A Front-tracking/Finite-Volume Navier-Stokes Solver for Direct Numerical Simulations of Multiphase Flows

Grétar Tryggvason

October 19, 2012

Contents

Chapter 1

Introduction

Multiphase flows are everywhere and understanding them is important for predicting the behavior of natural and industrial processes. Examples from Nature include rain and the gas and heat exchange between the atmosphere and the ocean, sandstorms and sediments and various aspects of volcanic eruptions. Boiling heat transfer and chemical processing in bubble columns are ubiquitous in power and chemical plants, the combustion of liquid fuel essentially always includes atomization and sprays are found in painting coating and cooling, irrigation, and a host of other applications.

Numerical simulations of fluid flows of increasing complexity are transforming our understanding of such systems and how we examine their behavior. Direct Numerical Simulations (DNS), where all continuum time and length scales are fully resolved, have had a particularly important impact. Such simulations have long been a standard tool for studies of turbulent single phase flows, but for multiphase flows, where two or more immiscible fluids separated by a sharp interface occupy the same domain, progress has been considerably slower. Nevertheless, such simulations are gradually becoming more common.

The purpose of this tutorial is to help you, the reader, get started doing direct numerical simulations (DNS) of multiphase flows. It is intended to be read from the beginning to the end and is structured in such a way that a computer code is introduced that gets more capable the further we go. This is the way a numerical code is often written and will, hopefully, ensure significant "ownership," or at least understanding, of the final product. I hope that you will find the tutorial easy to read and the concepts introduced relatively straight forward. Commensurate with my believe that the topic is simple—and that it is important to keep it as simple as possible—I have also tried to make the computer codes as readable as possible and it is my intention that they be read, just as the rest of the text.

This tutorial is not an introduction to either computations of multiphase flows in general nor DNS in particular. For a general introduction the former, the reader may want to consult

- A. Prosperetti and G. Tryggvason (editors and main contributors). Computational Methods for Multiphase Flow. Cambridge University Press, 2007. Paperback edition 2009. Reviewed in: J. Fluid Mech. 603 (2008), 474-475; Int'l. J. Multiphase Flow 34 (2008), 1096-1097.

For a fairly detailed treatment of DNS of multiphase flows, including both a description of numerical methods and a survey of results, we suggest

- G. Tryggvason, R. Scardovelli and S. Zaleski. Direct Numerical Simulations of Gas-Liquid Multiphase Flows. Cambridge University Press, 2011.

1.1 Computations

Computations of multiphase flows go back to the origin of computational fluid dynamics. The Marker and Cell (MAC) method developed at Los Alamos in the early sixties was designed specifically for free-surface and multiphase flows and used to simulate interfacial instabilities, gravity currents, waves and droplet collisions with walls, and other problems. The MAC method was followed by the Volume of Fluid (VOF) method, but although both methods produced impressive solutions, both were relatively inaccurate. In the late eighties and early nineties the development of other ideas, such as level sets and front-tracking methods, as well as improved VOF methods, marks the beginning of accurate and robust simulations of a variety of multiphase flows.

The majority of numerical methods currently used for detailed simulations of multiphase flows are based on the so-called one-field (or one-fluid) formulations of the Navier-Stokes equations, where one set of equations, describing the flow of all the fluids involved. The equations are usually solved on a regular structured grid, in most cases using a second order projection method where the solution is first updated without accounting for the pressure, the pressure is found from the divergence of the temporary solution, and the initial velocity is then corrected by adding the gradient of the pressure. The main difference between the various methods is how a marker function, identifying the different fluids, is updated.

1.2 Problem Specification

We will consider the unsteady motion of two or more incompressible and immiscible fluids where the location of the interface separating the different fluids will change with time. In some cases one fluid is immersed in the other as a disperse phase consisting of bubbles or drops but in other cases the interface is more complex and it is difficult to identify the disperse phase. And in many cases the phase layout undergoes topology changes where fluid masses coalesce or break up. Figure 1.1 shows the general situation where a convoluted interface separates two fluids but small bubbles or drops of either fluid can also be found fully immersed in the other fluid. For simplicity we will focus our attention on

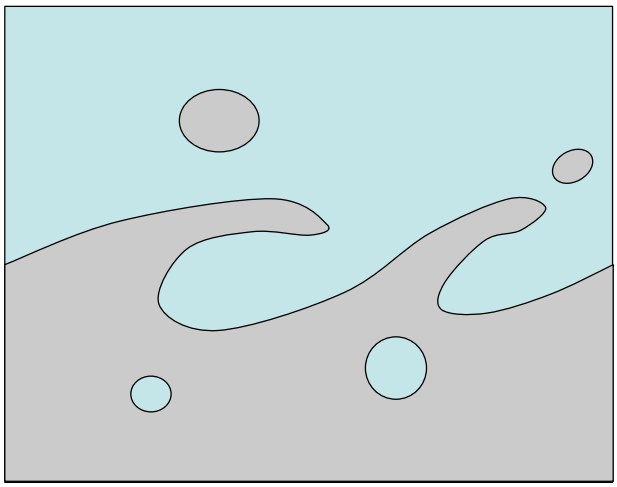

Figure 1.1: Multiphase flow. The different fluids can form a connected region or be dispersed in the other fluid as bubbles and drops.

disperse flows, often using the motion of only one bubble or a drop as a test case for our codes.

We generally define multiphase flows as two or more distinct phases or components flowing together. Thus, air bubbles and oil drops in water as well as vapor bubbles in liquids and fuel vapor and drops in sprays are multiphase flows. We could speak of multifluid flows when the fluids involved are distinct materials and reserve the multiphase to one fluid but different phases, but this is usually not done. The presence of two or more chemical species is not sufficient. Air, which is a mixture of several gases (such as oxygen, nitrogen, carbon-dioxide, and others) is generally not considered to be a multiphase flow. Similarly, water containing dissolved sugar or gases, is not multiphase flow. Here we will not consider miscible fluids, although often, particularly for short times, their evolution is very well described by standard approaches to describing multiphase flow. Often, multiphase flows are divided into Gas-liquid, Gas-solid and Three-phase flows. This is, however, somewhat incomplete since liquid-liquid (oil drops in water, for example) flows are often important and the difference between gas-liquid and liquid-liquid is simply the ratio of their properties. We shall thus simply distinguish between fluid-fluid and fluid-solid systems.

1.3 Governing Equations

The momentum equation for variable density and viscosity is:

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho\nabla\cdot\mathbf{uu} = -\nabla p + \rho\mathbf{g} + \nabla\cdot\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u}) + \mathbf{f}. \tag{1.1}$$

Here, $\mathbf{u}$ is the velocity, p the pressure, ρ is the density, and μ is the viscosity. $\mathbf{g}$ is the gravity acceleration and $\mathbf{f}$ is a body force other than gravity.

The density will generally change from one location to another, but we take the density of each fluid particle to remain constant, as the fluid particle moves with the flow. The density therefore changes according to

$$\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + \mathbf{u} \cdot \nabla \rho = 0, \tag{1.2}$$

where $D()/Dt$ is the convective derivative. For this case the conservation of mass equation for incompressible flow reduces to

$$\nabla \cdot \mathbf{u} = 0, \tag{1.3}$$

showing that volume is conserved. For the flow of two or more immiscible fluids, where one fluid is separated from the other by a sharp interface, the density is constant in each fluid, so that it can be reconstructed from the location of the interface, instead of solving equation (1.2) directly. Thus, the fluid dynamics problem consists of equations (1.1) and (1.3), with the understanding that the density must also be updated in some way.

In component form the Navier-Stokes equations, (1.1), for two-dimensional flows are:

$$\begin{aligned}
\rho\Big(\frac{\partial u}{\partial t} + \frac{\partial uu}{\partial x} + \frac{\partial uv}{\partial y}\Big) &= -\frac{\partial p}{\partial x} + \rho g_x + \frac{\partial}{\partial x} 2\mu \frac{\partial u}{\partial x} + \frac{\partial}{\partial y} \mu \Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big) + f_x \\
\rho\Big(\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial vv}{\partial y}\Big) &= -\frac{\partial p}{\partial y} + \rho g_y + \frac{\partial}{\partial x} \mu \Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big) + \frac{\partial}{\partial y} 2\mu \frac{\partial v}{\partial y} + f_y
\end{aligned} \tag{1.4}$$

and the continuity equation (1.3) is:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{1.5}$$

The advection terms can be written in several ways and it is easy to show that for incompressible flow:

$$\rho\Big(\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{uu}\Big) = \rho\Big(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}\Big) = \frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot \rho \mathbf{uu} \tag{1.6}$$

We will work mostly with the first form here.

It is also easily seen that for constant viscosity ($\mu = \mu_0$) the viscous terms can be written as:

$$\nabla \cdot \mu_o (\nabla \mathbf{u} + \nabla^T \mathbf{u}) = \mu_o \nabla^2 \mathbf{u}. \tag{1.7}$$

Or, for two-dimensional flow:

$$\begin{aligned}
\frac{\partial}{\partial x} 2\mu_0 \frac{\partial u}{\partial x} + \frac{\partial}{\partial y} \mu_0 \Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big) &= \mu_0 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\
\frac{\partial}{\partial x} \mu_0 \Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big) + \frac{\partial}{\partial y} 2\mu_0 \frac{\partial v}{\partial y} &= \mu_0 \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)
\end{aligned} \tag{1.8}$$

For a detailed discussion of the Navier-Stokes equations for multiphase flows, see the book by Tryggvason, Scardovelli and Zaleski.

1.4 Layout of the rest of the tutorial

In the next chapter we develop a very simple code for multiphase flow simulations, taking surface tension to be zero and the viscosities of both fluids to be the same. The density is first advected by a simple upwind method to allow us to present the fluid solver. Then we introduce front tracking to follow the interface motion (Chapter 3). We next introduce second order time integration, surface tension and viscosity that is different for the different fluids, in Chapter 4. This results in a relatively complete code that can be used to simulate a wide range of problems. To extend the code still further we introduce stretched grids and a better advection scheme in Chapter 5.

In the first few chapters we will write the codes in Matlab. Matlab is a particularly easy to use environment for code development and will allow us to write reasonably readable codes that are easily changed and experimented with. For more serious computations it is better, however, to switch to fortran or another advanced language.

Chapter 2

Simple Flow Solver

We start by describing a very simple Navier-Stokes solver for variable density flow. We will ignore surface tension, assume that the viscosities of both fluids are the same and use a first order time integration. To solve the density advection (equation 1.2), we will use a simple upwind method. By starting simple we will ensure that we always have a working code and this approach will also, hopefully, help the reader develop both an understanding of the code and good programming habits.

2.1 Governing Equations

The momentum equation, when surface tension is neglected, there is no body force except gravity, and the viscosity of both fluids is the same, is

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho\nabla\cdot\mathbf{uu} = -\nabla p + \rho\mathbf{g} + \mu_o\nabla^2\mathbf{u}. \tag{2.1}$$

We denote the viscosity by μ_o to emphasize that it is the same in both fluids. The conservation of mass equation for incompressible flow is unchanged and given by 1.3, which we repeat here for completeness

$$\nabla\cdot\mathbf{u} = 0 \tag{2.2}$$

2.2 Integration in Time

To integrate these equations in time, we split the momentum equation (equation 2.1) by first computing the velocity field without considering the pressure

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\mathbf{A}^n + \mathbf{g} + \frac{1}{\rho^n}\mathbf{D}^n \tag{2.3}$$

and then adding the pressure

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{\nabla_h p}{\rho^n}. \tag{2.4}$$

Here, the superscript n denotes a variable evaluated at the current time, t, and $n+1$ stands for a variable at the end of the time step (time $t+\Delta t$), $\mathbf{A}$ is a discrete approximation of the advection term, $\mathbf{D}$ is an approximation of the diffusion term, the subscript ∇_h denotes a discrete approximation of the gradient, and $\mathbf{u}^*$ is a temporary non-divergence-free velocity field. Adding equations (2.3) and (2.4) gives a numerical approximation to equation (2.1), where the time derivative has been approximated by a first order forward-in-time scheme.

The pressure must be determined in such a way that the final velocity field is divergence free at the end of the time step and satisfies the discrete version of equation (2.2):

$$\nabla_h \cdot \mathbf{u}^{n+1} = 0. \tag{2.5}$$

By taking the divergence of (2.4) and using (2.5) to eliminate $\mathbf{u}^{n+1}$, we get a Poisson equation for the pressure:

$$\nabla_h \cdot \left(\frac{1}{\rho^n} \nabla_h p \right) = \frac{1}{\Delta t} \nabla_h \cdot \mathbf{u}^*. \tag{2.6}$$

Once the pressure has been found, equation (2.4) can be used to find the projected velocity at time step $n+1$.

2.3 Spatial Discretization

To discretize the momentum equations, we use the Finite-Volume approach where the conservation principles of mass and momentum are applied to a small control volume, V. We define the average velocity in the control volume by:

$$\mathbf{u} = \frac{1}{V} \int_V \mathbf{u}(\mathbf{x}) dv. \tag{2.7}$$

To find numerical approximations for the advection and the diffusion terms in equation (2.3), we first define the average value over a control volume and then write the volume integral as a surface integral using the divergence theorem:

$$\mathbf{A} = \frac{1}{V} \int_V \nabla \cdot \mathbf{uu} \; dv = \frac{1}{V} \oint_S \mathbf{u}(\mathbf{u} \cdot \mathbf{n}) ds \tag{2.8}$$

and

$$\mathbf{D} = \frac{\mu_o}{V} \int_V \nabla^2 \mathbf{u} \; dv = \frac{\mu_o}{V} \int_V \nabla \cdot \nabla \mathbf{u} \; dv = \frac{\mu_o}{V} \oint_S \nabla \mathbf{u} \cdot \mathbf{n} \; ds. \tag{2.9}$$

The pressure term is approximated by the integral of the pressure gradient over a control volume:

$$\nabla_h p = \frac{1}{V} \int \nabla p \; dv = \frac{1}{V} \oint_S p\mathbf{n} \; ds. \tag{2.10}$$

So far, we have left the shape of the control volume unspecified. In principle we can use arbitrarily shaped control volumes but here we will work with structured Cartesian grids where the various variables are stored at points defined

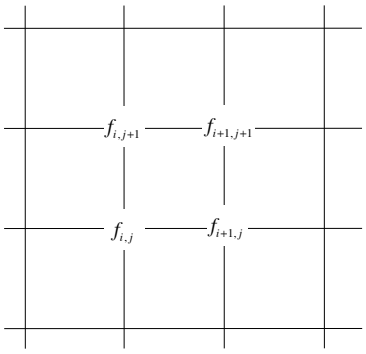

Figure 2.1: Regular structured grid

by the intersection of orthogonal grid lines. Figure 2.1 shows a sketch of such a grid. Regular structured grids have many benefits, the chief among them being that the connectivity of the grid points is implicit in the layout of the grid. There is no need to explicitly specify how the grid points are connected since the point to the right of point (i, j) is simply $(i + 1, j)$ and so on. This results in extremely efficient and relatively simple numerical codes. The orthogonality of the grid lines usually also helps the accuracy.

For the numerical code we are developing here, we divide the computational domain into rectangular equal sized control volumes. For incompressible flows, experience shows that it is simplest to use one control volume for the pressure and different control volumes for each of the velocity components. The motivation for doing so comes from considering the incompressibility conditions (equation 2.5). Integrating equation (2.5) over a control volume and converting the volume integral to a surface integral gives

$$\int_V \nabla \cdot \mathbf{u} dv = \oint_S \mathbf{u} \cdot \mathbf{n} ds = 0. \tag{2.11}$$

This shows that the inflow must be equal to the outflow. If the in- and outflow does not balance, then the pressure in the control volume must be increased or decreased to increase or decrease the flow in or out of the control volume.

Here we will take our control volumes to be aligned with the coordinate axis, where the horizontal axis is the x-direction and the vertical axis is the y-direction. The control volume for the pressure is shown in figure 2.2. In two-dimensions, the volume (actually the area, but we will speak of a volume to make the transition to three-dimensional flows easier) is given by $V = \Delta x \Delta y$, where Δx and Δy are the dimensions of the control volume in the x and y direction, respectively.

Approximating equation (2.5) by integrating over the edges of the control volume in figure 2.2, using the midpoint rule, yields:

$$\Delta y(u^{n+1}_{i+1/2,j} - u^{n+1}_{i-1/2,j}) + \Delta x(v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2}) = 0. \tag{2.12}$$

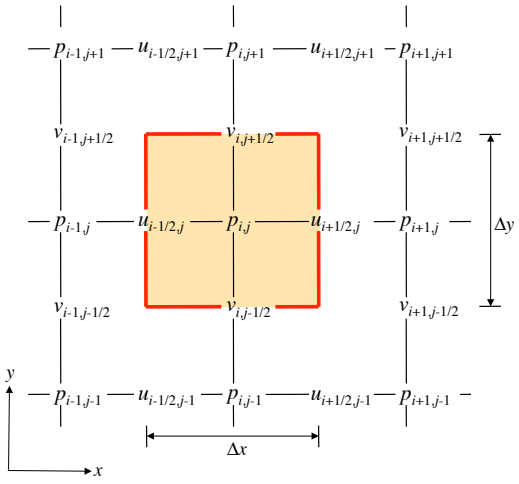

Figure 2.2: The notation used for a standard staggered mesh. The pressure is assumed to be known at the center of the control volume outlined by a thick solid line.

Here it is clear that we need the horizontal velocity components (u) at the vertical boundaries and the vertical velocity components (v) on the horizontal boundaries. Thus, it is natural to define new control volumes for each component, with the center of the control volume for the u-velocity component located at the middle of the vertical boundary of the pressure control volume and the control volume for the v-velocity component centered at the middle of the horizontal boundary of the pressure control volume. We can think of those control volumes as being displaced half a mesh to the right from the pressure node for the horizontal velocity and displaced half a mesh upward for the vertical velocity (see figure 2.3). It is customary to identify the pressure nodes by the indices (i, j) and to refer to the location of the u-velocity component by $(i+1/2, j)$ and the location of the v-velocity component by $(i, j+1/2)$. In an actual computer code the grids are, of course, simply shifted and each component referenced by an integer. This mesh is usually referred to as a staggered mesh or a grid. The density and other material properties are usually stored at the pressure nodes.

Using the grid in figures 2.2 and 2.3 along with the notation introduced above, the discrete approximations for the x and the y component of the predicted velocities (equation 2.3) are:

$$u^*_{i+1/2,j} = u^n_{i+1/2,j} + \Delta t \Bigg\{ \left(-A_x\right)^n_{i+1/2,j} + \left(g_x\right)^n_{i+1/2,j} + \frac{1}{\frac{1}{2}\left(\rho^n_{i+1,j} + \rho^n_{i,j}\right)} \left(D_x\right)^n_{i+1/2,j} \Bigg\} \tag{2.13}$$

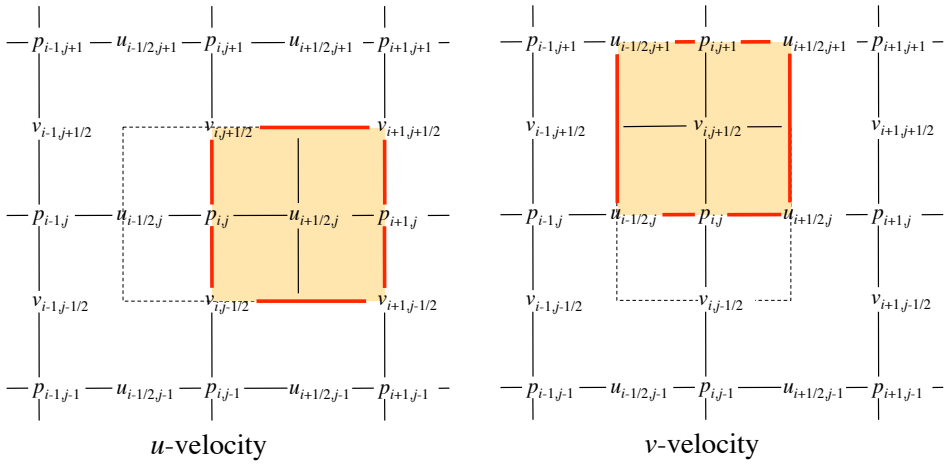

Figure 2.3: The notation used for a standard staggered mesh. The horizontal velocity components (u) are stored at the middle of the left and right edges of this control volume and the vertical velocity components (v) are stored at middle of the top and bottom edges.

and

$$v^*_{i,j+1/2} = v^n_{i,j+1/2} + \Delta t \Bigg\{ \left(-A_y\right)^n_{i,j+1/2} + \left(g_y\right)^n_{i,j+1/2} + \frac{1}{\frac{1}{2}(\rho^n_{i,j+1} + \rho^n_{i,j})} \left(D_y\right)^n_{i,j+1/2} \Bigg\}. \tag{2.14}$$

The equations for the correction velocities (equation 2.4) are:

$$u^{n+1}_{i+1/2,j} = u^*_{i+1/2,j} - \frac{\Delta t}{\frac{1}{2}(\rho^n_{i+1,j} + \rho^n_{i,j})} \frac{p_{i+1,j} - p_{i,j}}{\Delta x} \tag{2.15}$$

and

$$v^{n+1}_{i,j+1/2} = v^*_{i,j+1/2} - \frac{\Delta t}{\frac{1}{2}(\rho^n_{i,j+1} + \rho^n_{i,j})} \frac{p_{i,j+1} - p_{i,j}}{\Delta y}. \tag{2.16}$$

Some of the variables in these equations, such as the density, are needed at locations where they are not defined. For those values, we use linear interpolation (by taking the average) so that

$$\rho_{i+1/2,j} = \frac{1}{2}(\rho^n_{i+1,j} + \rho^n_{i,j}), \tag{2.17}$$

as so on. Assuming a linear variation also allows us to identify the value at the center of the control volume with the average value.

