERRQ(ierr);

    /* Get Vec arrays */
    ierr = DMDAVecGetArray(da,user->phi_0,&_phi); CHKERRQ(ierr);

    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    /* Compute exact solution */
    for(i = xs; i < xs+xm; i++){
        for(j = ys; j < ys+ym; j++){

            _phi[j][i] = pow((_xy[j][i].x - xbubble),2) + pow((_xy[j][i].y - ybubble),2) - R*R;

        }
    }

    /* Restore Vec arrays */
    ierr = DMDAVecRestoreArray(da,user->phi_0,&_phi); CHKERRQ(ierr);
```

```
    ierr = VecDestroy(&xy); CHKERRQ(ierr);
    ierr = DMDestroy(&cda); CHKERRQ(ierr);

    return(0);
}





#undef __FUNCT__
#define __FUNCT__ "Reinitialize"
PetscErrorCode Reinitialize(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscReal **_phi_0, **_phi, **_phi_new, **_Un, **_Vn;
    PetscInt i,j,xs,ys,xm,ym;
    PetscReal s, a, b, c, d, a_plus, a_minus, b_plus, b_minus,c_plus,
            c_minus, d_plus, d_minus, s_plus, s_minus, D, phi_x, phi_y;

    PetscReal phix, phiy, eps = user->h*user->h;

    PetscReal h = user->h, norm;

    int t, itmax = 5*user->M;        // max iterations;
    PetscReal dt = 0.5*h;            // time step size(dTau)

    if(user->counter == 0){
        itmax = 5*user->M;       // max iterations
    }else if(user->counter > 0){
        itmax = 50;        // max iterations
    }


    ierr = VecCopy(user->phi_0,user->phi); CHKERRQ(ierr);


    ierr = DMDAVecGetArray(da,user->phi_0,&_phi_0); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->phi,&_phi); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->phi_new,&_phi_new); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.U,&_Un); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.V,&_Vn); CHKERRQ(ierr);

    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    for(t = 0; t < itmax; t++){

        for (j=ys; j<ys+ym; j++) {
            for (i=xs; i<xs+xm; i++) {

                /* Compute a, b, c, d */
                if (i == 0){ // B.C.on a using Ghost Node
                    a = (_phi[j][i + 1] - _phi[j][i]) / h;
                }else{
                    a = (_phi[j][i] - _phi[j][i - 1]) / h;
                }

                if (i == user->M - 1){  // B.C.on b using Ghost Node
                    b = (_phi[j][i] - _phi[j][i - 1]) / h;
                }else{
                    b = (_phi[j][i + 1] - _phi[j][i]) / h;
                }

                if (j == 0){ //B.C.on c using Ghost Node
                    c = (_phi[j + 1][i] - _phi[j][i]) / h;
                }else{
                    c = (_phi[j][i] - _phi[j - 1][i]) / h;
                }
                if (j == user->N - 1){  //B.C.on d using Ghost Node
                    d = (_phi[j][i] - _phi[j - 1][i]) / h;
                }else{
                    d = (_phi[j + 1][i] - _phi[j][i]) / h;
                }

                // Compute phi_x
                if (i == 0){
                    phi_x = (_phi[j][i + 1] - _phi[j][i]) / h;
                }
                else if (i == user->N - 1){
                    phi_x = (_phi[j][i] - _phi[j][i - 1]) / h;
                }
                else{ phi_x = (_phi[j][i + 1] - _phi[j][i - 1]) / (2 * h); }

                // Compute phi_y
                if (j == 0){
                    phi_y = (_phi[j + 1][i] - _phi[j][i]) / h;
                }
                else if (j == user->N - 1){
                    phi_y = (_phi[j][i] - _phi[j - 1][i]) / h;
                }
                else{ phi_y = (_phi[j + 1][i] - _phi[j - 1][i]) / (2 * h); }