2.3.1 Discretization of the advection terms

For the advection terms (2.8), we first approximate the fluxes through each boundary by the value at the center of the boundary:

$$(A_x)_{i+1/2,j} = \frac{1}{\Delta x \Delta y}\Big\{\big[(uu)_{i+1,j} - (uu)_{i,j}\big]\Delta y + \big[(uv)_{i+1/2,j+1/2} - (uv)_{i+1/2,j-1/2}\big]\Delta x\Big\} \tag{2.18}$$

$$(A_y)_{i,j+1/2} = \frac{1}{\Delta x \Delta y}\Big\{\big[(uv)_{i+1/2,j+1/2} - (uv)_{i-1/2,j+1/2}\big]\Delta y + \big[(vv)_{i,j+1} - (vv)_{i,j}\big]\Delta x\Big\}. \tag{2.19}$$

We will start by using centered differencing for all spatial variables. Other alternatives can, of course, be used and those will be discussed later. In the centered, second-order scheme, the velocities at the boundaries of the velocity control volumes are found by linear interpolation and the momentum fluxes computed using these interpolated values. The x-component at $(i+1/2, j)$ is then

$$\begin{aligned}(A_x)^n_{i+1/2,j} = \frac{1}{\Delta x}\Bigg[&\Big(\frac{u^n_{i+3/2,j} + u^n_{i+1/2,j}}{2}\Big)^2 - \Big(\frac{u^n_{i+1/2,j} + u^n_{i-1/2,j}}{2}\Big)^2\Bigg] \\ &+\frac{1}{\Delta y}\Bigg[\Big(\frac{u^n_{i+1/2,j+1} + u^n_{i+1/2,j}}{2}\Big)\Big(\frac{v^n_{i+1,j+1/2} + v^n_{i,j+1/2}}{2}\Big) \\ &-\Big(\frac{u^n_{i+1/2,j} + u^n_{i+1/2,j-1}}{2}\Big)\Big(\frac{v^n_{i+1,j-1/2} + v^n_{i,j-1/2}}{2}\Big)\Bigg].\end{aligned} \tag{2.20}$$

The y-component, at the $(i, j+1/2)$ point, is:

$$\begin{aligned}(A_y)^n_{i,j+1/2} = \frac{1}{\Delta x}\Bigg[&\Big(\frac{u^n_{i+1/2,j} + u^n_{i+1/2,j+1}}{2}\Big)\Big(\frac{v^n_{i,j+1/2} + v^n_{i+1,j+1/2}}{2}\Big) \\ &-\Big(\frac{u^n_{i-1/2,j+1} + u^n_{i-1/2,j}}{2}\Big)\Big(\frac{v^n_{i,j+1/2} + v^n_{i-1,j+1/2}}{2}\Big)\Bigg] \\ &+\frac{1}{\Delta y}\Bigg[\Big(\frac{v^n_{i,j+3/2} + v^n_{i,j+1/2}}{2}\Big)^2 - \Big(\frac{v^n_{i,j+1/2} + v^n_{i,j-1/2}}{2}\Big)^2\Bigg].\end{aligned} \tag{2.21}$$

The centered second-order scheme is more accurate than any non-centered second-order scheme and usually gives the best results for fully resolved flows. It has, however, two serious shortcomings. The first problem is that for flows that are not fully resolved it can produce unphysical oscillations that can degrade the quality of the results. The second problem is that the centered second-order

scheme is unconditionally unstable for inviscid flows when used in combination with the explicit forward-in-time integration given by equations (2.3) and (2.4). It is only the addition of the viscosity terms that makes the scheme stable and if the viscosity is small, the time-step must be small. At high Reynolds numbers this usually results in excessively small time steps.

2.3.2 Discretization of the diffusion terms

Approximating the integral of the viscous fluxes around the boundaries of the velocity control volumes (equation 2.9) by the value at the midpoint of each edge times the length of the edge results in:

$$(D_x)^n_{i+1/2,j} = \frac{\mu_o}{\Delta x \Delta y}\Bigg\{\Bigg(\Big(\frac{\partial u}{\partial x}\Big)_{i+1,j} - \Big(\frac{\partial u}{\partial x}\Big)_{i,j}\Bigg)\Delta y + \Bigg(\Big(\frac{\partial u}{\partial y}\Big)_{i+1/2,j+1/2} - \Big(\frac{\partial u}{\partial y}\Big)_{i+1/2,j-1/2}\Bigg)\Delta x\Bigg\} \tag{2.22}$$

and

$$(D_y)^n_{i,j+1/2} = \frac{\mu_o}{\Delta x \Delta y}\Bigg\{\Bigg(\Big(\frac{\partial v}{\partial x}\Big)_{i+1/2,j+1/2} - \mu\Big(\frac{\partial v}{\partial x}\Big)_{i-1/2,j+1/2}\Bigg)\Delta y + \Bigg(\Big(\frac{\partial v}{\partial y}\Big)_{i,j+1} - \Big(\frac{\partial v}{\partial y}\Big)_{i,j}\Bigg)\Delta x\Bigg\}. \tag{2.23}$$

The velocity derivatives are found using the standard second-order centered differences, resulting in:

$$(D_x)^n_{i+1/2,j} = \mu_o\Bigg\{\Big(\frac{u^n_{i+3/2,j} - 2u^n_{i+1/2,j} + u^n_{i-1/2,j}}{\Delta x^2}\Big) + \Big(\frac{u^n_{i+1/2,j+1} - 2u^n_{i+1/2,j} + u^n_{i+1/2,j-1}}{\Delta y^2}\Big)\Bigg\} \tag{2.24}$$

and

$$(D_y)^n_{i,j+1/2} = \mu_o\Bigg\{\Big(\frac{v^n_{i+1,j+1/2} - 2v^n_{i,j+1/2} + v^n_{i-1,j+1/2}}{\Delta x^2}\Big) + \Big(\frac{v^n_{i,j+3/2} - 2v^n_{i,j+1/2} + v^n_{i,j-1/2}}{\Delta y^2}\Big)\Bigg\}. \tag{2.25}$$

It is worth emphasizing again that this relatively simple form of the viscous terms only holds for the special cases of two incompressible fluids with the same viscosities. For the more general case, where the different fluids have different viscosities, the full deformation tensor must be used, resulting in more complex forms. This will be done in Chapter 4.

2.4 The Pressure Equation

The pressure equation is derived by substituting the expression for the corrected velocity, equations (2.15) and (2.16), into the discrete mass conservation equation (2.12). The result is:

$$\frac{1}{\Delta x^2}\left(\frac{p_{i+1,j}-p_{i,j}}{\rho^n_{i+1,j}+\rho^n_{i,j}}-\frac{p_{i,j}-p_{i-1,j}}{\rho^n_{i,j}+\rho^n_{i-1,j}}\right)+\frac{1}{\Delta y^2}\left(\frac{p_{i,j+1}-p_{i,j}}{\rho^n_{i,j+1}+\rho^n_{i,j}}-\frac{p_{i,j}-p_{i,j-1}}{\rho^n_{i,j}+\rho^n_{i,j-1}}\right)$$
$$=\frac{1}{2\Delta t}\Big(\frac{u^*_{i+1/2,j}-u^*_{i-1/2,j}}{\Delta x}+\frac{v^*_{i,j+1/2}-v^*_{i,j-1/2}}{\Delta y}\Big). \quad (2.26)$$

Solving the pressure equation is usually the most expensive part of any simulation of incompressible flows and usually it is necessary to use advanced pressure solver to achieve reasonable computational times. Here, however, we will use a simple Successive Over Relaxation (SOR) method. To do so, we rearrange equation (2.26) and isolate $p_{i,j}$ on the left hand side. The pressure is then updated iteratively by substituting for pressure values on the right-hand side the approximate values $p^\alpha_{i,j}$ of the pressure from the previous iteration. In the SOR method we also take a weighted average of the updated value and the pressure $p^\alpha_{i,j}$ from the last iteration to compute the new approximation $p^{\alpha+1}_{i,j}$:

$$p^{\alpha+1}_{i,j}=$$
$$\beta\left[\frac{1}{\Delta x^2}\Big(\frac{1}{\rho^n_{i+1,j}+\rho^n_{i,j}}+\frac{1}{\rho^n_{i,j}+\rho^n_{i-1,j}}\Big)+\frac{1}{\Delta y^2}\Big(\frac{1}{\rho^n_{i,j+1}+\rho^n_{i,j}}+\frac{1}{\rho^n_{i,j}+\rho^n_{i,j-1}}\Big)\right]^{-1}$$
$$\left[\frac{1}{\Delta x^2}\Big(\frac{p^\alpha_{i+1,j}}{\rho^n_{i+1,j}+\rho^n_{i,j}}+\frac{p^{\alpha+1}_{i-1,j}}{\rho^n_{i,j}+\rho^n_{i-1,j}}\Big)+\frac{1}{\Delta y^2}\Big(\frac{p^\alpha_{i,j+1}}{\rho^n_{i,j+1}+\rho^n_{i,j}}+\frac{p^{\alpha+1}_{i,j-1}}{\rho^n_{i,j}+\rho^n_{i,j-1}}\Big)\right.$$
$$\left.-\frac{1}{2\Delta t}\Big(\frac{u^*_{i+1/2,j}-u^*_{i-1/2,j}}{\Delta x}+\frac{v^*_{i,j+1/2}-v^*_{i,j-1/2}}{\Delta y}\Big)\right]+(1-\beta)p^\alpha_{i,j}. \quad (2.27)$$

Here, we assume that the iteration is done by sweeping over the rows and columns so that when we update grid point (i,j), the value of p at grid points $(i-1,j)$ and $(i,j-1)$ have already been updated. β is a relaxation parameter and for over-relaxation we must have $\beta > 1$. For stability reasons we must also have $\beta < 2$. Taking $\beta = 1.2-1.5$ is usually a good compromise between stability and accelerated convergence. To assess how well the iteration has converged, we monitor the maximum difference between the pressure at successive iterations and stop, once this difference is small enough. A more sophisticated approach would be to monitor the *residual*, that is the difference between the left and the right hand side of equation (2.26), but this generally requires extra computational effort. While the SOR method works well for our purpose, for serious computations more advanced pressure solvers are generally used.

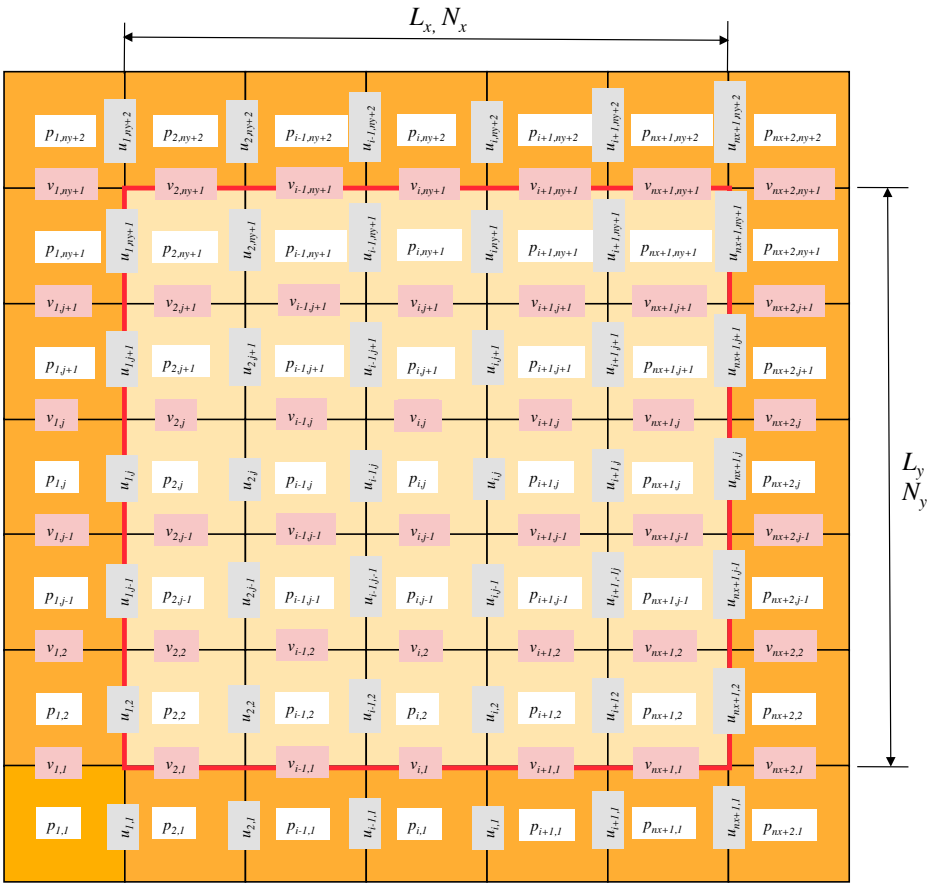

Figure 2.4: The notation used for a standard staggered MAC mesh. The pressure is assumed to be known at the center of the control volume outlined by a thick solid line. The horizontal velocity components (u) are stored at the middle of the left and right edges of this control volume and the vertical velocity components (v) are stored at middle of the top and bottom edges.

2.5 The Computational Domain

Here we take the computational domain to be a rectangle of size L_x by L_y, divided into N_x by N_y pressure control volumes. The pressure control volumes are placed inside the computational domain, such that the boundaries of the pressure control volumes at the edge of the domain coincide with the domain boundary. In addition to the control volumes inside the domain we usually provide one row of ghost cells outside the domain to help with implementing boundary conditions. Thus, the pressure array is dimensioned $p(N_x+2, N_y+2)$. Similarly, we will need ghost points for the tangential velocity, but not the normal velocity, which is given. The velocity arrays thus have dimensions: $u(N_x+1, N_y+2)$ and $v(N_x+2, N_y+1)$ and the pressure array is $p(N_x+2, N_y+2)$. Figure 2.4 shows the layout of the full grid.

2.6 Boundary Conditions

Before attempting to evolve the solution using the discrete equations that we have just derived, we must establish the appropriate boundary conditions. The use of a staggered mesh makes the derivation of the appropriate boundary conditions particularly straightforward. For the normal velocities the specification of the boundary values is very simple. Since the location of the center of the control volume coincides with the boundary, we can set the velocity equal to what it should be. For a rigid wall the normal velocity is usually zero and for inflow boundaries the normal velocity is generally given. For viscous flows we must also specify the tangential velocity and this is slightly more complex, since we do not store the velocity on the boundary. We do, however, have the velocity half a grid spacing inside the flow and using this value, along with the known value on the boundary, we can specify the "ghost" value at the center of the ghost cell outside the boundary. To do so we assume that we know the ghost value and that the wall velocity is given by a linear interpolation between the ghost velocity and the velocity just inside the domain. For the u-velocity on the bottom boundary, for example:

$$u_{wall} = (1/2)(u_{i,1} + u_{i,2}) \tag{2.28}$$

where u_{wall} is the tangent velocity on the wall and $u_{i,1}$ is the ghost velocity. Since the wall velocity and the velocity inside the domain, $u_{i,2}$, are known, we can easily find the ghost velocity

$$u_{i,1} = 2\ u_{wall} - u_{i,2}. \tag{2.29}$$

Similar equations are derived for the other boundaries. Notice that if the wall velocity is zero, the ghost velocity is simply a reflection of the velocity inside.

The boundary condition for the pressure become particularly simple on a staggered grid. Consider the vertical boundary in figure 2.5, on the left side of the domain and passing through node $(i-1/2, j)$, where the horizontal velocity

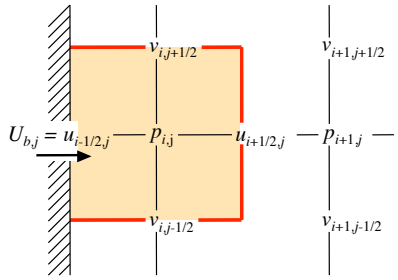

Figure 2.5: A control volume next to a boundary where the normal velocity is known

is $U_{b,j}$. The continuity equation for the cell next to the boundary, surrounding the pressure node (i, j), is

$$\frac{u^{n+1}_{i+1/2,j} - U_{b,j}}{\Delta x} + \frac{v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2}}{\Delta y} = 0. \tag{2.30}$$

Since the velocity at the left boundary is known, we only substitute the equations for the correction velocities, (2.15) and (2.16), for the three unknown velocities, through the top, bottom and right edge. Thus, the pressure equation for a point next to the left boundary becomes:

$$\frac{1}{\Delta x^2}\left(\frac{p_{i+1,j} - p_{i,j}}{\rho^n_{i+1,j} + \rho^n_{i,j}}\right) + \frac{1}{\Delta y^2}\left(\frac{p_{i,j+1} - p_{i,j}}{\rho^n_{i,j+1} + \rho^n_{i,j}} - \frac{p_{i,j} - p_{i,j-1}}{\rho^n_{i,j} + \rho^n_{i,j-1}}\right)$$
$$= \frac{1}{2\Delta t}\left(\frac{u^*_{i+1/2,j} - U_{b,j}}{\Delta x} + \frac{v^*_{i,j+1/2} - v^*_{i,j-1/2}}{\Delta y}\right). \tag{2.31}$$

Similar equations are derived for the pressure next to the other boundaries and for each corner point. Notice that it is not necessary to impose any new conditions on the pressure at the boundaries and that simply using incompressibility yields the correct boundary equations.

In principle we need separate equations for the points near the boundary, but we can get around this if we set $\rho^n_{i-1,j}$ equal to a very large number at these points (and keep the value of the ghost pressure to a finite number, such as zero) so that the term involving the pressure in the ghost cell becomes approximately zero. We shall use this trick in the simple code presented below.

2.7 Numerical Algorithm

The discrete equations written down above are solved in a sequence designed to make sure all variables are available when needed. Assuming that the initial flow field and density are given the algorithm is:

1. Find a temporary velocity field using equations (2.13) and (2.14) , with the advection and diffusion terms discretized using equations (2.20) and (2.21) and (2.24) and (2.25), respectively.

2. Find the pressure using equation (2.26), with the appropriate boundary conditions (2.31).

3. Correct the velocity field using equation (2.15) and (2.16).

4. Advect the density field

The code is stable if the time step is small enough. The step size is limited by the diffusion part alone and by a combined criteria that ensures that diffusion is large enough to keep the advection stable. For two-dimensional flow we must have:

$$\frac{\mu \Delta t}{\rho h^2} \leq \frac{1}{4} \qquad (\mathbf{u} \cdot \mathbf{u}) \frac{\rho \Delta t}{\mu} \leq 2. \tag{2.32}$$

Here, h is the smallest grid spacing (Δx or Δy) and $\mathbf{u}$ is the maximum velocity.

So far, we have not said anything about how the density field is advected. This can be done in several different ways and we will start by using a simple upwind approximation for the advection to solve equation (1.2). The simple upwind method is too diffusive to be of any real use in simulations of multiphase flows but allows us to introduce a working code before describing front tracking to advance the fluid interface.

2.8 Advecting the Density: Centered Differences with Diffusion

To advect the density we use equation (1.2), rewritten to isolate the time derivative:

$$\frac{\partial \rho}{\partial t} = -\mathbf{u} \cdot \nabla \rho. \tag{2.33}$$

For numerical implementation it is usually better to take advantage of the fact that since the flow field is incompressible (by equation 1.3), we can write equation (2.33) as

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{u}). \tag{2.34}$$

This is, of course, the mass conservation equation in its original form and working with this form allows us to ensure that mass is conserved exactly. Accurately advocating the density is oner of the major challenges in simulations of multiphase flows, but here we will use a simple centered difference scheme. The scheme is unstable in the absence of diffusion so we will add a diffusion term and use the viscosity of the fluid as a diffusion coefficient so that the stability limits are the same as for the fluids. In principle we should use the kinematic

viscosity, but since we will take the density to be of order one, either one will work. Thus, we modify equation (2.35) by writing:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{u}) + \mu_0 \nabla^2 \rho. \tag{2.35}$$

and use the discretization

$$\begin{aligned} \rho_{i,j}^{n+1} = \rho_{i,j}^{n} &- \frac{\Delta t}{\Delta x}\Big(u_{i+1/2,j}\frac{\rho_{i+1,j}+\rho_{i,j}}{2} - u_{i-1/2,j}\frac{\rho_{i-1,j}+\rho_{i,j}}{2}\Big) \\ &- \frac{\Delta t}{\Delta y}\Big(v_{i,j+1/2}\frac{\rho_{i,j+1}+\rho_{i,j}}{2} - u_{i,j-1/2}\frac{\rho_{i,j-1}+\rho_{i,j}}{2}\Big) \\ &+ \frac{\Delta t}{\Delta x^2}(\rho_{i+1,j} - 2\rho_{i,j} + \rho_{i-1,j}) + \frac{\Delta t}{\Delta y^2}(\rho_{i,j+1} - 2\rho_{i,j} + \rho_{i,j-1}) \end{aligned} \tag{2.36}$$

This is way to inaccurate for any serious computational work and is done here just so that we can test the code.

2.9 Numerical Code (MATLAB)

The code below, written in MATLAB shows an implementation of the algorithm described above, using the first order upwind method to advect the density. The code is written to simulate the motion of a drop initially located in the middle of a square domain. As gravity accelerates the drop downward, it becomes flatter and the outer rim is pulled back by the flow. This code can be downloaded from: code1.m

2.10 Results

Figure 2.6 shows the initial conditions and the final results from the code listed below, and figure 2.7 shows a three-dimensional view of the density profile at the final time (plotted using mesh(r)).

Often, we want to generate a movie of the results. In Matlab there are several ways to do that, but one of the simplest way is to insert the following line into the program, after the plot statement:

```
eval(['print -dpict ','temp',num2str(is)]);pause(0.01);
```

This prints out each frame as a pict file. The files can then be assembled into a movie.

2.11 Exercises

1. Run the code up to a later time, for the three different grid resolutions (adjust the time step to keep it stable, if needed) and compare the results. You can also compute the location of the center of mass versus time and see how it depends on the resolution.

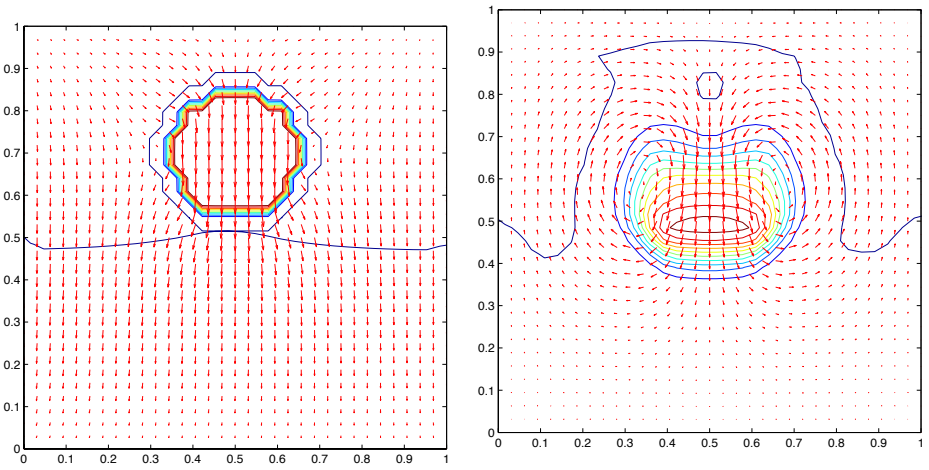

Figure 2.6: The density and the velocity field at time zero and after the drop has fallen a short distance.

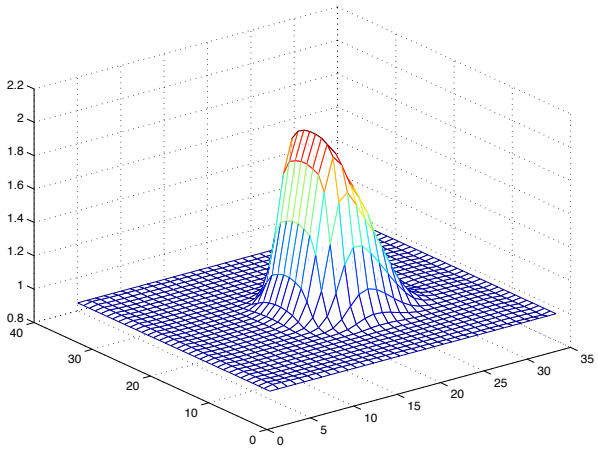

Figure 2.7: The density after the drop has fallen a short distance.

2. Rayleigh-Taylor Instability: Change the code to follow the evolution of a heavy fluid overlaying a lighter one. Make the side boundaries full slip, by setting the ghost tangential velocity equal to the velocity inside the domain, and change the initial condition to an interface perturbed by a cos wave of amplitude 0.1 times the domain width. You can the change the density difference and the viscosity to examine what effect these parameters have.
3. Change the discretization of the advection terms to the convective form and compare the results with predictions from the code as it is. For the relatively benign parameters used in the code the change should not make any noticeable difference.
4. Make the boundary conditions periodic in the vertical directions, change the domain to a square box, and follow the motion of a fluid blob as it crosses the boundary. Notice that the fluid starts to fall because of gravity and eventually we will have something like a parabolic velocity between the vertical walls.

```
%=========================================================================
% code1.m
% A very simple Navier-Stokes solver for a drop falling in a rectangular
% domain. The viscosity is taken to be a constant and a forward in time,
% centered in space discretization is used. The density is advected by a
% simple upwind scheme.
%=========================================================================
%domain size and physical variables
Lx=1.0;Ly=1.0;gx=0.0;gy=-100.0; rho1=1.0; rho2=2.0; m0=0.01; rro=rho1;
unorth=0;usouth=0;veast=0;vwest=0;time=0.0;
rad=0.15;xc=0.5;yc=0.7; % Initial drop size and location

% Numerical variables
nx=32;ny=32;dt=0.00125;nstep=100; maxit=200;maxError=0.001;beta=1.2;

% Zero various arrys
u=zeros(nx+1,ny+2); v=zeros(nx+2,ny+1); p=zeros(nx+2,ny+2);
ut=zeros(nx+1,ny+2); vt=zeros(nx+2,ny+1); tmp1=zeros(nx+2,ny+2);
uu=zeros(nx+1,ny+1); vv=zeros(nx+1,ny+1); tmp2=zeros(nx+2,ny+2);

% Set the grid
dx=Lx/nx;dy=Ly/ny;
for i=1:nx+2; x(i)=dx*(i-1.5);end; for j=1:ny+2; y(j)=dy*(j-1.5);end;

% Set density
r=zeros(nx+2,ny+2)+rho1;
for i=2:nx+1,for j=2:ny+1;
  if ( (x(i)-xc)^2+(y(j)-yc)^2 < rad^2), r(i,j)=rho2;end,
end,end
%================== START TIME LOOP====================================
 for is=1:nstep,is
    % tangential velocity at boundaries
    u(1:nx+1,1)=2*usouth-u(1:nx+1,2);u(1:nx+1,ny+2)=2*unorth-u(1:nx+1,ny+1);
    v(1,1:ny+1)=2*vwest-v(2,1:ny+1);v(nx+2,1:ny+1)=2*veast-v(nx+1,1:ny+1);

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity
      ut(i,j)=u(i,j)+dt*(-0.25*(((u(i+1,j)+u(i,j))^2-(u(i,j)+   ...
          u(i-1,j))^2)/dx+((u(i,j+1)+u(i,j))*(v(i+1,j)+         ...
          v(i,j))-(u(i,j)+u(i,j-1))*(v(i+1,j-1)+v(i,j-1)))/dy)+ ...
          m0/(0.5*(r(i+1,j)+r(i,j)))*(                          ...
                 (u(i+1,j)-2*u(i,j)+u(i-1,j))/dx^2+             ...
                 (u(i,j+1)-2*u(i,j)+u(i,j-1))/dy^2 )+gx    );
    end,end

    for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity
      vt(i,j)=v(i,j)+dt*(-0.25*(((u(i,j+1)+u(i,j))*(v(i+1,j)+   ...
          v(i,j))-(u(i-1,j+1)+u(i-1,j))*(v(i,j)+v(i-1,j)))/dx+  ...
          ((v(i,j+1)+v(i,j))^2-(v(i,j)+v(i,j-1))^2)/dy)+        ...
          m0/(0.5*(r(i,j+1)+r(i,j)))*(                          ...
                 (v(i+1,j)-2*v(i,j)+v(i-1,j))/dx^2+             ...
                 (v(i,j+1)-2*v(i,j)+v(i,j-1))/dy^2 )+gy    );
    end,end
%=========================================================================
    % Compute source term and the coefficient for p(i,j)
    rt=r; lrg=1000;
    rt(1:nx+2,1)=lrg;rt(1:nx+2,ny+2)=lrg;
    rt(1,1:ny+2)=lrg;rt(nx+2,1:ny+2)=lrg;
```

```
    for i=2:nx+1,for j=2:ny+1
        tmp1(i,j)= (0.5/dt)*( (ut(i,j)-ut(i-1,j))/dx+(vt(i,j)-vt(i,j-1))/dy );
        tmp2(i,j)=1.0/( (1./dx)*( 1./(dx*(rt(i+1,j)+rt(i,j)))+...
                                  1./(dx*(rt(i-1,j)+rt(i,j)))  )+...
                        (1./dy)*(1./(dy*(rt(i,j+1)+rt(i,j)))+...
                                  1./(dy*(rt(i,j-1)+rt(i,j)))   )   );
    end,end

    for it=1:maxit                % SOLVE FOR PRESSURE
      oldArray=p;
      for i=2:nx+1,for j=2:ny+1
          p(i,j)=(1.0-beta)*p(i,j)+beta* tmp2(i,j)*(...
          (1./dx)*( p(i+1,j)/(dx*(rt(i+1,j)+rt(i,j)))+...
                    p(i-1,j)/(dx*(rt(i-1,j)+rt(i,j))) )+...
          (1./dy)*( p(i,j+1)/(dy*(rt(i,j+1)+rt(i,j)))+...
                    p(i,j-1)/(dy*(rt(i,j-1)+rt(i,j))) ) - tmp1(i,j));
      end,end
      if max(max(abs(oldArray-p))) <maxError, break,end
    end

    for i=2:nx,for j=2:ny+1   % CORRECT THE u-velocity
          u(i,j)=ut(i,j)-dt*(2.0/dx)*(p(i+1,j)-p(i,j))/(r(i+1,j)+r(i,j));
    end,end

    for i=2:nx+1,for j=2:ny   % CORRECT THE v-velocity
          v(i,j)=vt(i,j)-dt*(2.0/dy)*(p(i,j+1)-p(i,j))/(r(i,j+1)+r(i,j));
    end,end
%=======ADVECT DENSITY using centered difference plus diffusion ==========
    ro=r;
    for i=2:nx+1,for j=2:ny+1
          r(i,j)=ro(i,j)-(0.5*dt/dx)*(u(i,j)*(ro(i+1,j)...
                 +ro(i,j))-u(i-1,j)*(ro(i-1,j)+ro(i,j)) )...
                       -(0.5* dt/dy)*(v(i,j)*(ro(i,j+1)...
                 +ro(i,j))-v(i,j-1)*(ro(i,j-1)+ro(i,j))  )...
                 +(m0*dt/dx/dx)*(ro(i+1,j)-2.0*ro(i,j)+ro(i-1,j))...
                 +(m0*dt/dy/dy)*(ro(i,j+1)-2.0*ro(i,j)+ro(i,j-1));
   end,end
%=========================================================================
   time=time+dt                   % plot the results
   uu(1:nx+1,1:ny+1)=0.5*(u(1:nx+1,2:ny+2)+u(1:nx+1,1:ny+1));
   vv(1:nx+1,1:ny+1)=0.5*(v(2:nx+2,1:ny+1)+v(1:nx+1,1:ny+1));
   for i=1:nx+1,xh(i)=dx*(i-1);end;     for j=1:ny+1,yh(j)=dy*(j-1);end
   hold off,contour(x,y,flipud(rot90(r))),axis equal,axis([0 Lx 0 Ly]);
   hold on;quiver(xh,yh,flipud(rot90(uu)),flipud(rot90(vv)),'r');
   pause(0.01)
end
```

Chapter 3

Advecting the Density in 2D using Front Tracking

Instead of advecting the density directly, by solving equation (1.2), we can move the interface between the different fluids and reconstruct the density from its location. This is usually called "front tracking." Here we describe a very simple implementation of front tracking and show how to modify the code introduced in the last chapter to update the density in that way. Unlike when we advect the density directly, using front tracking is a two step process: we move the boundary and then we construct the density field. In some cases we define an indicator function that is used to set the density and other material properties but here we use the density as the indicator function.

First, however, we have to decide how we represent the interface. Generally the interfaces must be dynamically restructured by adding and deleting points as the interface deforms. In three dimension the correct data structure can be critical for the successful implementation of the method. For two-dimensional flow, on the other hand, essentially any data structure can be made to work rather easily. Here we will use a simple ordered array of points where we represent the interface by markers whose coordinates are given by:

$$\mathbf{x}_f(l) = \big(x(l), y(l)\big), l = 1, ..., N_f. \tag{3.1}$$

Since the points are ordered so that point l comes after point $l-1$ and before point $l+1$, no additional information about their connectivity is needed. This makes it particularly simple to compute, for example, the distance between two points

$$\Delta s_{l,l-1} = \sqrt{(x(l) - x(l-1))^2 + (y(l) - y(l-1))^2}. \tag{3.2}$$

The front is shown schematically in figure 3.1. In the code presented below we will carry two extra points that are identical to the first and the second point so that $x(N_f + 1) = x(1)$ and $x(N_f + 2) = x(1)$ and so on. We will work with the points $l = 2 \rightarrow N_f + 1$ and use points 1 and $N_f + 2$ as ghost points to simplify operations on the front.

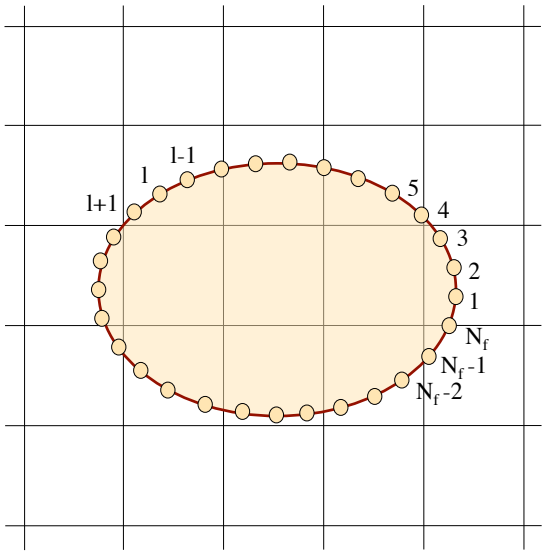

Figure 3.1: The front used to mark the interface between two fluids.

3.1 Moving the front

To move the marker points we interpolate their velocities from the velocity field given on the fixed fluid grid. Similarly, quantities such as the density jump and the surface tension are smoothed from the front onto the fixed grid. The interpolation and smoothing can be done in different ways but here we will use a bilinear interpolation and once we have determined the weights, we will use the same ones for the smoothing. Before communicating between the front and the fixed grid, we need to identify the fixed grid points closest to a given point on the front. For a rectangular domain of length L_x, divided into N_x equal sized control volumes, with the grid point where $i = 1$ corresponding to $x = 0$, the grid point to the left of front point x_f is given by:

$$i = \text{FLOOR}(x_f(l)/\Delta x) + 1 \tag{3.3}$$

where FLOOR stands for an operation rounding its operand *down* to its closest integer value. While this expression holds for the u-velocities on the grid shown in figure 2.4, it must be modified slightly for the v-velocities where the grid is displace half a grid spacing to the left. Then we must use

$$i = \text{FLOOR}((x_f(l) + 0.5\Delta x)/\Delta x) + 1. \tag{3.4}$$

For the j-direction we swap those. Figure 3.2 show the layout of the grid points along the x-axis and the effect of shifting the grid.

These expressions are also used when we smooth front quantities onto the velocity grid-points. Notice that while it is trivial to find which fixed grid points are close to a given front point, the reverse problem is much harder. Usually it is possible to arrange the code in such a way that we need to find fixed grid points close to a given front point, rather than the other way around.

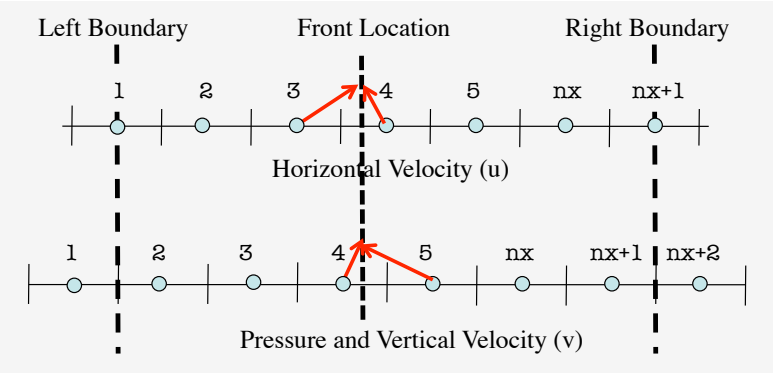

Figure 3.2: Finding which grid point is closest to a given front point. The red arrows show which grid points the front communicates with for for velocity and pressure.

After the fixed grid points that are closest to a given location on the front have been located, we can interpolate values on the fixed grid to the front. If we use a bilinear interpolation, then the front value ϕ_f^l is given by:

$$\phi_f^l = \phi_{i,j}\Big(\frac{x_{i+1}-x_f}{\Delta x}\Big)\Big(\frac{y_{j+1}-y_f}{\Delta y}\Big) + \phi_{i,j+1}\Big(\frac{x_{i+1}-x_f}{\Delta x}\Big)\Big(\frac{y_f-y_j}{\Delta y}\Big) + \phi_{i+1,j}\Big(\frac{x_f-x_i}{\Delta x}\Big)\Big(\frac{y_{j+1}-y_f}{\Delta y}\Big) + \phi_{i+1,j+1}\Big(\frac{x_f-x_i}{\Delta x}\Big)\Big(\frac{y_f-y_j}{\Delta y}\Big). \tag{3.5}$$

Here x_i is the x-coordinate of the vertical grid line to the left of the front point and y_j is the y-coordinate of the horizontal grid line below the front point, both found using equation (3.5). $\phi_{i,j}$ is the value of ϕ at the fixed grid point to the left and below the front point. See figure 3.3. The weights can be interpreted as the area fractions, as shown in the figure. Bilinear interpolation is therefore often referred to as area-weighting.

With the coefficients in equation (3.5) interpreted as weights, we can rewrite it as

$$\phi_f^l = w_{i,j}^l\phi_{i,j} + w_{i,j+1}^l\phi_{i,j+1} + w_{i+1,j}^l\phi_{i+1,j} + w_{i+1,j+1}^l\phi_{i+1,j+1}. \tag{3.6}$$

Other interpolation strategies are possible and in general, the interpolation is given by

$$\phi_f^l = \sum_{ij} w_{i,j}^l\phi_{i,j}, \tag{3.7}$$

where ϕ_f^l is a quantity on the front at location l. $\phi_{i,j}$ is the same quantity on the grid, $w_{i,j}^l$ is the weight of each grid point with respect to location l, and the summation is over the points on the fixed grid that are close to the front point. The weights can be selected in many different ways, but must always sum up to one:

$$\sum_{i,j} w_{i,j}^l = 1. \tag{3.8}$$

The bilinear interpolation given by equation (3.5) generally works very well and we will use it for most of the codes described in this tutorial.

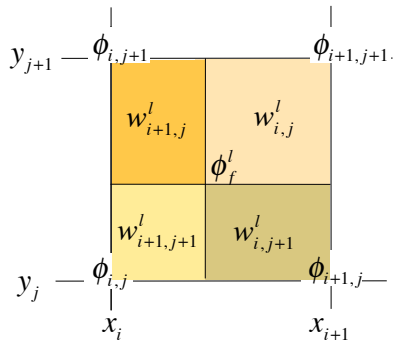

Figure 3.3: The interpretation of the weights in equation 3.5 as area fractions.

For the velocity, we apply equation (3.5) to each velocity component. For staggered grids, as we are using here, usually we interpolate each component separately, since they are given on different fixed grids. Once the velocity of the front ($\mathbf{u}_f^n$) has been interpolated from the grid, the front can be moved. If we use a simple explicit first-order time integration, then the new front location is given by

$$\mathbf{x}_f^{n+1} = \mathbf{x}_f^n + \Delta t \mathbf{u}_f^n \tag{3.9}$$

3.2 Constructing the density field

To find the density, given the location of the interface, we use the fact that the front marks the jump in the density. In terms of a Heaviside function defined by

$$H = \begin{cases} 1 & \text{fluid 1} \\ 0 & \text{fluid 2} \end{cases} \tag{3.10}$$

the density can be written as

$$\rho = \rho_1 H + (1 - H)\rho_2. \tag{3.11}$$

On the fixed grid the sharp jump is translated into a steep gradient on the fixed grid. The gradient of the density can be related to the jump by writing it as:

$$\nabla\rho = (\rho_1 - \rho_2)\nabla H = \Delta\rho \mathbf{n}\delta(n). \tag{3.12}$$

where $\Delta\rho = (\rho_1 - \rho_2)$ is the jump in the value of the density across the interface (usually a constant for each interface) and we have used that $\nabla H = \delta(n)\mathbf{n}$. Here, n is a coordinate normal to the interface. Since the front represents a δ-function, the transfer corresponds to the construction of an approximation to this δ-function on the fixed grid. This smoothing can be done in several different ways, but in many cases the transferred quantity is conserved and it is important to preserve the integrated value when moving from the front to the fixed grid. The interface quantity, ϕ_f, is usually expressed in a value per unit area (or length in two dimensions), but the grid value, $\phi_{i,j}$, should be given in terms of value per unit volume.

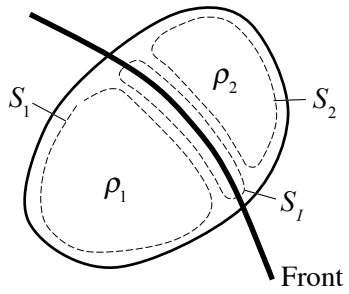

Figure 3.4: Conservation when smoothing interface quantities onto the grid.

While this is relatively obvious for quantities such as the surface force, it is true for the density gradient as well. This can be seen in the following way: Consider the domain sketched in figure 3.4 and integrate over the domain containing the interface. By the divergence theorem we can rewrite the integral as an integral over the boundary

$$\int_V \nabla \rho dv = \oint_S \rho \mathbf{n} ds \tag{3.13}$$

The contour S can be written as sum of three parts, one enclosing ρ_1, another enclosing ρ_2 and a third enclosing the interface, or $S = S_1 + S_2 + S_I$. Thus

$$\int_V \nabla \rho dv = \rho_1 \oint_{S_1} \mathbf{n} ds + \rho_2 \oint_{S_2} \mathbf{n} ds + \oint_{S_I} \rho ds \tag{3.14}$$

The first two integrals are zero, since $\oint \mathbf{n} ds = 0$ and the last integral is

$$\oint_{S_I} \rho ds = \int_{-S_I} \rho_1 \mathbf{n} ds + \int_{S_I} \rho_2 \mathbf{n} ds = \int (\rho_1 - \rho_2) \mathbf{n} ds. \tag{3.15}$$

Therefore

$$\int_V \nabla \rho dv = \int_{S_I} \Delta \rho \mathbf{n} ds. \tag{3.16}$$

where $\Delta \rho = \rho_1 - \rho_2$.

This can be generalized for any conserved variable ϕ and to ensure that the total value is conserved in the smoothing, we must require that:

$$\int_{\Delta v} \phi_{i,j}(\mathbf{x}) dv = \int_{\Delta s} \phi_f(s) ds. \tag{3.17}$$

where Δs is the area (length in two dimensions) of the interface that is smoothed to volume Δv (area in two dimensions). For two-dimensional flow, the discrete form of the grid value is therefore given by

$$\phi_{i,j} = \sum_l \phi_f^l w_{i,j}^l \frac{\Delta s_l}{\Delta x \Delta y} \tag{3.18}$$

where Δs_l is the area of element l and Δx and Δy are the grid spacings. Here we will use the same weights, $w_{i,j}^l$, for the smoothing as used for the interpolation.

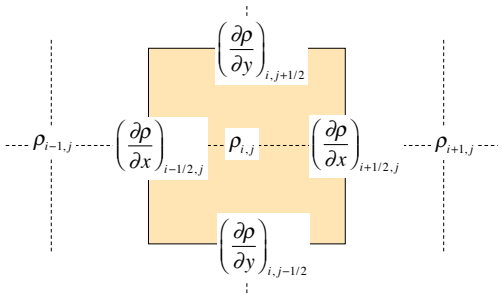

Figure 3.5: The generation of the density field.