                /* mollified sign function */
                s = _phi[j][i] / (sqrt(pow(_phi[j][i], 2) + (pow(phi_x, 2) + pow(phi_y, 2))*pow(h, 2)));

                /* Compute positive and negative contribution */
                a_plus = fmax(a, 0.0);
                a_minus = fmin(a, 0.0);
                b_plus = fmax(b, 0.0);
                b_minus = fmin(b, 0.0);
                c_plus = fmax(c, 0.0);
```

```
                c_minus = fmin(c, 0.0);
                d_plus = fmax(d, 0.0);
                d_minus = fmin(d, 0.0);
                s_plus = fmax(s, 0.0);
                s_minus = fmin(s, 0.0);


                int flag = 0;
                /* if interface exists -- Use Subcell Fix by Russo and Smereka */
                if(i > 0 && i < user->M-1 && j > 0 && j < user->N-1){
                    if (_phi[j][i]*_phi[j][i+1] < 0 || _phi[j][i]*_phi[j][i-1] < 0
                        || _phi[j][i]*_phi[j+1][i] < 0 || _phi[j][i]*_phi[j-1][i] < 0){

                        phix = ((_phi_0[j][i] + _phi_0[j][i+1])/2 - (_phi_0[j][i] + _phi_0[j][i-1])/2)/h;
                        phiy = ((_phi_0[j][i] + _phi_0[j+1][i])/2 - (_phi_0[j][i] + _phi_0[j-1][i])/2)/h;

                        D = _phi_0[j][i] / sqrt(pow(phix,2) + pow(phiy,2) + eps);
                        // sub-cell fix
                        _phi_new[j][i] = _phi[j][i] - (dt / h)*(sign(_phi_0[j][i])*fabs(_phi[j][i]) - D);
                        flag = 1;

                    }
                }

                if(flag == 0){
                    _phi_new[j][i] = _phi[j][i] - dt*(s_plus*(pow((fmax(pow(a_plus, 2), pow(b_minus, 2)))
                                    + (fmax(pow(c_plus, 2), pow(d_minus, 2))), 0.5) - 1)
                                    + s_minus*(pow((fmax(pow(a_minus, 2), pow(b_plus, 2)))
                                    + (fmax(pow(c_minus, 2), pow(d_plus, 2))), 0.5) - 1));
                }

            }
        }

        ierr = VecAXPY(user->phi,-1.0,user->phi_new); CHKERRQ(ierr);
        ierr = VecNorm(user->phi,NORM_2,&norm);
        if(norm <= 0.000001){
            ierr = VecCopy(user->phi_new,user->phi); CHKERRQ(ierr);
            break;
        }
        ierr = VecCopy(user->phi_new,user->phi); CHKERRQ(ierr);


    }

    PetscPrintf(PETSC_COMM_WORLD,"SDF: %d and norm = %e\n",t,norm);

    ierr = DMDAVecRestoreArray(da,user->phi_0,&_phi_0); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->phi,&_phi); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->phi_new,&_phi_new); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.U,&_Un); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.V,&_Vn); CHKERRQ(ierr);

    return(0);
}





#undef __FUNCT__
#define __FUNCT__ "sign"
PetscReal sign(PetscReal sgn){
    if (sgn < 0){ return (-1.0); }
    else if (sgn > 0){ return (1.0); }
    else return (0.0);
}




#undef __FUNCT__
#define __FUNCT__ "DefineVariables"
PetscErrorCode DefineVariables(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscInt i,j,xs,ys,xm,ym;
    PetscReal **_H, **_phi, **_Mu, **_rho;
    PetscReal eps = 1.5*user->h;
    double pi = 3.1415926535, rho_ratio = user->rho_ratio, mu_ratio = user->mu_ratio;

    ierr = DMDAVecGetArray(da,user->soln.mu,&_Mu); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->phi,&_phi); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->H,&_H); CHKERRQ(ierr);

    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    for (j=ys; j<ys+ym; j++) {
        for (i=xs; i<xs+xm; i++) {