We can, at least in principle, integrate the density gradient directly from a point where the density is known. If we distribute the gradient onto a staggered grid, the x-gradient is available at points $(i-1/2, j)$ and so on (see figure 3.5 for the two-dimensional case) and if the density at $(i-1, j)$ is known, then

$$\rho_{i,j} = \rho_{i-1,j} + \Delta x \left(\frac{\partial \rho}{\partial x}\right)_{i-1/2,j}. \tag{3.19}$$

By repeated application of this equation we can construct the density value everywhere, provided we start at a point where the density field is known. Although this operation only has to be performed at points near the front, since the density away from the front has not changed, we need to find a point from which to integrate. The density field can also depend slightly on the direction in which the integration is done and errors can accumulate if we integrate over several grid points, so that the value of the density on the other side of the interface may not be correct.

To get a more symmetric expression for the density at (i, j) we can integrate the density gradient from the other three grid points, $(i+1, j)$, $(i, j+1)$ and $(i, j-1)$ and average the results. The results can be rearranged to obtain:

$$\rho_{i,j} = 0.25 * \Big\{ \rho_{i-1,j} + \rho_{i+1,j} + \rho_{i,j-1} + \rho_{i,j+1} + \Delta x \left(\left(\frac{\partial \rho}{\partial x}\right)_{i-1/2,j} - \left(\frac{\partial \rho}{\partial x}\right)_{i+1/2,j} \right) + \Delta y \left(\left(\frac{\partial \rho}{\partial y}\right)_{i,j-1/2} - \left(\frac{\partial \rho}{\partial y}\right)_{i,j+1/2} \right) \Big\}. \tag{3.20}$$

which can be iterated to solved for $\rho_{i,j}$. This iteration can be accelerated using SOR or more advanced iterative techniques. In the code presented here, we update the density at every point in the domain, but in more advanced codes we can limit the updating to the vicinity of the interface. Iterating over the entire domain can generally spreads out the error so we can end up with errors in density away from the interface, if we stop before obtaining full convergence. Thus, limiting the iterations to only the points close to the interface is generally much more efficient.

One might be tempted to ask, why the rather elaborate approach to construct the density from the front location? Would it not be simpler to simply run over the front and set the density at the fixed grid points as a function of the minimum distance from the front? The answer is primarily that the present approach allows us to automatically deal with situations shown in figure 3.6 where two interfaces occupy

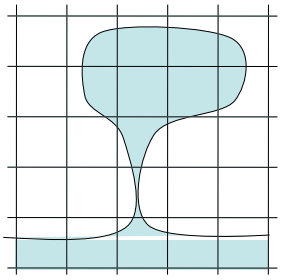

Figure 3.6: The generation of the density field close to a thin neck.

the same grid cell. By working with the gradients, which cancel on the fixed grid, the ambiguity of setting the value of a grid point next to a double interface disappears.

3.3 Restructuring the front

Since the velocity of front points varies, generally the distance between front points will change as they move. They can, in particular, move sufficiently far apart so that the front no longer is well resolved. The points can also move close together and it is usually useful to delete points that crowd together, although it is not as important as adding points.

For an interface in a two-dimensional flow, consisting of ordered points, adding and deleting points is straightforward. We start by copying the front information (the coordinates of the points) into temporary array ($\mathbf{x}_{old}$) and then copy the points back into the original array one by one, checking the distance between the point we are copying and the last point that we copied. If the distance between the points is too large, we create a new point half way between the two and put it where the next point should go. Once that is done we copy our point into the next empty location of the original array. If the distance is less than a prescribed minimum we do not copy the point into the new array and move to the next point. If the distance is above the minimum distance and below the maximum distance, we copy the point directly

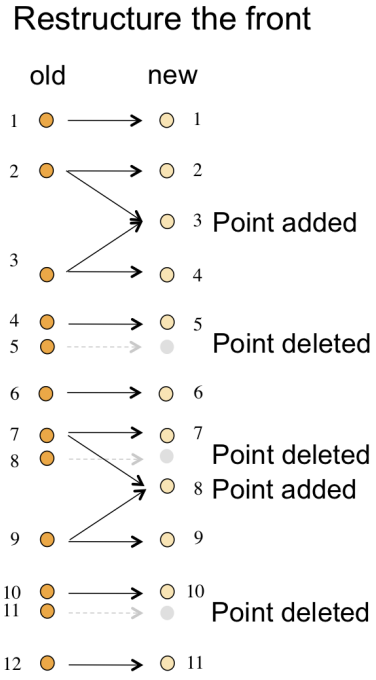

Figure 3.7: Restructuring the front. The front point vector are first copied to a new vector and the points are then copied back, one by one, adding and deleting points as needed.

into the new array. In this approach the number of front points will generally change, depending on how many points we add or delete. Figure 3.7 shows the restructuring schematically. The array on the left is the temporary one, holding the original front points, which are copied one-by-one into the new array on the right. Notice that this operations allows the last point to be deleted so in order to close the front, we set the location of the last point equal to the first one.

In the sample code below, we use linear interpolation to find the location of the new point, when we need to add one. Generally it is better to use a higher order interpolation, particularly when surface tension is included. When the front is represented by an unstructured grid—as are needed for three-dimensional flows—a more elaborate approach is generally necessary.

The most obvious place to restructure the front is right after we have moved it and before we distribute the gradients and this is what is done in the code here. For the second order code, however, where we take two steps and then average the results, we can only restructure the front once per time step and thus will move it to a different place.

3.4 Numerical Code (MATLAB)

In the code at the end of this chapter we have replaced the advection of the density used in the code presented above, by the front tracking code described in this section. This code can be downloaded from: code2.m

3.5 Results

Figure 3.8 shows the initial conditions and the final results from the code listed below, and figure 3.9 shows a three-dimensional view of the density profile at the final time (plotted using mesh(r)). Obviously the density is much better preserved than when we advocated it using the advection diffusion equation.

3.6 Exercises

1. Run the code up to a later time, for the three different grid resolutions (adjust the time step to keep it stable, if needed) and compare the results. You can also compute the location of the center of mass versus time and see how it depends on the resolution.
2. Rayleigh-Taylor Instability: Change the code to follow the evolution of a heavy fluid overlaying a lighter one. Make the side boundaries full slip, by setting the ghost tangential velocity equal to the velocity inside the domain, and change the initial condition to an interface perturbed by a cos wave of amplitude 0.1 times the domain width. You can the change the density difference and the viscosity to examine what effect these parameters have.
3. Modify the code to allow simulations of two drops. The simplest, although not the most elegant, approach is to split the front loops into two parts and do each drop separately.

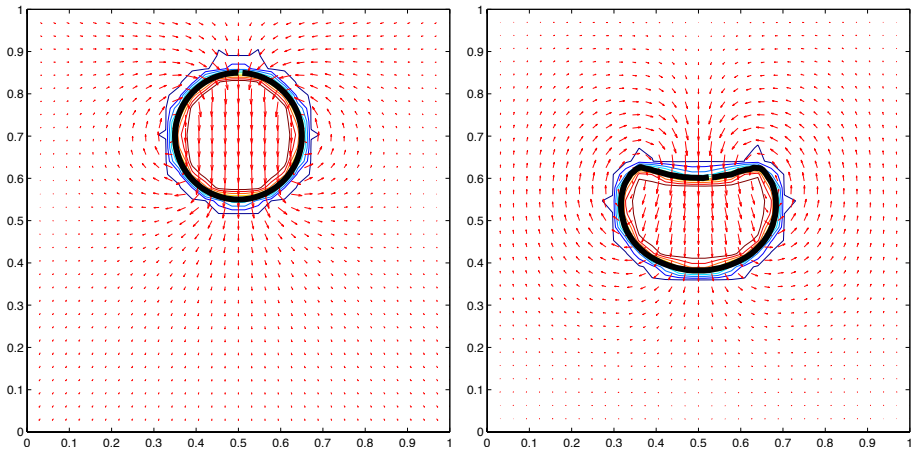

Figure 3.8: The front and the velocity field at time zero and after the drop has fallen a short distance.

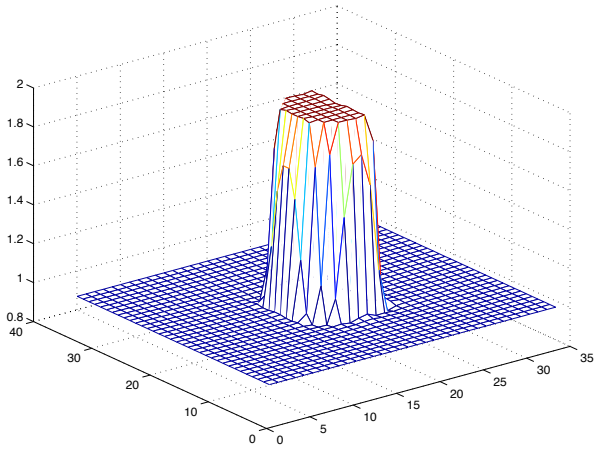

Figure 3.9: The density after the drop has fallen a short distance.

```
%========================================================================
% code2.m:
% A very simple Navier-Stokes solver for a drop falling in a rectangular
% box. A forward in time, centered in space discretization is used.
% The density is advected by a front tracking scheme and a stretched grid
% is used, allowing us to concentrate the grid points in specific areas
%========================================================================
%domain size and physical variables
Lx=1.0;Ly=1.0;gx=0.0;gy=-100.0; rho1=1.0; rho2=2.0; m0=0.01; rro=rho1;
unorth=0;usouth=0;veast=0;vwest=0;time=0.0;
rad=0.15;xc=0.5;yc=0.7; % Initial drop size and location

% Numerical variables
nx=32;ny=32;dt=0.00125;nstep=100; maxit=200;maxError=0.001;beta=1.2;

% Zero various arrys
u=zeros(nx+1,ny+2);  v=zeros(nx+2,ny+1);  p=zeros(nx+2,ny+2);
ut=zeros(nx+1,ny+2); vt=zeros(nx+2,ny+1); tmp1=zeros(nx+2,ny+2);
uu=zeros(nx+1,ny+1); vv=zeros(nx+1,ny+1); tmp2=zeros(nx+2,ny+2);

% Set the grid
dx=Lx/nx;dy=Ly/ny;
for i=1:nx+2; x(i)=dx*(i-1.5);end; for j=1:ny+2; y(j)=dy*(j-1.5);end;

% Set density in the domain and the drop
r=zeros(nx+2,ny+2)+rho1;
for i=2:nx+1,for j=2:ny+1;
  if ( (x(i)-xc)^2+(y(j)-yc)^2 < rad^2), r(i,j)=rho2; end,
end,end

%================== SETUP THE FRONT ==================
Nf=100; xf=zeros(1,Nf+2);yf=zeros(1,Nf+2);
uf=zeros(1,Nf+2);vf=zeros(1,Nf+2);
tx=zeros(1,Nf+2);ty=zeros(1,Nf+2);

for l=1:Nf+2, xf(l)=xc-rad*sin(2.0*pi*(l-1)/(Nf));
              yf(l)=yc+rad*cos(2.0*pi*(l-1)/(Nf));end
%================== START TIME LOOP=====================================
 for is=1:nstep,is
    fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
 %---------------------------------------------------------
    % tangential velocity at boundaries
    u(1:nx+1,1)=2*usouth-u(1:nx+1,2);u(1:nx+1,ny+2)=2*unorth-u(1:nx+1,ny+1);
    v(1,1:ny+1)=2*vwest-v(2,1:ny+1);v(nx+2,1:ny+1)=2*veast-v(nx+1,1:ny+1);

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity
      ut(i,j)=u(i,j)+dt*(-0.25*(((u(i+1,j)+u(i,j))^2-(u(i,j)+   ...
          u(i-1,j))^2)/dx+((u(i,j+1)+u(i,j))*(v(i+1,j)+         ...
          v(i,j))-(u(i,j)+u(i,j-1))*(v(i+1,j-1)+v(i,j-1)))/dy)+ ...
          (m0/(0.5*(r(i+1,j)+r(i,j))) )*(                          ...
                  (u(i+1,j)-2*u(i,j)+u(i-1,j))/dx^2+            ...
                  (u(i,j+1)-2*u(i,j)+u(i,j-1))/dy^2 )+  gx);
    end,end

    for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity
      vt(i,j)=v(i,j)+dt*(-0.25*(((u(i,j+1)+u(i,j))*(v(i+1,j)+   ...
          v(i,j))-(u(i-1,j+1)+u(i-1,j))*(v(i,j)+v(i-1,j)))/dx+  ...
```

```
          ((v(i,j+1)+v(i,j))^2-(v(i,j)+v(i,j-1))^2)/dy)+        ...
          (m0/(0.5*(r(i,j+1)+r(i,j))) )*(                          ...
                 (v(i+1,j)-2*v(i,j)+v(i-1,j))/dx^2+            ...
                 (v(i,j+1)-2*v(i,j)+v(i,j-1))/dy^2 )+   gy);
    end,end
%=========================================================================
    %  Compute source term and the coefficient for p(i,j)
    rt=r; lrg=1000;
    rt(1:nx+2,1)=lrg;rt(1:nx+2,ny+2)=lrg;
    rt(1,1:ny+2)=lrg;rt(nx+2,1:ny+2)=lrg;

    for i=2:nx+1,for j=2:ny+1
        tmp1(i,j)= (0.5/dt)*( (ut(i,j)-ut(i-1,j))/dx+(vt(i,j)-vt(i,j-1))/dy );
        tmp2(i,j)=1.0/( (1./dx)*( 1./(dx*(rt(i+1,j)+rt(i,j)))+...
                                  1./(dx*(rt(i-1,j)+rt(i,j)))  )+...
                        (1./dy)*(1./(dy*(rt(i,j+1)+rt(i,j)))+...
                                 1./(dy*(rt(i,j-1)+rt(i,j)))   )   );
    end,end

    for it=1:maxit                % SOLVE FOR PRESSURE
      oldArray=p;
      for i=2:nx+1,for j=2:ny+1
          p(i,j)=(1.0-beta)*p(i,j)+beta* tmp2(i,j)*(...
          (1./dx)*( p(i+1,j)/(dx*(rt(i+1,j)+rt(i,j)))+...
                      p(i-1,j)/(dx*(rt(i-1,j)+rt(i,j))) )+...
          (1./dy)*( p(i,j+1)/(dy*(rt(i,j+1)+rt(i,j)))+...
                      p(i,j-1)/(dy*(rt(i,j-1)+rt(i,j))) ) - tmp1(i,j));
      end,end
      if max(max(abs(oldArray-p))) <maxError, break,end
    end

    for i=2:nx,for j=2:ny+1   % CORRECT THE u-velocity
          u(i,j)=ut(i,j)-dt*(2.0/dx)*(p(i+1,j)-p(i,j))/(r(i+1,j)+r(i,j));
    end,end

    for i=2:nx+1,for j=2:ny   % CORRECT THE v-velocity
          v(i,j)=vt(i,j)-dt*(2.0/dy)*(p(i,j+1)-p(i,j))/(r(i,j+1)+r(i,j));
    end,end
 %================== ADVECT FRONT =====================
for l=2:Nf+1
        ip=floor(xf(l)/dx)+1; jp=floor((yf(l)+0.5*dy)/dy)+1;
        ax=xf(l)/dx-ip+1;ay=(yf(l)+0.5*dy)/dy-jp+1;
        uf(l)=(1.0-ax)*(1.0-ay)*u(ip,jp)+ax*(1.0-ay)*u(ip+1,jp)+...
                (1.0-ax)*ay*u(ip,jp+1)+ax*ay*u(ip+1,jp+1);

        ip=floor((xf(l)+0.5*dx)/dx)+1; jp=floor(yf(l)/dy)+1;
        ax=(xf(l)+0.5*dx)/dx-ip+1;ay=yf(l)/dy-jp+1;
    vf(l)=(1.0-ax)*(1.0-ay)*v(ip,jp)+ax*(1.0-ay)*v(ip+1,jp)+...
                (1.0-ax)*ay*v(ip,jp+1)+ax*ay*v(ip+1,jp+1);
   end

   for i=2:Nf+1, xf(i)=xf(i)+dt*uf(i); yf(i)=yf(i)+dt*vf(i);end  %MOVE THE FRONT
   xf(1)=xf(Nf+1);yf(1)=yf(Nf+1);xf(Nf+2)=xf(2);yf(Nf+2)=yf(2);
%------------ Add points to the front ------------
   xfold=xf;yfold=yf; j=1;
   for l=2:Nf+1
      ds=sqrt( ((xfold(l)-xf(j))/dx)^2 + ((yfold(l)-yf(j))/dy)^2);
      if (ds > 0.5)
```

```
        j=j+1;xf(j)=0.5*(xfold(l)+xf(j-1));yf(j)=0.5*(yfold(l)+yf(j-1));
        j=j+1;xf(j)=xfold(l);yf(j)=yfold(l);
      elseif (ds < 0.25)
        % DO NOTHING!
      else
       j=j+1;xf(j)=xfold(l);yf(j)=yfold(l);
      end
   end
   Nf=j-1;
   xf(1)=xf(Nf+1);yf(1)=yf(Nf+1);xf(Nf+2)=xf(2);yf(Nf+2)=yf(2);
%------------ distribute gradient --------------
   fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
   for l=2:Nf+1
       nfx=-0.5*(yf(l+1)-yf(l-1))*(rho2-rho1);
       nfy=0.5*(xf(l+1)-xf(l-1))*(rho2-rho1);  % Normal vector

       ip=floor(xf(l)/dx)+1; jp=floor((yf(l)+0.5*dy)/dy)+1;
       ax=xf(l)/dx-ip+1; ay=(yf(l)+0.5*dy)/dy-jp+1;
       fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dx/dy;
       fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dx/dy;
       fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dx/dy;
       fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dx/dy;

       ip=floor((xf(l)+0.5*dx)/dx)+1; jp=floor(yf(l)/dy)+1;
       ax=(xf(l)+0.5*dx)/dx-ip+1; ay=yf(l)/dy-jp+1;
       fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dx/dy;
       fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dx/dy;
       fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dx/dy;
       fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dx/dy;
   end
%------------ construct the density --------------
   for iter=1:maxit
     oldArray=r;
     for i=2:nx+1,for j=2:ny+1
       r(i,j)=0.25*(r(i+1,j)+r(i-1,j)+r(i,j+1)+r(i,j-1)+...
                dx*fx(i-1,j)-dx*fx(i,j)+...
                dy*fy(i,j-1)-dy*fy(i,j));
     end,end
     if max(max(abs(oldArray-r))) <maxError, break,end
   end
%=============================================================================
   time=time+dt                    % plot the results
   uu(1:nx+1,1:ny+1)=0.5*(u(1:nx+1,2:ny+2)+u(1:nx+1,1:ny+1));
   vv(1:nx+1,1:ny+1)=0.5*(v(2:nx+2,1:ny+1)+v(1:nx+1,1:ny+1));
   for i=1:nx+1,xh(i)=dx*(i-1);end;     for j=1:ny+1,yh(j)=dy*(j-1);end
   hold off,contour(x,y,flipud(rot90(r))),axis equal,axis([0 Lx 0 Ly]);
   hold on;quiver(xh,yh,flipud(rot90(uu)),flipud(rot90(vv)),'r');
   plot(xf(1:Nf),yf(1:Nf),'k','linewidth',5);pause(0.01)
end
```

Chapter 4

A More Complete Code

Here we change the code introduced in the last section to make it more complete by allowing for different viscosities in the different fluids and by introducing surface tension. We will also make the time integration second order and allow for the possibility of subtracting the hydrostatic pressure gradient.

4.1 Surface Tension

In simulations of immiscible multiphase flows we are frequently interested in length scales where surface tension is important. Surface tension acts only at the interface and the force per unit area is given by

$$\mathbf{f}_\sigma = \sigma\kappa\mathbf{n} \tag{4.1}$$

where σ is the surface tension coefficient and κ is the curvature. For two-dimensional flow, we have

$$\kappa\mathbf{n} = \frac{\partial \mathbf{t}}{\partial s} \tag{4.2}$$

where $\mathbf{t}$ is a tangent to the interface. To allow us to construct the force per unit volume on the fixed grid, we need the total force on an interface segment, or

$$\delta\mathbf{f}_\sigma^l = \sigma \int_{\Delta s_l} \frac{\partial \mathbf{t}}{\partial s}\, ds = \sigma(\mathbf{t}_{l+1/2} - \mathbf{t}_{l-1/2}). \tag{4.3}$$

Once the force at each interface point has been found, it can be smoothed onto the grid in exactly the same way as the density jump (equation 3.18). The only thing to notice is that the force found above is the force on a segment of a finite length of the front and we therefore do not need to multiply by Δs. Thus we smooth it using:

$$(\mathbf{f}_\sigma)_{i,j} = \sum_l \frac{\delta\mathbf{f}_\sigma^l w_{i,j}^l}{\Delta x \Delta y}. \tag{4.4}$$

Once we have the force on the fixed grid, we can add it to the Navier-Stokes equations. Since the surface tension is zero except at the interface, it is added as a singular force, given by

$$\sigma\kappa\mathbf{n}\delta(n) \tag{4.5}$$

where δ is a one-dimensional delta function of the normal coordinate n.

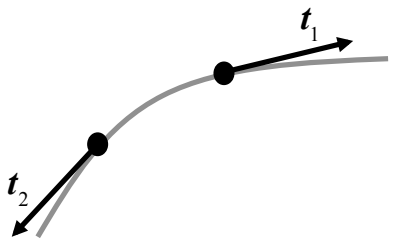

Figure 4.1: Surface tension on a finite interface segment.

4.2 Different Viscosities

For flows where the viscosity of the different fluids is different, we must use the full deformation tensor when computing the viscous stresses. The momentum equation then is

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho\nabla\mathbf{u}\mathbf{u} = -\nabla p + \rho\mathbf{g} + \nabla\cdot\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u}) + \mathbf{f} \tag{4.6}$$

For a finite volume method the viscous term can be written as

$$\mathbf{D} = \frac{1}{V}\int_V \nabla\cdot\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u})dv = \frac{1}{V}\oint_S \mu(\nabla\mathbf{u} + \nabla^T\mathbf{u})\cdot\mathbf{n}\; ds. \tag{4.7}$$

Approximating the integral of the viscous fluxes around the boundaries of the velocity control volumes (equation 2.9) by the value at the midpoint of each edge times the length of the edge results in:

$$\begin{aligned}(D_x)^n_{i+1/2,j} = \\ \frac{1}{\Delta x\Delta y}\Bigg\{\Bigg(2\Big(\mu\frac{\partial u}{\partial x}\Big)_{i+1,j} - 2\Big(\mu\frac{\partial u}{\partial x}\Big)_{i,j}\Bigg)\Delta y \\ +\Bigg(\mu\Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big)_{i+1/2,j+1/2} - \mu\Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big)_{i+1/2,j-1/2}\Bigg)\Delta x\Bigg\}\end{aligned} \tag{4.8}$$

and

$$\begin{aligned}(D_y)^n_{i,j+1/2} = \\ \frac{1}{\Delta x\Delta y}\Bigg\{\Bigg(\mu\Big(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\Big)_{i+1/2,j+1/2} - \mu\Big(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\Big)_{i-1/2,j+1/2}\Bigg)\Delta y \\ +\Bigg(2\Big(\mu\frac{\partial v}{\partial y}\Big)_{i,j+1} - 2\Big(\mu\frac{\partial v}{\partial y}\Big)_{i,j}\Bigg)\Delta x\Bigg\}.\end{aligned} \tag{4.9}$$

The velocity derivatives are found using the standard second-order centered differences, resulting in:

$$
\begin{aligned}
&(D_x)^n_{i+1/2,j} = \\
&\frac{1}{\Delta x}\Bigg\{2\mu^n_{i+1,j}\Big(\frac{u^n_{i+3/2,j}-u^n_{i+1/2,j}}{\Delta x}\Big) - 2\mu^n_{i,j}\Big(\frac{u^n_{i+1/2,j}-u^n_{i-1/2,j}}{\Delta x}\Big)\Bigg\} \\
&+\frac{1}{\Delta y}\Bigg\{\mu^n_{i+1/2,j+1/2}\Big(\frac{u^n_{i+1/2,j+1}-u^n_{i+1/2,j}}{\Delta y}+\frac{v^n_{i+1,j+1/2}-v^n_{i,j+1/2}}{\Delta x}\Big) \\
&\quad -\mu^n_{i+1/2,j-1/2}\Big(\frac{u^n_{i+1/2,j}-u^n_{i+1/2,j-1}}{\Delta y}+\frac{v^n_{i+1,j-1/2}-v^n_{i,j-1/2}}{\Delta x}\Big)\Bigg\}
\end{aligned}
\tag{4.10}
$$

and

$$
\begin{aligned}
&(D_y)^n_{i,j+1/2} = \\
&\frac{1}{\Delta x}\Bigg\{\Big(\mu^n_{i+1/2,j+1/2}\Big(\frac{u^n_{i+1/2,j+1}-u^n_{i+1/2,j}}{\Delta y}+\frac{v^n_{i+1,j+1/2}-v^n_{i,j+1/2}}{\Delta x}\Big) \\
&\quad -\mu^n_{i-1/2,j+1/2}\Big(\frac{u^n_{i-1/2,j+1}-u^n_{i-1/2,j}}{\Delta y}+\frac{v^n_{i,j+1/2}-v^n_{i-1,j+1/2}}{\Delta x}\Big)\Bigg\} \\
&+\frac{1}{\Delta y}\Bigg\{2\mu^n_{i,j+1}\Big(\frac{v^n_{i,j+3/2}-v^n_{i,j+1/2}}{\Delta y}\Big) - 2\mu^n_{i,j}\Big(\frac{v^n_{i,j+1/2}-v^n_{i,j-1/2}}{\Delta y}\Big)\Bigg\}.
\end{aligned}
\tag{4.11}
$$

To find the viscosity at the points where it is not defined, the simplest approach is to use linear interpolation, yielding:

$$
\begin{aligned}
\mu^n_{i+1/2,j+1/2} &= \frac{1}{4}(\mu^n_{i+1,j}+\mu^n_{i+1,j+1}+\mu^n_{i,j+1}+\mu^n_{i,j}) \\
\mu^n_{i+1/2,j-1/2} &= \frac{1}{4}(\mu^n_{i+1,j-1}+\mu^n_{i+1,j}+\mu^n_{i,j}+\mu^n_{i,j-1}) \\
\mu^n_{i-1/2,j+1/2} &= \frac{1}{4}(\mu^n_{i,j}+\mu^n_{i,j+1}+\mu^n_{i-1,j+1}+\mu^n_{i-1,j})
\end{aligned}
\tag{4.12}
$$

We note that instead of finding the viscosities at points where it is not defined using an arithmetic average, it is sometimes beneficial to use the geometric mean.

4.3 Second Order in Time

The code described above is only first order in time. It can easily be made second order by the following simple predictor-corrector method. The simplest such method is:

$$
\begin{aligned}
f^* &= f^n + \Delta t\Big(\frac{df}{dt}\Big)^n \\
f^{n+1} &= f^n + \frac{\Delta t}{2}\Big(\Big(\frac{df}{dt}\Big)^n + \Big(\frac{df}{dt}\Big)^*\Big)
\end{aligned}
\tag{4.13}
$$

for the integration of $f(t)$ forward in time. Here, the star on the derivative in the second step means that the derivative is found using f^*. For our purpose it is better

to rearrange the steps slightly

$$f^* = f^n + \Delta t\Big(\frac{df}{dt}\Big)^n$$
$$f^{**} = f^* + \Delta t\Big(\frac{df}{dt}\Big)^*$$
$$f^{n+1} = \frac{1}{2}\Big(f^n + f^{**}\Big) \tag{4.14}$$

The second set can be shown to be equivalent to the first one by substituting the second equation into the last one. The importance of the second form is that we can take two first order time steps and then average the new and the old results. This makes it particularly simple to extend our first order in time code to second order. We simply store the results at the beginning of a step, take two steps and average the new and old results. The only complication is that since this averaging includes the front variables, we need to move the restructuring of the front to the very beginning or the end of the time loop so that we average over the same front points.

4.4 Keeping Flows in Periodic Domains Stationary

For flows in domains that are periodic in the direction that gravity acts in, particularly if the horizontal boundaries are also periodic, there is nothing holding the fluid region in place and it will undergo free fall once gravity is turned on. The simplest way to prevent this is to add a body force equal to the weight of the fluid region that opposes the pull of gravity. Thus, we replace the pressure and gravity term by:

$$\nabla p + \rho\mathbf{g} - \rho_{avg}\mathbf{g} = \nabla p + (\rho - \rho_{avg})\mathbf{g} \tag{4.15}$$

This can be shown to conserve momentum exactly (at least in theory) but if the fluid gains momentum due to small error, there is nothing that reduces the momentum back to what it should be. For simulations with periodic boundary conditions in all directions, we therefore sometimes see a slight drift in the direction of gravity, unless we explicitly correct for it.

4.5 More Complete Code

A code that is second order in time with surface tension and variable viscosity This code can be downloaded from: code3.m

4.6 Results

Figure 4.2 shows the initial conditions and the results at three later times from the code listed below. The drops is kept nearly cylindrical by surface tension, until it boundless off the bottom wall.

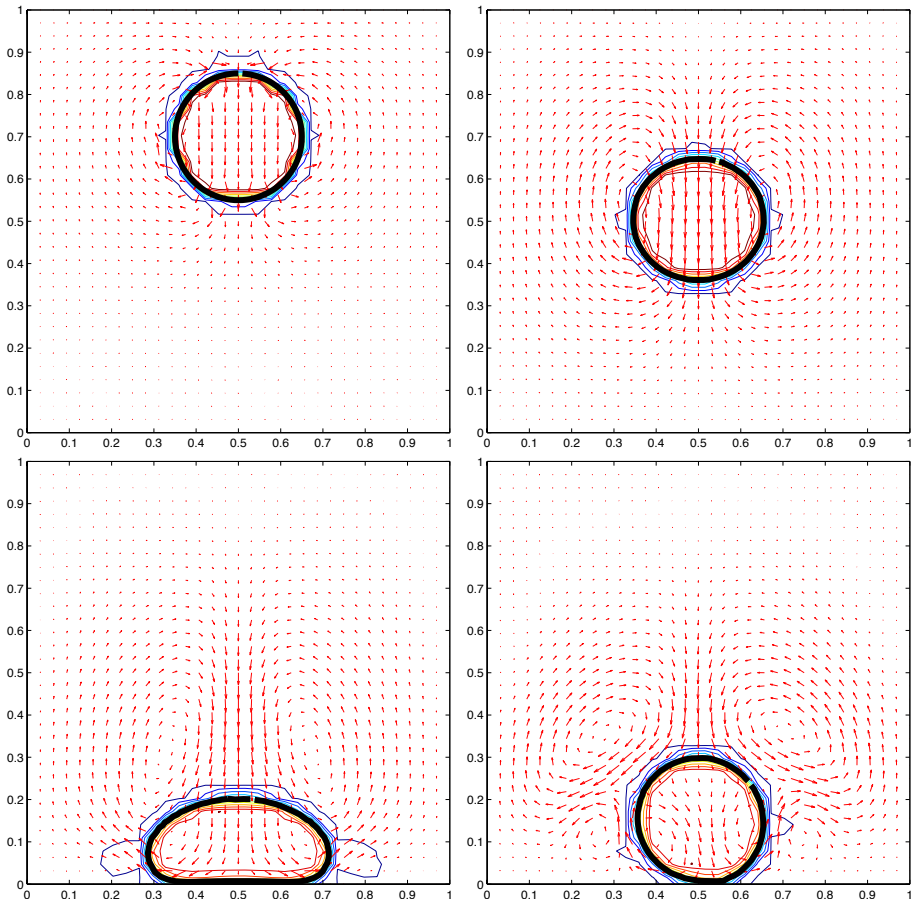

Figure 4.2: The front and the velocity field at time zero and three later times. In the last frame (lower right), the drop has bounced back and is falling again to the wall.

4.7 Exercises

1. Run the code for the three different grid resolutions (adjust the time step to keep it stable, if needed) and compare the results. You can also compute the location of the center of mass versus time and see how it depends on the resolution.
2. Examine the effect of changing the various parameters, such as the surface tension, density difference and viscosity and how the necessary resolution changes.
3. Rayleigh-Taylor Instability: Change the code to follow the evolution of a heavy fluid overlaying a lighter one. Make the side boundaries full slip, by setting the ghost tangential velocity equal to the velocity inside the domain, and change the initial condition to an interface perturbed by a cos wave of amplitude 0.1 times the domain width. You can the change the density difference, the surface tension and the viscosity to examine what effect these parameters have.
4. Modify the code to allow simulations of two bubbles. The simplest, although not the most elegant, approach is to split the front loops into two parts and do each bubble separately. Examine their interactions as they rise.
5. Modify the code to simulate the heat transfer from an initially hot drop. The heat transfer is governed by

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T = \frac{1}{\rho c} \nabla \cdot k \nabla T \tag{4.16}$$

   where T is the temperature, k is the thermal conductivity, and c is the heat capacity. The temperature at the walls can be assumed to remain constant.
6. Make the surface tension a decreasing function of the temperature and simulate the thermocapillary motion of a drop in a temperature gradient. You will have to solve the energy equation (from the problem above) on the grid and then interpolate the temperature onto the front points. When the surface tension is variable, the surface force is given by $\mathbf{f}_\sigma = \partial \sigma \mathbf{t} / \partial s$.

```
%========================================================================
% code3.m
% A very simple Navier-Stokes solver for a drop falling in a rectangular
% box. A forward in time, centered in space discretization is used.
% The density is advected by a front tracking scheme and surface tension
% and variable viscosity is included
%========================================================================
%domain size and physical variables
Lx=1.0;Ly=1.0;gx=0.0;gy=100.0; rho1=1.0; rho2=2.0;
m1=0.01; m2=0.05; sigma=10; rro=rho1;
unorth=0;usouth=0;veast=0;vwest=0;time=0.0;
rad=0.15;xc=0.5;yc=0.7; % Initial drop size and location

% Numerical variables
nx=32;ny=32;dt=0.00125;nstep=300; maxit=200;maxError=0.001;beta=1.5;

% Zero various arrys
u=zeros(nx+1,ny+2); v=zeros(nx+2,ny+1); p=zeros(nx+2,ny+2);
ut=zeros(nx+1,ny+2); vt=zeros(nx+2,ny+1); tmp1=zeros(nx+2,ny+2);
uu=zeros(nx+1,ny+1); vv=zeros(nx+1,ny+1); tmp2=zeros(nx+2,ny+2);
fx=zeros(nx+2,ny+2); fy=zeros(nx+2,ny+2);
 un=zeros(nx+1,ny+2); vn=zeros(nx+2,ny+1);   % second order

% Set the grid
dx=Lx/nx;dy=Ly/ny;
for i=1:nx+2; x(i)=dx*(i-1.5);end; for j=1:ny+2; y(j)=dy*(j-1.5);end;

% Set density and viscosity in the domain and the drop
r=zeros(nx+2,ny+2)+rho1;m=zeros(nx+2,ny+2)+m1;
rn=zeros(nx+2,ny+2); mn=zeros(nx+2,ny+2);   % second order
for i=2:nx+1,for j=2:ny+1;
  if ( (x(i)-xc)^2+(y(j)-yc)^2 < rad^2), r(i,j)=rho2;m(i,j)=m2;end,
end,end

%================== SETUP THE FRONT ===================
Nf=100; xf=zeros(1,Nf+2); yf=zeros(1,Nf+2);
xfn=zeros(1,Nf+2); yfn=zeros(1,Nf+2);     % second order
uf=zeros(1,Nf+2); vf=zeros(1,Nf+2);
tx=zeros(1,Nf+2); ty=zeros(1,Nf+2);

for l=1:Nf+2, xf(l)=xc-rad*sin(2.0*pi*(l-1)/(Nf));
               yf(l)=yc+rad*cos(2.0*pi*(l-1)/(Nf));end
%================== START TIME LOOP=====================================
 for is=1:nstep,is

   un=u; vn=v; rn=r; mn=m; xfn=xf; yfn=yf;   % second order%%%
   for  substep=1:2    % second order

%------------------ FIND SURFACE TENSION --------------
    fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
     for l=1:Nf+1,
        ds=sqrt((xf(l+1)-xf(l))^2+(yf(l+1)-yf(l))^2);
        tx(l)=(xf(l+1)-xf(l))/ds;
        ty(l)=(yf(l+1)-yf(l))/ds; % Tangent vectors
    end
    tx(Nf+2)=tx(2);ty(Nf+2)=ty(2);
for l=2:Nf+1     % Distribute to the fixed grid
```

```
    nfx=sigma*(tx(l)-tx(l-1));nfy=sigma*(ty(l)-ty(l-1));

        ip=floor(xf(l)/dx)+1; jp=floor((yf(l)+0.5*dy)/dy)+1;
        ax=xf(l)/dx-ip+1; ay=(yf(l)+0.5*dy)/dy-jp+1;
        fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dx/dy;
        fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dx/dy;
        fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dx/dy;
        fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dx/dy;

        ip=floor((xf(l)+0.5*dx)/dx)+1; jp=floor(yf(l)/dy)+1;
        ax=(xf(l)+0.5*dx)/dx-ip+1; ay=yf(l)/dy-jp+1;
        fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dx/dy;
        fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dx/dy;
        fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dx/dy;
        fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dx/dy;
    end
  %-------------------------------------------------------
  % tangential velocity at boundaries
    u(1:nx+1,1)=2*usouth-u(1:nx+1,2);u(1:nx+1,ny+2)=2*unorth-u(1:nx+1,ny+1);
    v(1,1:ny+1)=2*vwest-v(2,1:ny+1);v(nx+2,1:ny+1)=2*veast-v(nx+1,1:ny+1);

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity-ADVECTION
      ut(i,j)=u(i,j)+dt*(-0.25*(((u(i+1,j)+u(i,j))^2-(u(i,j)+   ...
          u(i-1,j))^2)/dx+((u(i,j+1)+u(i,j))*(v(i+1,j)+         ...
          v(i,j))-(u(i,j)+u(i,j-1))*(v(i+1,j-1)+v(i,j-1)))/dy)+ ...
             fx(i,j)/(0.5*(r(i+1,j)+r(i,j)))                    ...
           - (1.0 -rro/(0.5*(r(i+1,j)+r(i,j))) )*gx      );
    end,end

    for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity-ADVECTION
      vt(i,j)=v(i,j)+dt*(-0.25*(((u(i,j+1)+u(i,j))*(v(i+1,j)+   ...
          v(i,j))-(u(i-1,j+1)+u(i-1,j))*(v(i,j)+v(i-1,j)))/dx+  ...
          ((v(i,j+1)+v(i,j))^2-(v(i,j)+v(i,j-1))^2)/dy)+        ...
          fy(i,j)/(0.5*(r(i,j+1)+r(i,j)))                      ...
            - (1.0 -rro/(0.5*(r(i,j+1)+r(i,j))) )*gy      );
    end,end

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity-DIFFUSION
      ut(i,j)=ut(i,j)+dt*(...
               (1./dx)*2.*(m(i+1,j)*(1./dx)*(u(i+1,j)-u(i,j)) -        ...
                  m(i,j)  *(1./dx)*(u(i,j)-u(i-1,j)) )                 ...
          +(1./dy)*( 0.25*(m(i,j)+m(i+1,j)+m(i+1,j+1)+m(i,j+1))*       ...
            ((1./dy)*(u(i,j+1)-u(i,j)) + (1./dx)*(v(i+1,j)-v(i,j)) ) -  ...
                0.25*(m(i,j)+m(i+1,j)+m(i+1,j-1)+m(i,j-1))*            ...
          ((1./dy)*(u(i,j)-u(i,j-1))+ (1./dx)*(v(i+1,j-1)- v(i,j-1))) )...
           )/(0.5*(r(i+1,j)+r(i,j))   );
    end,end

   for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity-DIFFUSION
      vt(i,j)=vt(i,j)+dt*(...
          (1./dx)*( 0.25*(m(i,j)+m(i+1,j)+m(i+1,j+1)+m(i,j+1))*        ...
           ((1./dy)*(u(i,j+1)-u(i,j)) + (1./dx)*(v(i+1,j)-v(i,j)) ) -  ...
                0.25*(m(i,j)+m(i,j+1)+m(i-1,j+1)+m(i-1,j))*            ...
          ((1./dy)*(u(i-1,j+1)-u(i-1,j))+ (1./dx)*(v(i,j)- v(i-1,j))) )...
           +(1./dy)*2.*(m(i,j+1)*(1./dy)*(v(i,j+1)-v(i,j)) -           ...
                  m(i,j) *(1./dy)*(v(i,j)-v(i,j-1)) )                  ...
           )/(0.5*(r(i,j+1)+r(i,j))   );
    end,end
```

```
%=========================================================================
    % Compute source term and the coefficient for p(i,j)
    rt=r; lrg=1000;
    rt(1:nx+2,1)=lrg;rt(1:nx+2,ny+2)=lrg;
    rt(1,1:ny+2)=lrg;rt(nx+2,1:ny+2)=lrg;

    for i=2:nx+1,for j=2:ny+1
        tmp1(i,j)= (0.5/dt)*( (ut(i,j)-ut(i-1,j))/dx+(vt(i,j)-vt(i,j-1))/dy );
        tmp2(i,j)=1.0/( (1./dx)*( 1./(dx*(rt(i+1,j)+rt(i,j)))+...
                                  1./(dx*(rt(i-1,j)+rt(i,j)))  )+...
                        (1./dy)*(1./(dy*(rt(i,j+1)+rt(i,j)))+...
                                 1./(dy*(rt(i,j-1)+rt(i,j)))   )   );
    end,end

    for it=1:maxit                % SOLVE FOR PRESSURE
      oldArray=p;
      for i=2:nx+1,for j=2:ny+1
          p(i,j)=(1.0-beta)*p(i,j)+beta* tmp2(i,j)*(...
          (1./dx)*( p(i+1,j)/(dx*(rt(i+1,j)+rt(i,j)))+...
                    p(i-1,j)/(dx*(rt(i-1,j)+rt(i,j))) )+...
          (1./dy)*( p(i,j+1)/(dy*(rt(i,j+1)+rt(i,j)))+...
                    p(i,j-1)/(dy*(rt(i,j-1)+rt(i,j))) ) - tmp1(i,j));
      end,end
      if max(max(abs(oldArray-p))) <maxError, break,end
    end

    for i=2:nx,for j=2:ny+1   % CORRECT THE u-velocity
          u(i,j)=ut(i,j)-dt*(2.0/dx)*(p(i+1,j)-p(i,j))/(r(i+1,j)+r(i,j));
    end,end

    for i=2:nx+1,for j=2:ny   % CORRECT THE v-velocity
          v(i,j)=vt(i,j)-dt*(2.0/dy)*(p(i,j+1)-p(i,j))/(r(i,j+1)+r(i,j));
    end,end
 %================== ADVECT FRONT =====================
for l=2:Nf+1
        ip=floor(xf(l)/dx)+1; jp=floor((yf(l)+0.5*dy)/dy)+1;
        ax=xf(l)/dx-ip+1;ay=(yf(l)+0.5*dy)/dy-jp+1;
        uf(l)=(1.0-ax)*(1.0-ay)*u(ip,jp)+ax*(1.0-ay)*u(ip+1,jp)+...
                (1.0-ax)*ay*u(ip,jp+1)+ax*ay*u(ip+1,jp+1);

        ip=floor((xf(l)+0.5*dx)/dx)+1; jp=floor(yf(l)/dy)+1;
        ax=(xf(l)+0.5*dx)/dx-ip+1;ay=yf(l)/dy-jp+1;
    vf(l)=(1.0-ax)*(1.0-ay)*v(ip,jp)+ax*(1.0-ay)*v(ip+1,jp)+...
                (1.0-ax)*ay*v(ip,jp+1)+ax*ay*v(ip+1,jp+1);
    end

    for i=2:Nf+1, xf(i)=xf(i)+dt*uf(i); yf(i)=yf(i)+dt*vf(i);end  %MOVE THE FRONT
xf(1)=xf(Nf+1);yf(1)=yf(Nf+1);xf(Nf+2)=xf(2);yf(Nf+2)=yf(2);

%------------ distribute gradient --------------
   fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
   for l=2:Nf+1
       nfx=-0.5*(yf(l+1)-yf(l-1))*(rho2-rho1);
       nfy=0.5*(xf(l+1)-xf(l-1))*(rho2-rho1);  % Normal vector

       ip=floor(xf(l)/dx)+1; jp=floor((yf(l)+0.5*dy)/dy)+1;
       ax=xf(l)/dx-ip+1; ay=(yf(l)+0.5*dy)/dy-jp+1;
       fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dx/dy;
```

```
       fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dx/dy;
       fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dx/dy;
       fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dx/dy;

       ip=floor((xf(l)+0.5*dx)/dx)+1; jp=floor(yf(l)/dy)+1;
       ax=(xf(l)+0.5*dx)/dx-ip+1; ay=yf(l)/dy-jp+1;
       fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dx/dy;
       fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dx/dy;
       fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dx/dy;
       fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dx/dy;
   end
%------------ construct the density --------------
   for iter=1:maxit
     oldArray=r;
     for i=2:nx+1,for j=2:ny+1
       r(i,j)=(1.0-beta)*r(i,j)+beta*...
                0.25*(r(i+1,j)+r(i-1,j)+r(i,j+1)+r(i,j-1)+...
                dx*fx(i-1,j)-dx*fx(i,j)+...
                dy*fy(i,j-1)-dy*fy(i,j));
     end,end
     if max(max(abs(oldArray-r))) <maxError, break,end
   end
%------------ update the viscosity --------------
   m=zeros(nx+2,ny+2)+m1;
   for i=2:nx+1,for j=2:ny+1
      m(i,j)=m1+(m2-m1)*(r(i,j)-rho1)/(rho2-rho1);
   end,end

end % end the time iteration---second order

u=0.5*(u+un); v=0.5*(v+vn); r=0.5*(r+rn); m=0.5*(m+mn);  % second order
xf=0.5*(xf+xfn); yf=0.5*(yf+yfn);   % second order

   %------------ Add points to the front ------------
   xfold=xf;yfold=yf; j=1;
   for l=2:Nf+1
      ds=sqrt( ((xfold(l)-xf(j))/dx)^2 + ((yfold(l)-yf(j))/dy)^2);
      if (ds > 0.5)
        j=j+1;xf(j)=0.5*(xfold(l)+xf(j-1));yf(j)=0.5*(yfold(l)+yf(j-1));
        j=j+1;xf(j)=xfold(l);yf(j)=yfold(l);
      elseif (ds < 0.25)
        % DO NOTHING!
      else
       j=j+1;xf(j)=xfold(l);yf(j)=yfold(l);
      end
   end
   Nf=j-1;
   xf(1)=xf(Nf+1);yf(1)=yf(Nf+1);xf(Nf+2)=xf(2);yf(Nf+2)=yf(2);
%=============================================================================
   time=time+dt                    % plot the results
   uu(1:nx+1,1:ny+1)=0.5*(u(1:nx+1,2:ny+2)+u(1:nx+1,1:ny+1));
   vv(1:nx+1,1:ny+1)=0.5*(v(2:nx+2,1:ny+1)+v(1:nx+1,1:ny+1));
   for i=1:nx+1,xh(i)=dx*(i-1);end;     for j=1:ny+1,yh(j)=dy*(j-1);end
   hold off,contour(x,y,flipud(rot90(r))),axis equal,axis([0 Lx 0 Ly]);
   hold on;quiver(xh,yh,flipud(rot90(uu)),flipud(rot90(vv)),'r');
   plot(xf(1:Nf),yf(1:Nf),'k','linewidth',5);pause(0.01)
end
```