            /* compute heaviside */
            if(_phi[j][i] < -eps)
                _H[j][i] = 0;
            else if(fabs(_phi[j][i]) <= eps)
                _H[j][i] = 0.5*(1.0 + _phi[j][i]/eps + sin(pi*_phi[j][i]/eps)/pi);
            else if(_phi[j][i] > eps)
                _H[j][i] = 1;

            /* Define Viscosity */
            _Mu[j][i] = _H[j][i] + (mu_ratio)*(1.0 - _H[j][i]);

            /* Define Density */
```

```
                _rho[j][i] = _H[j][i] + (rho_ratio)*(1.0 - _H[j][i]);

            }
        }

    ierr = DMDAVecRestoreArray(da,user->soln.mu,&_Mu); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.rho,&_rho); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->phi,&_phi); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->H,&_H); CHKERRQ(ierr);

    return(0);
}



#undef __FUNCT__
#define __FUNCT__ "AdvectInterface"
PetscErrorCode AdvectInterface(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscReal **_phi_new, **_phi, **_U, **_V;
    PetscReal phix, phiy;
    PetscReal h = user->h, dt = user->dt;
    PetscInt i,j,xs,ys,xm,ym;

    ierr = DMDAVecGetArray(da,user->phi_new,&_phi_new); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->phi,&_phi); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.U,&_U); CHKERRQ(ierr);
    ierr = DMDAVecGetArray(da,user->soln.V,&_V); CHKERRQ(ierr);

    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    for (j=ys; j<ys+ym; j++) {
        for (i=xs; i<xs+xm; i++) {

            if(i == 0){
                phix = (_phi[j][i+1] - _phi[j][i]) / h;
            }else if(i == user->M-1){
                phix = (_phi[j][i] - _phi[j][i - 1]) / h;
            }else if(i !=0 && i != user->M-1){
                if(_U[j][i] > 0){
                    phix = (_phi[j][i] - _phi[j][i-1])/h;
                }else if(_U[j][i] < 0){
                    phix = (_phi[j][i+1] - _phi[j][i])/h;
                }else if(_U[j][i] == 0){
                    phix = 0.0;
                }
            }


            if(j == 0){
                phiy = (_phi[j+1][i] - _phi[j][i]) / h;
            }else if(j == user->N-1){
                phiy = (_phi[j][i] - _phi[j-1][i]) / h;
            }else if(j != 0 && j != user->N-1){
                if(_V[j][i] > 0){
                    phiy = (_phi[j][i] - _phi[j-1][i])/h;
                }else if(_V[j][i] < 0){
                    phiy = (_phi[j+1][i] - _phi[j][i])/h;
                }else if(_V[j][i] == 0){
                    phiy = 0.0;
                }
            }

                _phi_new[j][i] = _phi[j][i] - dt*(_U[j][i]*phix + _V[j][i]*phiy);
        }
    }

    ierr = DMDAVecRestoreArray(da,user->phi_new,&_phi_new); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->phi,&_phi); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.U,&_U); CHKERRQ(ierr);
    ierr = DMDAVecRestoreArray(da,user->soln.V,&_V); CHKERRQ(ierr);

    ierr = VecCopy(user->phi_new,user->phi_0); CHKERRQ(ierr);

    return(0);
}


#undef __FUNCT__
#define __FUNCT__ "ComputeTimeStep"
extern PetscErrorCode ComputeTimeStep(UserContext *user){