Chapter 5

More Advanced Code

The code introduced in the last chapter is a fully functional code and can be used to examine a wide variety of problems (although only for two-dimensional flows). Here we introduce two changes that fist of all allow us to use the grid points a little more economically and to more easily simulate higher Reynolds number flows.

5.1 Stretched Grids

A regular structured uniform grid is relatively simple to implement and works for problems where all parts of the domain need to be well resolved. Often, however, we encounter situations where we would like the boundaries to be far away, or where we would like to use a fine grid in part of the domain, such as near boundaries. As it turns out, modifying the code described above to allow for unevenly spaced grid lines is fairly simple. Each grid line remains straight so this is far from a general grid refinement strategy but nevertheless, such stretched grids allow us to do a number of problems that would require an excessive number of grid points if the grid spacing was uniform. Figure 5.1 show a stretched grid where the resolution is concentrated near the middle of the left boundary.

We start by mapping the physical domain (where the grid lines are unevenly spaced) into a new domain where the grid lines are evenly spaced. The coordinates in the physical domain are a function of the mapped coordinates:

$$x = x(\xi), \quad y = y(\eta) \tag{5.1}$$

Since x is a function of ξ and y is a function of Δx only, writing the governing equations in the new coordinate systems is much simpler then for a complete mapping where each coordinate depends of both of the mapped coordinates. For our case, we have $f = f(x(\xi), y(\eta))$ so the derivative is

$$\frac{\partial f}{\partial \xi} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial \xi} \tag{5.2}$$

or

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial \xi} / \frac{\partial x}{\partial \xi} \tag{5.3}$$

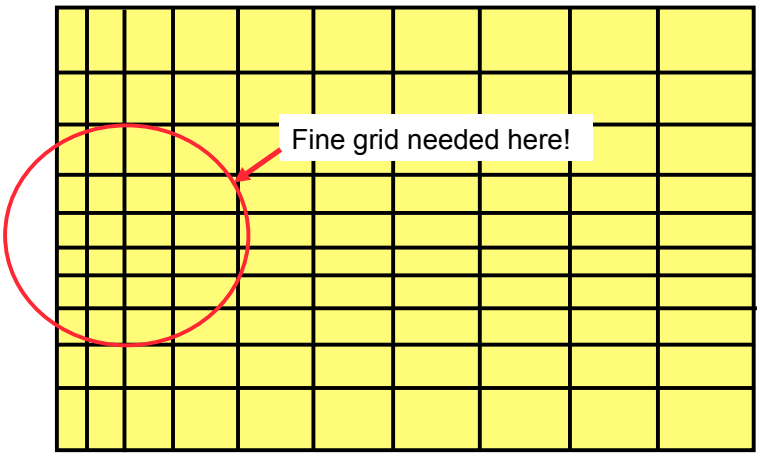

Figure 5.1: A stretched grid where the gridlines are straight but unevenly spaced. Here the grid is fine near the middle of the left boundary.

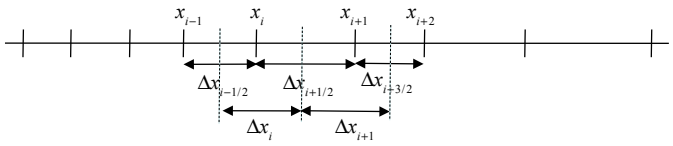

Figure 5.2: The notation for the x-axis.

Using a second order centered difference approximation for the changes in the x coordinate with ξ gives

$$\frac{\partial x}{\partial \xi} \approx \frac{\Delta x}{\Delta \xi} = \Delta x \tag{5.4}$$

if we take $\Delta\xi = 1$. Since the total size of the domain in the mapped coordinates is arbitrary, it is a common convention to take it equal to the total number of grid points, so each grid line is separated by exactly unity. Thus, there is no need to divide by the constant grid spacing in the mapped domain.

Since the grid lines are no longer uniformly spaced, we set up an an array with their locations. Usually we specify the locations of the velocity points $x_{i+1/2}$ and $y_{j+1/2}$ since those will coincide with the boundaries of the domain and find the location of the pressure points by interpolation: $x_i = (1/2)(x_{i+1/2} - x_{i-1/2})$ and $y_j = (1/2)(x_{j+1/2} - y_{j-1/2})$. The grid spacings at the pressure and velocity point are then precomputed by $\Delta x_i = x_{i+1/2} - x_{i-1/2}$ and $\Delta x_{i+1/2} = x_i - x_{i-1}$. For the y direction we have $\Delta y_i = y_{j+1/2} - y_{j-1/2}$ and $\Delta y_{j+1/2} = y_j - y_{j-1}$. The notational convention, using the x axis as an example, is shown in figure 5.2.

Developing numerical approximations for the governing equations on stretched grid is therefore straight forward. Instead of using one universal Δx (and Δy) we must use the local value. For completeness we list the numerical approximations below, even

though some of the equations heave been listed before and in other cases the changes are relatively minimal.

We start with the Navier-Stokes equations for two-dimensional flows:

$$\frac{\partial u}{\partial t}+\frac{\partial uu}{\partial x}+\frac{\partial uv}{\partial x}=\frac{-1}{\rho}\frac{\partial p}{\partial x}+\Big(\frac{\rho_0}{\rho}-1\Big)g_x+\frac{1}{\rho}\left(\frac{\partial}{\partial x}2\mu\frac{\partial u}{\partial x}+\frac{\partial}{\partial y}\mu\Big(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\Big)+f_x\right)$$
$$\frac{\partial v}{\partial t}+\frac{\partial uv}{\partial x}+\frac{\partial vv}{\partial x}=\frac{-1}{\rho}\frac{\partial p}{\partial y}+\Big(\frac{\rho_0}{\rho}-1\Big)g_y+\frac{1}{\rho}\left(\frac{\partial}{\partial x}\mu\Big(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\Big)+\frac{\partial}{\partial y}2\mu\frac{\partial v}{\partial y}+f_y\right) \quad (5.5)$$

A numerical approximation for the velocities consist of a predictor step

$$\frac{u^*_{i+1/2,j}-u^n_{i+1/2,j}}{\Delta t}+(A_x)^n_{i+1/2,j}=$$
$$+\frac{1}{\frac{1}{2}(\rho^n_{i+1,j}+\rho^n_{i,j})}\left[\big(D_x\big)^n_{i+1/2,j}+\big(\rho_o-\frac{1}{2}(\rho^n_{i+1,j}+\rho^n_{i,j})\big)g_x+(f_x)^n_{i+1/2,j}\right] \quad (5.6)$$

and

$$\frac{v^*_{i,j+1/2}-v^n_{i,j+1/2}}{\Delta t}+(A_y)^n_{i,j+1/2}=\Big(\frac{\rho_0}{\frac{1}{2}(\rho^n_{i,j+1}+\rho^n_{i,j})}-1\Big)g_y$$
$$+\frac{1}{\frac{1}{2}(\rho^n_{i,j+1}+\rho^n_{i,j})}\left[\big(D_y\big)^n_{i,j+1/2}+(f_y)^n_{i,j+1/2}\right]. \quad (5.7)$$

and a correction step

$$\frac{u^{n+1}_{i+1/2,j}-u^*_{i+1/2,j}}{\Delta t}=\frac{1}{\frac{1}{2}(\rho^n_{i+1,j}+\rho^n_{i,j})}\ \frac{p_{i+1,j}-p_{i,j}}{\Delta x_{i+1/2}} \quad (5.8)$$

and

$$\frac{v^{n+1}_{i,j+1/2}-v^*_{i,j+1/2}}{\Delta t}=\frac{1}{\frac{1}{2}(\rho^n_{i,j+1}+\rho^n_{i,j})}\ \frac{p_{i,j+1}-p_{i,j}}{\Delta y_{j+1/2}}. \quad (5.9)$$

The x-component of the advection, at $(i+1/2,j)$, is given by

$$(A_x)_{i+1/2,j}=$$
$$\frac{1}{\Delta x_{i+1/2}}\Big\{(uu)_{i+1,j}-(uu)_{i,j}\Big\}+\frac{1}{\Delta y_j}\Big\{(uv)_{i+1/2,j+1/2}-(uv)_{i+1/2,j-1/2}\Big\}$$
$$=\frac{1}{\Delta x_{i+1/2}}\left[\Big(\frac{u^n_{i+3/2,j}+u^n_{i+1/2,j}}{2}\Big)^2-\Big(\frac{u^n_{i+1/2,j}+u^n_{i-1/2,j}}{2}\Big)^2\right\}$$
$$+\frac{1}{\Delta y_j}\Big\{\Big(\frac{u^n_{i+1/2,j+1}+u^n_{i+1/2,j}}{2}\Big)\Big(\frac{v^n_{i+1,j+1/2}+v^n_{i,j+1/2}}{2}\Big)$$
$$-\Big(\frac{u^n_{i+1/2,j}+u^n_{i+1/2,j-1}}{2}\Big)\Big(\frac{v^n_{i+1,j-1/2}+v^n_{i,j-1/2}}{2}\Big)\Big\} \quad (5.10)$$

The y-component of the advection, at the $(i, j+1/2)$ point, is:

$$
\begin{aligned}
&(A_y)_{i,j+1/2} = \\
&\frac{1}{\Delta x_i}\Big\{(uv)_{i+1/2,j+1/2} - (uv)_{i-1/2,j+1/2}\Big\} + \frac{1}{\Delta y_{j+1/2}}\Big\{(vv)_{i,j+1} - (vv)_{i,j}\Big\}. \\
&= \frac{1}{\Delta x_i}\Big\{\Big(\frac{u^n_{i+1/2,j} + u^n_{i+1/2,j+1}}{2}\Big)\Big(\frac{v^n_{i,j+1/2} + v^n_{i+1,j+1/2}}{2}\Big) \\
&\quad -\Big(\frac{u^n_{i-1/2,j+1} + u^n_{i-1/2,j}}{2}\Big)\Big(\frac{v^n_{i,j+1/2} + v^n_{i-1,j+1/2}}{2}\Big)\Big\} \\
&+\frac{1}{\Delta y_{j+1/2}}\Big\{\Big(\frac{v^n_{i,j+3/2} + v^n_{i,j+1/2}}{2}\Big)^2 - \Big(\frac{v^n_{i,j+1/2} + v^n_{i,j-1/2}}{2}\Big)^2\Big\}. \qquad (5.11)
\end{aligned}
$$

The diffusion terms are:

$$
\begin{aligned}
&(D_x)^n_{i+1/2,j} = \\
&\frac{1}{\Delta x_{i+1/2}}\left\{2\Big(\mu\frac{\partial u}{\partial x}\Big)_{i+1,j} - 2\Big(\mu\frac{\partial u}{\partial x}\Big)_{i,j}\right\} \\
&+\frac{1}{\Delta y_j}\left\{\left(\mu\Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big)\right)_{i+1/2,j+1/2} - \left(\mu\Big(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\Big)\right)_{i+1/2,j-1/2}\right\} \\
&= \frac{1}{\Delta x_{i+1/2}}\left\{2\mu^n_{i+1,j}\Big(\frac{u^n_{i+3/2,j} - u^n_{i+1/2,j}}{\Delta x_{i+1}}\Big) - 2\mu^n_{i,j}\Big(\frac{u^n_{i+1/2,j} - u^n_{i-1/2,j}}{\Delta x_i}\Big)\right\} \\
&\quad +\frac{1}{\Delta y_j}\left\{\mu^n_{i+1/2,j+1/2}\Big(\frac{u^n_{i+1/2,j+1} - u^n_{i+1/2,j}}{\Delta y_{j+1/2}} + \frac{v^n_{i+1,j+1/2} - v^n_{i,j+1/2}}{\Delta x_{i+1/2}}\Big)\right. \\
&\quad \left. -\mu^n_{i+1/2,j-1/2}\Big(\frac{u^n_{i+1/2,j} - u^n_{i+1/2,j-1}}{\Delta y_{j-1/2}} + \frac{v^n_{i+1,j-1/2} - v^n_{i,j-1/2}}{\Delta x_{i+1/2}}\Big)\right\} \qquad (5.12)
\end{aligned}
$$

$$
\begin{aligned}
&(D_y)^n_{i,j+1/2} = \\
&\frac{1}{\Delta x_i}\left\{\left(\mu\Big(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\Big)\right)_{i+1/2,j+1/2} - \left(\mu\Big(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\Big)\right)_{i-1/2,j+1/2}\right\} \\
&+\frac{1}{\Delta y_{j+1/2}}\left\{2\Big(\mu\frac{\partial v}{\partial y}\Big)_{i,j+1} - 2\Big(\mu\frac{\partial v}{\partial y}\Big)_{i,j}\right\} \\
&= \frac{1}{\Delta x_i}\left\{\mu^n_{i+1/2,j+1/2}\Big(\frac{u^n_{i+1/2,j+1} - u^n_{i+1/2,j}}{\Delta y_{j+1/2}} + \frac{v^n_{i+1,j+1/2} - v^n_{i,j+1/2}}{\Delta x_{i+1/2}}\Big)\right. \\
&\quad \left. -\mu^n_{i-1/2,j+1/2}\Big(\frac{u^n_{i-1/2,j+1} - u^n_{i-1/2,j}}{\Delta y_{j+1/2}} + \frac{v^n_{i,j+1/2} - v^n_{i-1,j+1/2}}{\Delta x_{i-1/2}}\Big)\right\} \\
&+\frac{1}{\Delta y_{j+1/2}}\left\{2\mu^n_{i,j+1}\Big(\frac{v^n_{i,j+3/2} - v^n_{i,j+1/2}}{\Delta y_{j+1}}\Big) - 2\mu^n_{i,j}\Big(\frac{v^n_{i,j+1/2} - v^n_{i,j-1/2}}{\Delta y_j}\Big)\right\}. \qquad (5.13)
\end{aligned}
$$

Where the viscosity at the points where it is not defined, is found by linear interpola-

tion:

$$\begin{aligned}\mu^n_{i+1/2,j+1/2} &= \frac{1}{4}(\mu^n_{i+1,j} + \mu^n_{i+1,j+1} + \mu^n_{i,j+1} + \mu^n_{i,j}) \\ \mu^n_{i+1/2,j-1/2} &= \frac{1}{4}(\mu^n_{i+1,j-1} + \mu^n_{i+1,j} + \mu^n_{i,j} + \mu^n_{i,j-1}) \\ \mu^n_{i-1/2,j+1/2} &= \frac{1}{4}(\mu^n_{i,j} + \mu^n_{i,j+1} + \mu^n_{i-1,j+1} + \mu^n_{i-1,j})\end{aligned} \tag{5.14}$$

The incompressibility condition is

$$\frac{1}{\Delta x_i}\Big(u^{n+1}_{i+1/2,j} - u^{n+1}_{i-1/2,j}\Big) + \frac{1}{\Delta y_j}\Big(v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2}\Big) = 0 \tag{5.15}$$

Substitution the equation for the correction velocities (equations 5.9 and 5.8) in to the incompressibility conditions gives a pressure equation:

$$\begin{aligned}&\frac{1}{\Delta x_i}\left\{\frac{1}{\frac{1}{2}(\rho^n_{i+1,j}+\rho^n_{i,j})}\Big(\frac{p_{i+1,j}-p_{i,j}}{\Delta x_{i+1/2}}\Big) - \frac{1}{\frac{1}{2}(\rho^n_{i,j}+\rho^n_{i-1,j})}\Big(\frac{p_{i,j}-p_{i-1,j}}{\Delta x_{i-1/2}}\Big)\right\} \\ &+\frac{1}{\Delta y_j}\left\{\frac{1}{\frac{1}{2}(\rho^n_{i,j+1}+\rho^n_{i,j})}\Big(\frac{p_{i,j+1}-p_{i,j}}{\Delta y_{j+1/2}}\Big) - \frac{1}{\frac{1}{2}(\rho^n_{i,j}+\rho^n_{i,j-1})}\Big(\frac{p_{i,j}-p_{i,j-1}}{\Delta y_{j-1/2}}\Big)\right\} \\ &= \frac{1}{\Delta t}\left\{\frac{u^*_{i+1/2,j}-u^*_{i-1/2,j}}{\Delta x_i} + \frac{v^*_{i,j+1/2}-v^*_{i,j-1/2}}{\Delta y_j}\right\}\end{aligned} \tag{5.16}$$

Defining

$$S_{i,j} = \frac{u^*_{i+1/2,j}-u^*_{i-1/2,j}}{\Delta x_i} + \frac{v^*_{i,j+1/2}-v^*_{i,j-1/2}}{\Delta y_j} \tag{5.17}$$

and

$$\begin{aligned}C_{i,j} = &\frac{1}{\Delta x_i}\left\{\frac{1}{\frac{1}{2}(\rho^n_{i+1,j}+\rho^n_{i,j})}\Big(\frac{1}{\Delta x_{i+1/2}}\Big) - \frac{1}{\frac{1}{2}(\rho^n_{i,j}+\rho^n_{i-1,j})}\Big(\frac{1}{\Delta x_{i-1/2}}\Big)\right\} \\ &+\frac{1}{\Delta y_j}\left\{\frac{1}{\frac{1}{2}(\rho^n_{i,j+1}+\rho^n_{i,j})}\Big(\frac{1}{\Delta y_{j+1/2}}\Big) - +\frac{1}{\frac{1}{2}(\rho^n_{i,j}+\rho^n_{i,j-1})}\Big(\frac{1}{\Delta y_{j-1/2}}\Big)\right\}\end{aligned} \tag{5.18}$$

we can rewrite the pressure equation as

$$\begin{aligned}p_{i,j}C_{i,j} = &\frac{1}{\Delta x_i}\left\{\frac{1}{\frac{1}{2}(\rho^n_{i+1,j}+\rho^n_{i,j})}\Big(\frac{p_{i+1,j}}{\Delta x_{i+1/2}}\Big) + \frac{1}{\frac{1}{2}(\rho^n_{i,j}+\rho^n_{i-1,j})}\Big(\frac{p_{i-1,j}}{\Delta x_{i-1/2}}\Big)\right\} \\ &+\frac{1}{\Delta y_j}\left\{\frac{1}{\frac{1}{2}(\rho^n_{i,j+1}+\rho^n_{i,j})}\Big(\frac{p_{i,j+1}}{\Delta y_{j+1/2}}\Big) + \frac{1}{\frac{1}{2}(\rho^n_{i,j}+\rho^n_{i,j-1})}\Big(\frac{p_{i,j-1}}{\Delta y_{j-1/2}}\Big)\right\} - \frac{S_{i,j}}{\Delta t}\end{aligned} \tag{5.19}$$

which can be solved by iteration.

5.1.1 Advecting the Front

The main challenge in coupling the front and the stretched grid is how to find the fixed grid points that are closest to a given front point. For grids with a fixed spacing this is straight forward since there is a direct relation between a spatial location. For uneven grid spacing this relationship does not exist. However, if we know the coordinate of

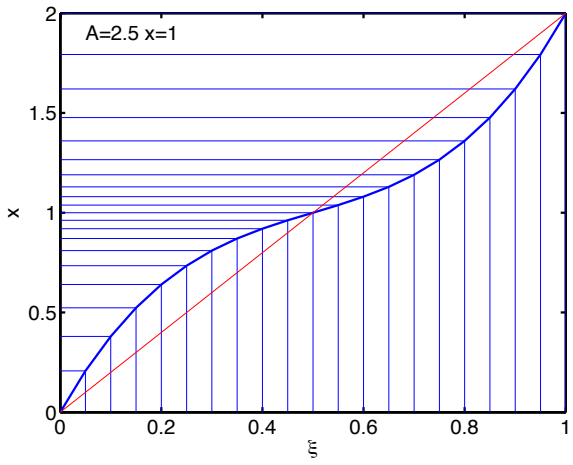

Figure 5.3: A simple mapping function to generate a well resolved internal region.

a front point in the mapped domain, the direct relationship is restored. We therefore store the location of the front points in the mapped domain, called sxf (l) and syf(l) in the code below, and advect the front by updating these coordinates. To find the physical coordinates of each front point, we then interpolate from the fixed grid. The velocities on the fixed grids are the rate at which a fluid particle moves in physical space so we need to relate the motion of a front point in the mapped domain to the physical velocity. This is easily done since

$$u = \frac{\partial x}{\partial t} = \frac{\partial x}{\partial \xi}\frac{\partial \xi}{\partial t}. \tag{5.20}$$

Thus,

$$\frac{\partial \xi}{\partial t} = \frac{u}{\partial x/\partial \xi} \approx \frac{u}{\Delta x}, \tag{5.21}$$

and a similar expression for the v velocity. Thus, when the velocity is interpolated from the fixed grid to the front, we interpolate $u_{i,j}/\Delta x_i$ and $v_{i,j}/\Delta y_j$ rather than $(u_{i,j}, v_{i,j})$ as we do for the evenly spaced grid.

When interpolating information from the fixed grid to the front and smoothing information from the front to the fixed grid, we need to use the size of the grid spacing at the point we are working with. Point addition and deletion is done in the mapped space, but all other operations on the front itself take place in the physical coordinates and are unchanged.

5.1.2 Grid Generation

There are many ways to set the distribution of gridlines across the computational domain. The task is to find a mapping $x = x(\xi)$ that results in the desired distribution of grid lines. Here we will use a particularly simple approach and use a cubic polynomial to map the gridlines from an equispaced distribution to one where we cluster the lines

in specific regions. The mapping function does not need to be an analytical function but using one makes laying down the gridlines simple. Since the grid lines have to cover the computational domain, we select a function that is equal to 0 at $\xi = 0$ and L at $\xi = 1$. A linear mapping, $x = L\xi/$ accomplishes this task but results in evenly spaced grid lines. To make the mapping nonlinear we add a function that is zero at the endpoints to the line. $x(\xi)/L = \xi + f(\xi) * \xi * (1 - \xi)$ does so. The function $f(\xi)$ can be selected in many ways. In general we will want it to change sign somewhere in the domain and taking $f(\xi) = \pm(\xi_c - \xi)$ ensure that it does so at $\xi = \xi_c$. At ξ_c we have $x/L = \xi$, since $f(\xi) = 0$, so $\xi_c = Lx_c$ there. Thus, our complete mapping function is (multiplying through by L):

$$x(\xi) = L\xi + A\xi(1 - \xi)(x_c - L\xi) \tag{5.22}$$

where A can be positive or Negative.

In figure 5.3 we plot $x = x(\xi)$ for a $1 \leq x \leq 2$ domain mapped onto $1 \leq \xi \leq 1$ domain using $A = 2.5$ and $x_c = 1$. Notice that $A > 0$ results in grid points clustered inside the domain around $x = x_c$ and $A < 0$ produces a grid with points clustered near the walls, with the grid being coarsest around $x = x_c$.

5.1.3 Stretched Grid Code

This code can be downloaded from: code4.m

```
LOOKS LIKE THIS CODE IS ONLY FIRST ORDER!!!!!
%========================================================================
% code4.m
% A very simple Navier-Stokes solver for a drop falling in a rectangular
% box. A forward in time, centered in space discretization is used.
% The density is advected by a front tracking scheme and a stretched grid
% is used, allowing us to concentrate the grid points in specific areas
%========================================================================
%domain size and physical variables
Lx=1.0;Ly=1.0;gx=0.0;gy=100.0; rho1=1.0; rho2=2.0;
m1=0.01; m2=0.05; sigma=10; rro=rho1;
un=0;us=0;ve=0;vw=0;time=0.0;
rad=0.15;xc=0.5;yc=0.7; % Initial drop size and location

% Numerical variables
nx=32;ny=32;dt=0.00125;nstep=300;maxit=2000;maxError=0.0001;beta=1.5;
dx=Lx/nx;dy=Ly/ny;

% Zero various arrys
u=zeros(nx+1,ny+2); v=zeros(nx+2,ny+1); p=zeros(nx+2,ny+2);
ut=zeros(nx+1,ny+2); vt=zeros(nx+2,ny+1); tmp1=zeros(nx+2,ny+2);
uu=zeros(nx+1,ny+1); vv=zeros(nx+1,ny+1); tmp2=zeros(nx+2,ny+2);
fx=zeros(nx+2,ny+2); fy=zeros(nx+2,ny+2);
dk=zeros(1,nx+2); dl=zeros(1,ny+2); dkh=zeros(1,nx+1); dlh=zeros(1,ny+1);

% Set the stretched grid
for i=1:nx+2,s=Lx*(i-1.5)/(nx); x(i)=1.5*s*(0.5-s)*(1-s)+s;end;
for j=1:ny+2,s=Ly*(j-1.5)/(ny); y(j)=1.5*s*(0.5-s)*(1-s)+s;end;
for i=1:nx+1, xh(i)=0.5*(x(i+1)+x(i));end;
for j=1:ny+1, yh(j)=0.5*(y(j+1)+y(j));end
for i=1:nx+1,dkh(i)=x(i+1)-x(i);end;
for j=1:ny+1,dlh(j)=y(j+1)-y(j);end;
```

```
for i=2:nx+1,dk(i)=xh(i)-xh(i-1);end;
for j=2:ny+1,dl(j)=yh(j)-yh(j-1);end;
dk(1)=dk(2);dk(nx+2)=dk(nx+1); dl(1)=dl(2);dl(ny+2)=dl(ny+1);

% Set density and viscosity in the domain and the drop
r=zeros(nx+2,ny+2)+rho1;m=zeros(nx+2,ny+2)+m1;
for i=2:nx+1,for j=2:ny+1;
  if ( (x(i)-xc)^2+(y(j)-yc)^2 < rad^2), r(i,j)=rho2;m(i,j)=m2;end,
end,end

%================== SETUP THE FRONT ===================
Nf=100; xf=zeros(1,Nf+2);yf=zeros(1,Nf+2);
uf=zeros(1,Nf+2);vf=zeros(1,Nf+2);
tx=zeros(1,Nf+2);ty=zeros(1,Nf+2);

for l=1:Nf+2, xf(l)=xc-rad*sin(2.0*pi*(l-1)/(Nf));
               yf(l)=yc+rad*cos(2.0*pi*(l-1)/(Nf));end
%------------find the mapped front coordinates--------
sxf=zeros(1,Nf+2);syf=zeros(1,Nf+2);
for l=1:Nf+2,
   for i=1:nx+1
     if xf(l) > x(i+1), % DO NOTHING
     else
       sxf(l)=i*(x(i+1)-xf(l))/dkh(i)+(i+1)*(xf(l)-x(i))/dkh(i);break
     end
end
for j=1:ny+1
     if yf(l) >= y(j+1), % DO NOTHING
     else
       syf(l)=j*(y(j+1)-yf(l))/dlh(j)+(j+1)*(yf(l)-y(j))/dlh(j);break
     end
   end
end
%================== START TIME LOOP=====================================
 for is=1:nstep,is
%------------------ FIND SURFACE TENSION --------------
    fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
     for l=1:Nf+1,
        ds=sqrt((xf(l+1)-xf(l))^2+(yf(l+1)-yf(l))^2);
        tx(l)=(xf(l+1)-xf(l))/ds;
        ty(l)=(yf(l+1)-yf(l))/ds; % Tangent vectors
    end
    tx(Nf+2)=tx(2);ty(Nf+2)=ty(2);

for l=2:Nf+1
    nfx=sigma*(tx(l)-tx(l-1));nfy=sigma*(ty(l)-ty(l-1));

        ip=floor(sxf(l)-0.5);jp=floor(syf(l));
        ax=sxf(l)-0.5-ip;ay=syf(l)-jp;
        fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dkh(ip)/dl(jp);
        fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dkh(ip+1)/dl(jp);
        fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dkh(ip)/dl(jp+1);
        fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dkh(ip+1)/dl(jp+1);

        ip=floor(sxf(l));jp=floor(syf(l)-0.5);
        ax=sxf(l)-ip;ay=syf(l)-0.5-jp;
        fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dk(ip)/dlh(jp);
        fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dk(ip+1)/dlh(jp);
```

```
        fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dk(ip)/dlh(jp+1);
        fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dk(ip+1)/dlh(jp+1);
    end
%-------------------------------------------------------
    % tangential velocity at boundaries
    u(1:nx+1,1)=2*us-u(1:nx+1,2);u(1:nx+1,ny+2)=2*un-u(1:nx+1,ny+1);
    v(1,1:ny+1)=2*vw-v(2,1:ny+1);v(nx+2,1:ny+1)=2*ve-v(nx+1,1:ny+1);

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity-ADVECTION
      ut(i,j)=u(i,j)+dt*(-0.25*(((u(i+1,j)+u(i,j))^2-(u(i,j)+   ...
          u(i-1,j))^2)/dkh(i)+((u(i,j+1)+u(i,j))*(v(i+1,j)+         ...
          v(i,j))-(u(i,j)+u(i,j-1))*(v(i+1,j-1)+v(i,j-1)))/dl(j))+ ...
             fx(i,j)/(0.5*(r(i+1,j)+r(i,j)))                      ...
           - (1.0 -rro/(0.5*(r(i+1,j)+r(i,j))) )*gx      );
    end,end

    for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity-ADVECTION
      vt(i,j)=v(i,j)+dt*(-0.25*(((u(i,j+1)+u(i,j))*(v(i+1,j)+   ...
          v(i,j))-(u(i-1,j+1)+u(i-1,j))*(v(i,j)+v(i-1,j)))/dk(i)+  ...
          ((v(i,j+1)+v(i,j))^2-(v(i,j)+v(i,j-1))^2)/dlh(j))+        ...
          fy(i,j)/(0.5*(r(i,j+1)+r(i,j)))                      ...
            - (1.0 -rro/(0.5*(r(i,j+1)+r(i,j))) )*gy      );
    end,end

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity-DIFFUSION
      ut(i,j)=ut(i,j)+dt*(...
               (1./dkh(i))*2.*(m(i+1,j)*(1./dk(i+1))*(u(i+1,j)-u(i,j)) -        ...
                  m(i,j)  *(1./dk(i))*(u(i,j)-u(i-1,j)) )                  ...
          +(1./dl(j))*( 0.25*(m(i,j)+m(i+1,j)+m(i+1,j+1)+m(i,j+1))*        ...
            ((1./dlh(j))*(u(i,j+1)-u(i,j)) + (1./dkh(i))*(v(i+1,j)-v(i,j)) ) -  ...
                0.25*(m(i,j)+m(i+1,j)+m(i+1,j-1)+m(i,j-1))*            ...
            ((1./dlh(j-1))*(u(i,j)-u(i,j-1))+ (1./dkh(i-1))*(v(i+1,j-1)- v(i,j-1))) )...
            )/(0.5*(r(i+1,j)+r(i,j))   );
    end,end

   for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity-DIFFUSION
      vt(i,j)=vt(i,j)+dt*(...
          (1./dk(i))*( 0.25*(m(i,j)+m(i+1,j)+m(i+1,j+1)+m(i,j+1))*        ...
           ((1./dlh(j))*(u(i,j+1)-u(i,j)) + (1./dkh(i))*(v(i+1,j)-v(i,j)) ) -  ...
                0.25*(m(i,j)+m(i,j+1)+m(i-1,j+1)+m(i-1,j))*            ...
          ((1./dlh(j))*(u(i-1,j+1)-u(i-1,j))+ (1./dkh(i-1))*(v(i,j)- v(i-1,j))) )...
           +(1./dlh(j))*2.*(m(i,j+1)*(1./dl(j+1))*(v(i,j+1)-v(i,j)) -           ...
                  m(i,j) *(1./dl(j))*(v(i,j)-v(i,j-1)) )                  ...
           )/(0.5*(r(i,j+1)+r(i,j))   );
    end,end
%=======================================================================
    % Compute source term and the coefficient for p(i,j)
    rt=r; lrg=1000;
    rt(1:nx+2,1)=lrg;rt(1:nx+2,ny+2)=lrg;
    rt(1,1:ny+2)=lrg;rt(nx+2,1:ny+2)=lrg;

    for i=2:nx+1,for j=2:ny+1
        tmp1(i,j)= (0.5/dt)*( (ut(i,j)-ut(i-1,j))/dk(i)+(vt(i,j)-vt(i,j-1))/dl(j) );
        tmp2(i,j)=1.0/( (1./dk(i))*( 1./(dkh(i)*(rt(i+1,j)+rt(i,j)))+...
                                     1./(dkh(i-1)*(rt(i-1,j)+rt(i,j)))  )+...
                        (1./dl(j))*(1./(dlh(j)*(rt(i,j+1)+rt(i,j)))+...
                                    1./(dlh(j-1)*(rt(i,j-1)+rt(i,j)))   )   );
    end,end
```

```
    for it=1:maxit               % SOLVE FOR PRESSURE
      oldArray=p;
      for i=2:nx+1,for j=2:ny+1
          p(i,j)=(1.0-beta)*p(i,j)+beta* tmp2(i,j)*(...
          (1./dk(i))*( p(i+1,j)/(dkh(i)*(rt(i+1,j)+rt(i,j)))+...
                       p(i-1,j)/(dkh(i-1)*(rt(i-1,j)+rt(i,j))) )+...
          (1./dl(j))*( p(i,j+1)/(dlh(j)*(rt(i,j+1)+rt(i,j)))+...
                       p(i,j-1)/(dlh(j-1)*(rt(i,j-1)+rt(i,j))) ) - tmp1(i,j));
      end,end
      if max(max(abs(oldArray-p))) <maxError, break,end
    end

    for i=2:nx,for j=2:ny+1   % CORRECT THE u-velocity
          u(i,j)=ut(i,j)-dt*(2.0/dkh(i))*(p(i+1,j)-p(i,j))/(r(i+1,j)+r(i,j));
    end,end

    for i=2:nx+1,for j=2:ny   % CORRECT THE v-velocity
          v(i,j)=vt(i,j)-dt*(2.0/dlh(j))*(p(i,j+1)-p(i,j))/(r(i,j+1)+r(i,j));
    end,end
%================== ADVECT FRONT =====================
for l=2:Nf+1
        ip=floor(sxf(l)-0.5);jp=floor(syf(l)); ax=sxf(l)-0.5-ip;ay=syf(l)-jp;
        uf(l)=(1.0-ax)*(1.0-ay)*u(ip,jp)/dkh(ip)+ax*(1.0-ay)*u(ip+1,jp)/dkh(ip+1)+...
                (1.0-ax)*ay*u(ip,jp+1)/dkh(ip)+ax*ay*u(ip+1,jp+1)/dkh(ip+1);

        ip=floor(sxf(l));jp=floor(syf(l)-0.5); ax=sxf(l)-ip;ay=syf(l)-0.5-jp;
    vf(l)=(1.0-ax)*(1.0-ay)*v(ip,jp)/dlh(jp)+ax*(1.0-ay)*v(ip+1,jp)/dlh(jp)+...
                (1.0-ax)*ay*v(ip,jp+1)/dlh(jp+1)+ax*ay*v(ip+1,jp+1)/dlh(jp+1);
    end
for i=2:Nf+1, sxf(i)=sxf(i)+dt*uf(i); syf(i)=syf(i)+dt*vf(i);end  %MOVE THE FRONT
sxf(1)=sxf(Nf+1);syf(1)=syf(Nf+1);sxf(Nf+2)=sxf(2);syf(Nf+2)=syf(2);

%------------ Add points to the front ------------
   sxfold=sxf;syfold=syf; j=1;
   for l=2:Nf+1
      ds=sqrt( (sxfold(l)-sxf(j))^2 + (syfold(l)-syf(j))^2);
      if (ds > 0.5)
        j=j+1;sxf(j)=0.5*(sxfold(l)+sxf(j-1));syf(j)=0.5*(syfold(l)+syf(j-1));
        j=j+1;sxf(j)=sxfold(l);syf(j)=syfold(l);
      elseif (ds < 0.25)
        % DO NOTHING!
      else
       j=j+1;sxf(j)=sxfold(l);syf(j)=syfold(l);
      end
   end
   Nf=j-1;
   sxf(1)=sxf(Nf+1);syf(1)=syf(Nf+1);sxf(Nf+2)=sxf(2);syf(Nf+2)=syf(2);

%-----------find the coordinates in real space-------------
   for l=2:Nf+1
      ip=floor(sxf(l));jp=floor(syf(l));
      xf(l)=(ip+1-sxf(l))*x(ip)+(sxf(l)-ip)*x(ip+1);
      yf(l)=(jp+1-syf(l))*y(jp)+(syf(l)-jp)*y(jp+1);
   end
   xf(1)=xf(Nf+1);yf(1)=yf(Nf+1);xf(Nf+2)=xf(2);yf(Nf+2)=yf(2);

 %------------ distribute gradient --------------
   fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
```

```
   for l=2:Nf+1
       nfx=-0.5*(yf(l+1)-yf(l-1))*(rho2-rho1);
       nfy=0.5*(xf(l+1)-xf(l-1))*(rho2-rho1);  % Normal vector

       ip=floor(sxf(l)-0.5);jp=floor(syf(l));
       ax=sxf(l)-0.5-ip;ay=syf(l)-jp;
       fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dkh(ip)/dl(jp);
       fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dkh(ip+1)/dl(jp);
       fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dkh(ip)/dl(jp+1);
       fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dkh(ip+1)/dl(jp+1);

       ip=floor(sxf(l));jp=floor(syf(l)-0.5);
       ax=sxf(l)-ip;ay=syf(l)-0.5-jp;
       fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dk(ip)/dlh(jp);
       fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dk(ip+1)/dlh(jp);
       fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dk(ip)/dlh(jp+1);
       fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dk(ip+1)/dlh(jp+1);
   end
%------------ construct the density --------------
   for iter=1:maxit
     oldArray=r;
     for i=2:nx+1,for j=2:ny+1
       r(i,j)=(1.0-beta)*r(i,j)+beta*0.25*...
                (r(i+1,j)+r(i-1,j)+r(i,j+1)+r(i,j-1)+...
                dkh(i-1)*fx(i-1,j)-dkh(i)*fx(i,j)+...
                dlh(j-1)*fy(i,j-1)-dlh(j)*fy(i,j));
     end,end
     if max(max(abs(oldArray-r))) <maxError, break,end
   end
%------------ update the viscosity --------------
   m=zeros(nx+2,ny+2)+m1;
   for i=2:nx+1,for j=2:ny+1
      m(i,j)=m1+(m2-m1)*(r(i,j)-rho1)/(rho2-rho1);
   end,end
%================================================================================
   time=time+dt                    % plot the results
   uu(1:nx+1,1:ny+1)=0.5*(u(1:nx+1,2:ny+2)+u(1:nx+1,1:ny+1));
   vv(1:nx+1,1:ny+1)=0.5*(v(2:nx+2,1:ny+1)+v(1:nx+1,1:ny+1));
   hold off,contour(x,y,flipud(rot90(r))),axis equal,axis([0 Lx 0 Ly]);
   hold on;quiver(xh,yh,flipud(rot90(uu)),flipud(rot90(vv)),'r');
   plot(xf(1:Nf),yf(1:Nf),'k','linewidth',5);pause(0.01)
end
```

5.1.4 Results

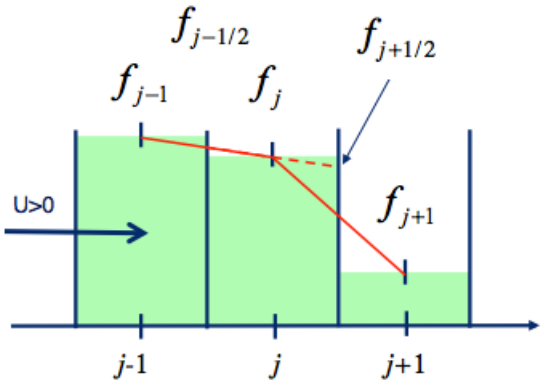

Figure 5.4: The interpolation used by ENO.

5.2 Advection by ENO

The centered difference advection scheme used so far is unstable in the absence of viscous diffusion and as the viscosity becomes smaller, we need to take smaller and smaller time steps to allow diffusion to stabilize it. Furthermore, although it is very accurate for fully resolved flows, it falls apart rapidly if the resolution is marginal. As the viscosity is reduced, the likelihood of small and poorly resolved scales increases so the centered difference scheme typically is not a good choice for high Reynolds number flows.

Here we introduce a more robust advection scheme whose stability is controlled only by the flow velocity and the grid size ($|\mathbf{u}|_{max}\Delta t \leq (\Delta x, \Delta y)_{min}$) and who is more robust for low viscosity flow. Several such schemes exist, but we use a simple second-order Runge-Kutta/ENO introduced by [**?**].