    PetscErrorCode ierr;
    PetscReal dts, dtv, dtv1, dtv2, dtc, dtc1, dtc2, dtf, dtf1, dtf2;
    PetscReal rhoWater = 1.0, rhoAir = 0.001226;
    PetscReal muWater = 0.01137, muAir = 0.000178;
    double pi = 3.1415926535;
    PetscReal umax, umin, vmax, vmin,u,v, RHSumax, RHSumin, RHSvmax,RHSvmin,f1,f2;

    dts = sqrt(((rhoWater + rhoAir)*user->Bond)/(8.0*pi))*pow(user->h,1.5);

    dtv1 = (3.0/14.0*user->Re)*rhoWater*(user->h*user->h)/muWater;

    dtv2 = (3.0/14.0*user->Re)*rhoAir*(user->h*user->h)/muAir;

    ierr = VecMax(user->soln.U,NULL,&umax); CHKERRQ(ierr);
    ierr = VecMin(user->soln.U,NULL,&umin); CHKERRQ(ierr);
    ierr = VecMax(user->soln.V,NULL,&vmax); CHKERRQ(ierr);
    ierr = VecMin(user->soln.V,NULL,&vmin); CHKERRQ(ierr);
    ierr = VecMax(user->RHSu,NULL,&RHSumax); CHKERRQ(ierr);
    ierr = VecMin(user->RHSu,NULL,&RHSumin); CHKERRQ(ierr);
    ierr = VecMax(user->RHSv,NULL,&RHSvmax); CHKERRQ(ierr);
```

```c
    ierr = VecMin(user->RHSv,NULL,&RHSvmin); CHKERRQ(ierr);

    u = fmax(fabs(umax),fabs(umin));
    v = fmax(fabs(vmax),fabs(vmin));

    dtc1 = user->h/u;
    dtc2 = user->h/v;

    dtv = fmin(dtv1,dtv2);
    dtc = fmin(dtc1,dtc2);

    f1 = fmax(fabs(RHSumax),fabs(RHSumin));
    f2 = fmax(fabs(RHSvmax),fabs(RHSvmin));

    dtf1 = 2*user->h/f1;
    dtf2 = 2*user->h/f2;

    dtf = fmin(dtf1,dtf2);

    user->dt_old = user->dt;

    user->dt = 0.5*fmin(fmin(fmin(dts,dtv),dtc),dtf);

    user->time = user->time + user->dt;

    PetscPrintf(PETSC_COMM_WORLD,"Physical Time: %g\n",user->time);
    PetscPrintf(PETSC_COMM_WORLD,"dt: %g\n",user->dt);
    PetscPrintf(PETSC_COMM_WORLD,"umax: %g, vmax: %g\n",u,v);

    return(0);
}




#undef __FUNCT__
#define __FUNCT__ "WriteOutput"
extern PetscErrorCode WriteOutput(UserContext *user){

    PetscErrorCode ierr;

    /* normal to interface */
    //WriteVec(user->normalX,"normalX"); CHKERRQ(ierr);
    //WriteVec(user->normalY,"normalY"); CHKERRQ(ierr);

    /* write intermediate velocity field */
    //ierr = WriteVec(user->solstar.U,"Ustar"); CHKERRQ(ierr);
    //ierr = WriteVec(user->solstar.V,"Vstar"); CHKERRQ(ierr);

    /* RHS of projection step */
    //WriteVec(user->RHS,"b"); CHKERRQ(ierr);

    /* write jacobian matrix */
    //ierr = WriteMat(user->Jac,"Jac"); CHKERRQ(ierr);

    /* write the KSP solution */
    //ierr = WriteVec(user->solstar.P,"KSP"); CHKERRQ(ierr);

    /* velocity at n+1 */
    ierr = WriteVec(user->solnp1.U,"Unp1"); CHKERRQ(ierr);
    ierr = WriteVec(user->solnp1.V,"Vnp1"); CHKERRQ(ierr);
    //ierr = WriteVec(user->solnp1.P,"Pnp1"); CHKERRQ(ierr);

    /* define level set */
    //ierr = WriteVec(user->phi_0,"levelset"); CHKERRQ(ierr);

    /* reinitialize */
    //ierr = WriteVec(user->phi,"phi1"); CHKERRQ(ierr);