THE FOLLOWING NEEDS TO BE UPDATED. We first rewrite the advection part of the momentum equations as $\phi_t = L\phi$. Assuming that $\phi^n_{i,j}$ and $\mathbf{u}^n_{i,j}$ are valid discrete values defined at $t = t^n$, $x = x_i$, and $y = y_j$, we advance the solution to $t = t^{n+1}$ by first finding a predicted value for the level set function

$$\phi^*_{i,j} = \phi^n_{i,j} + \Delta t L\phi^n \tag{5.23}$$

and then correcting the solution by

$$\phi^{n+1}_{i,j} = \phi^n_{i,j} + \frac{\Delta t}{2}(L\phi^n + L\phi^*). \tag{5.24}$$

The operator $L\phi$ is discretized by

$$L\phi = -u_{i,j}\frac{\phi_{i+1/2,j} - \phi_{i-1/2,j}}{\Delta x} - v_{i,j}\frac{\phi_{i,j+1/2} - \phi_{i,j-1/2}}{\Delta y}, \tag{5.25}$$

where the value of ϕ at the cell boundaries is found by

$$\phi_{i+1/2,j} = \begin{cases} \phi_{i,j} + \frac{1}{2}M(D^+_x\phi_{i,j}, D^-_x\phi_{i,j}), & \frac{1}{2}(u_{i+1,j} + u_{i,j}) > 0 \\ \phi_{i+1,j} - \frac{1}{2}M(D^+_x\phi_{i+1,j}, D^-_x\phi_{i+1,j}), & \frac{1}{2}(u_{i+1,j} + u_{i,j}) < 0. \end{cases} \tag{5.26}$$

M is a switch defined by

$$M(a,b) = \begin{cases} a, & |a| < |b| \\ b, & |b| \le |a| \end{cases} = \frac{1}{2}\Big[(a-b)\ \mathrm{sign}(|a|-|b|) + a + b\Big] \tag{5.27}$$

and the differences are found by

$$\begin{aligned} D_x^+ \phi_{i,j} &= \phi_{i+1,j} - \phi_{i,j} \\ D_x^- \phi_{i,j} &= \phi_{i,j} - \phi_{i-1,j}. \end{aligned} \tag{5.28}$$

The equation for $\phi_{i,j+1/2}$ is similar.

5.3 Advecting the Front Across Periodic Boundaries

FROM TSZ—NEEDS TO BE UPDATED In many cases we wish to simulate periodic domains where the front can move out of the domain on one side and reappear in through the other side. This can be done in a very simple way by recognizing that there is no need for the front to occupy the same period as the fixed grid. All that is needed is to correctly identify the grid point that corresponds to a given front position. A slight modification of the operation in chapter 2 accomplishes this:

$$i = \mathrm{FLOOR}\left(\mathrm{MOD}(x_f, L_x) \times N_x / L_x\right). \tag{5.29}$$

Here MOD(A,B) is an operation that gives the remainder after A has been divided by B. For closed fronts, such as those representing the surface of a bubble or a drop, nothing else needs to be changed. For periodic fronts, the end point in one period is connected to the first point in the next period, but only one period is actually computed. When computing the length of such elements, or a curve is fitted through the end points, it is therefore necessary to correct for the positions of the points in the next period.

5.3.1 Stretched Grid plus ENO Code

This code can be downloaded from: code5.m. We also need the minabs.m function

```
LOOKS LIKE THIS CODE IS ONLY FIRST ORDER!!!!!

%========================================================================
% A very simple Navier-Stokes solver for a drop falling in a rectangular
% box. A forward in time, centered in space discretization is used.
% The density is advected by a front tracking scheme.
% This version uses a stretched grid and ENO for the advection
%========================================================================
%domain size and physical variables
Lx=1.0;Ly=1.0;gx=0.0;gy=100.0; rho1=1.0; rho2=2.0;
m1=0.01; m2=0.05; sigma=10; rro=rho1;
un=0;us=0;ve=0;vw=0;time=0.0;
rad=0.15;xc=0.5;yc=0.7; % Initial drop size and location

% Numerical variables
nx=32;ny=32;dt=0.00125;nstep=300;maxit=200;maxError=0.001;beta=1.2;

% Zero various arrys
u=zeros(nx+1,ny+2); v=zeros(nx+2,ny+1); p=zeros(nx+2,ny+2);
ut=zeros(nx+1,ny+2); vt=zeros(nx+2,ny+1); tmp1=zeros(nx+2,ny+2);
uu=zeros(nx+1,ny+1); vv=zeros(nx+1,ny+1); tmp2=zeros(nx+2,ny+2);
fx=zeros(nx+2,ny+2); fy=zeros(nx+2,ny+2);
dk=zeros(1,nx+2); dl=zeros(1,ny+2); dkh=zeros(1,nx+1); dlh=zeros(1,ny+1);
ux=zeros(nx+2,ny+2); uy=zeros(nx+2,ny+2);
vx=zeros(nx+2,ny+2); vy=zeros(nx+2,ny+2);

% Set the stretched grid
for i=1:nx+2, s=Lx*(i-1.5)/(nx); x(i)=1.5*s*(0.5-s)*(1-s)+s; end;
for j=1:ny+2, s=Ly*(j-1.5)/(ny); y(j)=1.5*s*(0.5-s)*(1-s)+s; end;
for i=1:nx+1, xh(i)=0.5*(x(i+1)+x(i)); end;
for j=1:ny+1, yh(j)=0.5*(y(j+1)+y(j)); end
for i=1:nx+1, dkh(i)=x(i+1)-x(i); end;
for j=1:ny+1, dlh(j)=y(j+1)-y(j); end;
for i=2:nx+1, dk(i)=xh(i)-xh(i-1); end;
for j=2:ny+1, dl(j)=yh(j)-yh(j-1); end;
dk(1)=dk(2);dk(nx+2)=dk(nx+1); dl(1)=dl(2);dl(ny+2)=dl(ny+1);

% Set density and viscosity in the domain and the drop
r=zeros(nx+2,ny+2)+rho1;m=zeros(nx+2,ny+2)+m1;
for i=2:nx+1,for j=2:ny+1;
  if ( (x(i)-xc)^2+(y(j)-yc)^2 < rad^2), r(i,j)=rho2;m(i,j)=m2; end,
end,end

%================== SETUP THE FRONT ==================
Nf=100; xf=zeros(1,Nf+2);yf=zeros(1,Nf+2);
uf=zeros(1,Nf+2);vf=zeros(1,Nf+2);
tx=zeros(1,Nf+2);ty=zeros(1,Nf+2);

for l=1:Nf+2, xf(l)=xc-rad*sin(2.0*pi*(l-1)/(Nf));
              yf(l)=yc+rad*cos(2.0*pi*(l-1)/(Nf));end
%------------find the mapped front coordinates--------
sxf=zeros(1,Nf+2);syf=zeros(1,Nf+2);
for l=1:Nf+2,
   for i=1:nx+1
     if xf(l) > x(i+1), % DO NOTHING
     else
       sxf(l)=i*(x(i+1)-xf(l))/dkh(i)+(i+1)*(xf(l)-x(i))/dkh(i);break
```

```
      end
    end
    for j=1:ny+1
      if yf(l) >= y(j+1), % DO NOTHING
      else
        syf(l)=j*(y(j+1)-yf(l))/dlh(j)+(j+1)*(yf(l)-y(j))/dlh(j);break
      end
    end
end
%================== START TIME LOOP=====================================
for is=1:nstep,is
%------------------ FIND SURFACE TENSION --------------
    fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
     for l=1:Nf+1,
        ds=sqrt((xf(l+1)-xf(l))^2+(yf(l+1)-yf(l))^2);
        tx(l)=(xf(l+1)-xf(l))/ds;
        ty(l)=(yf(l+1)-yf(l))/ds; % Tangent vectors
    end
    tx(Nf+2)=tx(2);ty(Nf+2)=ty(2);

for l=2:Nf+1
    nfx=sigma*(tx(l)-tx(l-1));nfy=sigma*(ty(l)-ty(l-1));

        ip=floor(sxf(l)-0.5);jp=floor(syf(l));
        ax=sxf(l)-0.5-ip;ay=syf(l)-jp;
        fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dkh(ip)/dl(jp);
        fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dkh(ip+1)/dl(jp);
        fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dkh(ip)/dl(jp+1);
        fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dkh(ip+1)/dl(jp+1);

        ip=floor(sxf(l));jp=floor(syf(l)-0.5);
        ax=sxf(l)-ip;ay=syf(l)-0.5-jp;
        fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dk(ip)/dlh(jp);
        fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dk(ip+1)/dlh(jp);
        fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dk(ip)/dlh(jp+1);
        fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dk(ip+1)/dlh(jp+1);
    end
%-----------------------------------------------------------
    % tangential velocity at boundaries
    u(1:nx+1,1)=2*us-u(1:nx+1,2);u(1:nx+1,ny+2)=2*un-u(1:nx+1,ny+1);
    v(1,1:ny+1)=2*vw-v(2,1:ny+1);v(nx+2,1:ny+1)=2*ve-v(nx+1,1:ny+1);

    for i=2:nx-1,for j=2:ny+1;     % finding the ENO values for the face velocities
      ss=0.5*sign(u(i+1,j)+u(i,j));
      ux(i+1,j)=(0.5+ss)*(u(i,j)+0.5*minabs((u(i+1,j)-u(i,j)),(u(i,j)-u(i-1,j))) )...
           +(0.5-ss)*(u(i+1,j)-0.5*minabs((u(i+2,j)-u(i+1,j)),(u(i+1,j)-u(i,j))) );
    end,end
    ux(2,2:ny+1)=0.5*u(2,2:ny+1)+u(1,2:ny+1); ux(nx+1,2:ny+1)=0.5*u(nx,2:ny+1)+u(nx+1,2:ny+1); %Bdry

    for i=2:nx,for j=3:ny;
      ss=0.5*sign(v(i+1,j)+v(i,j));
      uy(i,j+1)=(0.5+ss)*(u(i,j)+0.5*minabs((u(i,j+1)-u(i,j)),(u(i,j)-u(i,j-1))) )...
           +(0.5-ss)*(u(i,j+1)-0.5*minabs((u(i,j+2)-u(i,j+1)),(u(i,j+1)-u(i,j))) );
    end,end
    uy(1:nx+1,2)=0.0;uy(1:nx+1,ny+1)=0.0; % Bottom and Top

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity-ADVECTION
      ut(i,j)=u(i,j)+dt*(-(u(i,j)*(ux(i+1,j)-ux(i,j))/dkh(i)+...
```

```
      0.25*(v(i,j-1)+v(i+1,j-1)+v(i+1,j)+v(i,j))*(uy(i,j+1)-uy(i,j))/dl(j) )+...
            fx(i,j)/(0.5*(r(i+1,j)+r(i,j)))   ...
          - (1.0 -rro/(0.5*(r(i+1,j)+r(i,j))) )*gx      );
    end,end

    for i=3:nx,for j=2:ny;     % finding the ENO values for the face velocities
      ss=0.5*sign(u(i,j+1)+u(i,j));
      vx(i+1,j)=(0.5+ss)*(v(i,j)+0.5*minabs((v(i+1,j)-v(i,j)),(v(i,j)-v(i-1,j))) )...
           +(0.5-ss)*(v(i+1,j)-0.5*minabs((v(i+2,j)-v(i+1,j)),(v(i+1,j)-v(i,j))) );
    end,end
    vx(2,1:ny+1)=0.0;vx(nx+1,1:ny+1)=0.0; % Left and Right

    for i=2:nx+1,for j=2:ny-1;
      ss=0.5*sign(v(i,j+1)+v(i,j));
      vy(i,j+1)=(0.5+ss)*(v(i,j)+0.5*minabs((v(i,j+1)-v(i,j)),(v(i,j)-v(i,j-1))) )...
           +(0.5-ss)*(v(i,j+1)-0.5*minabs((v(i,j+2)-v(i,j+1)),(v(i,j+1)-v(i,j))) );
    end,end
    vy(2:nx+1,2)=0.5*v(2:nx+1,2)+v(2:nx+1,1); vy(2:nx+1,ny+1)=0.5*v(2:nx+1,ny)+v(2:nx+1,ny+1); %Bdry

    for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity-ADVECTION
      vt(i,j)=v(i,j)+dt*(-(0.25*(u(i-1,j)+u(i,j)+u(i,j+1)+...
              u(i-1,j+1))*(vx(i+1,j)-vx(i,j))/dk(i)+  ...
          v(i,j)*(vy(i,j+1)-vy(i,j))/dlh(j))+        ...
          fy(i,j)/(0.5*(r(i,j+1)+r(i,j)))                    ...
            - (1.0 -rro/(0.5*(r(i,j+1)+r(i,j))) )*gy      );
    end,end

    for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity-DIFFUSION
      ut(i,j)=ut(i,j)+dt*(...
               (1./dkh(i))*2.*(m(i+1,j)*(1./dk(i+1))*(u(i+1,j)-u(i,j)) -        ...
                  m(i,j)  *(1./dk(i))*(u(i,j)-u(i-1,j)) )                  ...
         +(1./dl(j))*( 0.25*(m(i,j)+m(i+1,j)+m(i+1,j+1)+m(i,j+1))*        ...
           ((1./dlh(j))*(u(i,j+1)-u(i,j)) + (1./dkh(i))*(v(i+1,j)-v(i,j)) ) -  ...
                0.25*(m(i,j)+m(i+1,j)+m(i+1,j-1)+m(i,j-1))*            ...
           ((1./dlh(j-1))*(u(i,j)-u(i,j-1))+ (1./dkh(i-1))*(v(i+1,j-1)- v(i,j-1))) )...
           )/(0.5*(r(i+1,j)+r(i,j)))   );
    end,end

   for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity-DIFFUSION
      vt(i,j)=vt(i,j)+dt*(...
          (1./dk(i))*( 0.25*(m(i,j)+m(i+1,j)+m(i+1,j+1)+m(i,j+1))*        ...
           ((1./dlh(j))*(u(i,j+1)-u(i,j)) + (1./dkh(i))*(v(i+1,j)-v(i,j)) ) -  ...
                0.25*(m(i,j)+m(i,j+1)+m(i-1,j+1)+m(i-1,j))*            ...
          ((1./dlh(j))*(u(i-1,j+1)-u(i-1,j))+ (1./dkh(i-1))*(v(i,j)- v(i-1,j))) )...
           +(1./dlh(j))*2.*(m(i,j+1)*(1./dl(j+1))*(v(i,j+1)-v(i,j)) -           ...
                  m(i,j) *(1./dl(j))*(v(i,j)-v(i,j-1)) )                  ...
           )/(0.5*(r(i,j+1)+r(i,j)))   );
    end,end
%=========================================================================
    % Compute source term and the coefficient for p(i,j)
    rt=r; lrg=1000;
    rt(1:nx+2,1)=lrg;rt(1:nx+2,ny+2)=lrg;
    rt(1,1:ny+2)=lrg;rt(nx+2,1:ny+2)=lrg;

    for i=2:nx+1,for j=2:ny+1
        tmp1(i,j)= (0.5/dt)*( (ut(i,j)-ut(i-1,j))/dk(i)+(vt(i,j)-vt(i,j-1))/dl(j) );
        tmp2(i,j)=1.0/( (1./dk(i))*( 1./(dkh(i)*(rt(i+1,j)+rt(i,j)))+...
                                     1./(dkh(i-1)*(rt(i-1,j)+rt(i,j)))  )+...
```

```
                        (1./dl(j))*(1./(dlh(j)*(rt(i,j+1)+rt(i,j)))+...
                                    1./(dlh(j-1)*(rt(i,j-1)+rt(i,j)))   )   );
    end,end

    for it=1:maxit                % SOLVE FOR PRESSURE
      oldArray=p;
      for i=2:nx+1,for j=2:ny+1
          p(i,j)=(1.0-beta)*p(i,j)+beta* tmp2(i,j)*(...
          (1./dk(i))*( p(i+1,j)/(dkh(i)*(rt(i+1,j)+rt(i,j)))+...
                       p(i-1,j)/(dkh(i-1)*(rt(i-1,j)+rt(i,j))) )+...
          (1./dl(j))*( p(i,j+1)/(dlh(j)*(rt(i,j+1)+rt(i,j)))+...
                       p(i,j-1)/(dlh(j-1)*(rt(i,j-1)+rt(i,j))) ) - tmp1(i,j));
      end,end
      if max(max(abs(oldArray-p))) <maxError, break,end
    end

    for i=2:nx,for j=2:ny+1   % CORRECT THE u-velocity
          u(i,j)=ut(i,j)-dt*(2.0/dkh(i))*(p(i+1,j)-p(i,j))/(r(i+1,j)+r(i,j));
    end,end

    for i=2:nx+1,for j=2:ny   % CORRECT THE v-velocity
          v(i,j)=vt(i,j)-dt*(2.0/dlh(j))*(p(i,j+1)-p(i,j))/(r(i,j+1)+r(i,j));
     end,end
%================== ADVECT FRONT =====================
for l=2:Nf+1
        ip=floor(sxf(l)-0.5);jp=floor(syf(l)); ax=sxf(l)-0.5-ip;ay=syf(l)-jp;
        uf(l)=(1.0-ax)*(1.0-ay)*u(ip,jp)/dkh(ip)+ax*(1.0-ay)*u(ip+1,jp)/dkh(ip+1)+...
                (1.0-ax)*ay*u(ip,jp+1)/dkh(ip)+ax*ay*u(ip+1,jp+1)/dkh(ip+1);

        ip=floor(sxf(l));jp=floor(syf(l)-0.5); ax=sxf(l)-ip;ay=syf(l)-0.5-jp;
    vf(l)=(1.0-ax)*(1.0-ay)*v(ip,jp)/dlh(jp)+ax*(1.0-ay)*v(ip+1,jp)/dlh(jp)+...
                (1.0-ax)*ay*v(ip,jp+1)/dlh(jp+1)+ax*ay*v(ip+1,jp+1)/dlh(jp+1);
    end
for i=2:Nf+1, sxf(i)=sxf(i)+dt*uf(i); syf(i)=syf(i)+dt*vf(i);end  %MOVE THE FRONT
sxf(1)=sxf(Nf+1);syf(1)=syf(Nf+1);sxf(Nf+2)=sxf(2);syf(Nf+2)=syf(2);

%------------ Add points to the front ------------
   sxfold=sxf;syfold=syf; j=1;
   for l=2:Nf+1
      ds=sqrt( (sxfold(l)-sxf(j))^2 + (syfold(l)-syf(j))^2);
      if (ds > 0.5)
        j=j+1;sxf(j)=0.5*(sxfold(l)+sxf(j-1));syf(j)=0.5*(syfold(l)+syf(j-1));
        j=j+1;sxf(j)=sxfold(l);syf(j)=syfold(l);
      elseif (ds < 0.25)
        % DO NOTHING!
      else
       j=j+1;sxf(j)=sxfold(l);syf(j)=syfold(l);
      end
   end
   Nf=j-1;
   sxf(1)=sxf(Nf+1);syf(1)=syf(Nf+1);sxf(Nf+2)=sxf(2);syf(Nf+2)=syf(2);
%-----------find the coordinates in real space-------------
   for l=2:Nf+1
      ip=floor(sxf(l));jp=floor(syf(l));
      xf(l)=(ip+1-sxf(l))*x(ip)+(sxf(l)-ip)*x(ip+1);
      yf(l)=(jp+1-syf(l))*y(jp)+(syf(l)-jp)*y(jp+1);
   end
   xf(1)=xf(Nf+1);yf(1)=yf(Nf+1);xf(Nf+2)=xf(2);yf(Nf+2)=yf(2);
```

```
%------------ distribute gradient --------------
   fx=zeros(nx+2,ny+2);fy=zeros(nx+2,ny+2);  % Set fx & fy to zero
   for l=2:Nf+1
       nfx=-0.5*(yf(l+1)-yf(l-1))*(rho2-rho1);
       nfy=0.5*(xf(l+1)-xf(l-1))*(rho2-rho1);  % Normal vector

       ip=floor(sxf(l)-0.5);jp=floor(syf(l));
       ax=sxf(l)-0.5-ip;ay=syf(l)-jp;
       fx(ip,jp)    =fx(ip,jp)+(1.0-ax)*(1.0-ay)*nfx/dkh(ip)/dl(jp);
       fx(ip+1,jp)  =fx(ip+1,jp)+ax*(1.0-ay)*nfx/dkh(ip+1)/dl(jp);
       fx(ip,jp+1)  =fx(ip,jp+1)+(1.0-ax)*ay*nfx/dkh(ip)/dl(jp+1);
       fx(ip+1,jp+1)=fx(ip+1,jp+1)+ax*ay*nfx/dkh(ip+1)/dl(jp+1);

       ip=floor(sxf(l));jp=floor(syf(l)-0.5);
       ax=sxf(l)-ip;ay=syf(l)-0.5-jp;
       fy(ip,jp)    =fy(ip,jp)+(1.0-ax)*(1.0-ay)*nfy/dk(ip)/dlh(jp);
       fy(ip+1,jp)  =fy(ip+1,jp)+ax*(1.0-ay)*nfy/dk(ip+1)/dlh(jp);
       fy(ip,jp+1)  =fy(ip,jp+1)+(1.0-ax)*ay*nfy/dk(ip)/dlh(jp+1);
       fy(ip+1,jp+1)=fy(ip+1,jp+1)+ax*ay*nfy/dk(ip+1)/dlh(jp+1);
   end
%------------ construct the density --------------
   r=zeros(nx+2,ny+2)+rho1;
   for iter=1:maxit
     oldArray=r;
     for i=2:nx+1,for j=2:ny+1
       r(i,j)=0.25*(r(i+1,j)+r(i-1,j)+r(i,j+1)+r(i,j-1)+...
                dkh(i-1)*fx(i-1,j)-dkh(i)*fx(i,j)+...
                dlh(j-1)*fy(i,j-1)-dlh(j)*fy(i,j));
     end,end
     if max(max(abs(oldArray-r))) <maxError, break,end
   end
%------------ update the viscosity --------------
   m=zeros(nx+2,ny+2)+m1;
   for i=2:nx+1,for j=2:ny+1
      m(i,j)=m1+(m2-m1)*(r(i,j)-rho1)/(rho2-rho1);
   end,end
%=========================================================================
   time=time+dt                    % plot the results
   uu(1:nx+1,1:ny+1)=0.5*(u(1:nx+1,2:ny+2)+u(1:nx+1,1:ny+1));
   vv(1:nx+1,1:ny+1)=0.5*(v(2:nx+2,1:ny+1)+v(1:nx+1,1:ny+1));
   hold off,contour(x,y,flipud(rot90(r))),axis equal,axis([0 Lx 0 Ly]);
   hold on;quiver(xh,yh,flipud(rot90(uu)),flipud(rot90(vv)),'r');
   plot(xf(1:Nf),yf(1:Nf),'k','linewidth',5);pause(0.01)
end

function x=minabs(a,b)
x=0.5*(sign(abs(b)-abs(a))*(a-b)+a+b);
```