    /* define variables */
    //WriteVec(user->H,"Heaviside");
    //WriteVec(user->soln.mu,"Mu");
    //WriteVec(user->soln.rho,"Rho");

    /* advect level set */
    ierr = WriteVec(user->phi_new,"phinp1"); CHKERRQ(ierr);

    return(0);
}





#undef __FUNCT__
#define __FUNCT__ "ConserveMass"
PetscErrorCode ConserveMass(DM da, UserContext *user){

    PetscErrorCode ierr;
    PetscReal h = user->h;
    PetscInt i,j,xs,ys,xm,ym;
    PetscReal **_H, **_phi, delta;
    double pi = 3.1415926535, eps = 2*user->h;

    ierr = DMDAVecGetArray(da,user->H,&_H);
    ierr = DMDAVecGetArray(da,user->phi,&_phi);

    ierr = DMDAGetCorners(da,&xs,&ys,NULL,&xm,&ym,NULL); CHKERRQ(ierr);

    user->VolumeNew = 0.0; user->Length = 0.0;

    for (j=ys; j<ys+ym; j++) {
        for (i=xs; i<xs+xm; i++) {
```

```
            /* compute volume */
            if(user->counter == 0){
                user->Volume0 += (1.0 - _H[j][i])*h*h;
            }

            else if(user->counter > 0){
                user->VolumeNew += (1.0 - _H[j][i])*h*h;

                /* compute length */
                if( fabs(_phi[j][i]) <= 1.5*h){
                    delta = 0.5*(1 + cos(pi*_phi[j][i]/eps))/eps;
                }else if(fabs(_phi[j][i]) > 1.5*h){
                    delta = 0.0;
                }
                user->Length += delta*h*h;

            }
        }
    }

    if(user->counter > 0){
        /* compute dphi */
        user->dphi = (user->VolumeNew - user->Volume0) / user->Length;

        for (j=ys; j<ys+ym; j++) {
            for (i=xs; i<xs+xm; i++) {
                /* update phi */
                _phi[j][i] += user->dphi;
            }
        }

    }


    if(user->counter == 0){
        PetscPrintf(PETSC_COMM_WORLD,"Initial volume: %g\n",user->Volume0);
    }else if(user->counter > 0){
        printf("New volume: %g and Length: %g\n",user->VolumeNew, user->Length);
        printf("volume change: %g and dphi: %g\n",user->VolumeNew - user->Volume0,user->dphi);
    }

    ierr = DMDAVecRestoreArray(da,user->H,&_H);
    ierr = DMDAVecRestoreArray(da,user->phi,&_phi);


    if(user->counter > 0){
        FILE *volume, *length;
        volume =  fopen("volume.dat","a+");
        length =  fopen("length.dat","a+");
        fprintf(volume,"%g\t",user->VolumeNew);
        fprintf(length,"%g\t",user->Length);
        fclose(volume);
        fclose(length);
    }

    return(0);
}




// can be used only if matlab is configured
#undef __FUNCT__
#define __FUNCT__ "Screenshot"
PetscErrorCode Screenshot(UserContext *user,Engine *ep){

    PetscErrorCode ierr;

    /* velocity at n+1 */
    ierr = WriteVec(user->solnp1.U,"Unp1"); CHKERRQ(ierr);
    ierr = WriteVec(user->solnp1.V,"Vnp1"); CHKERRQ(ierr);
    ierr = WriteVec(user->solnp1.P,"Pnp1"); CHKERRQ(ierr);
    /* level set */
    ierr = WriteVec(user->phi_new,"phinp1"); CHKERRQ(ierr);

    char counter[16]; sprintf(counter,"counter = %4d;",user->counter);
    char time[16]; sprintf(time,"time = %.3f;",user->time);

    engEvalString(ep,counter);
    engEvalString(ep,time);

    if(user->counter == 0){
        engEvalString(ep,"cd Output;");
    }
    engEvalString(ep,"screenshot;");


    return(0);
}
```