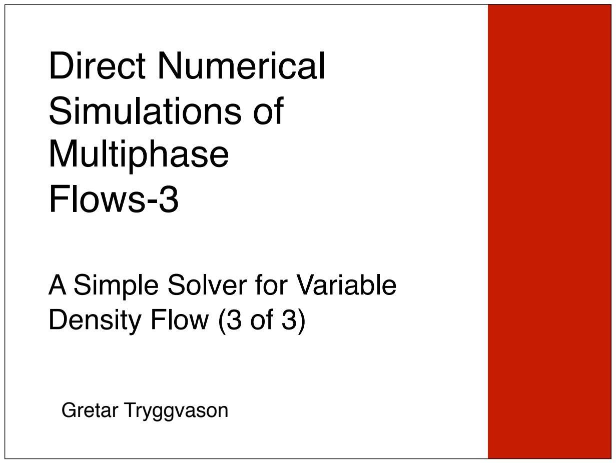
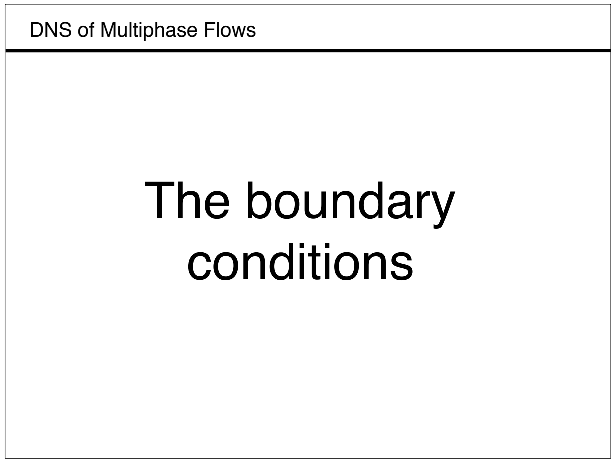


Direct Numerical Simulations of Multiphase Flows-3

A Simple Solver for Variable Density Flow (3 of 3)

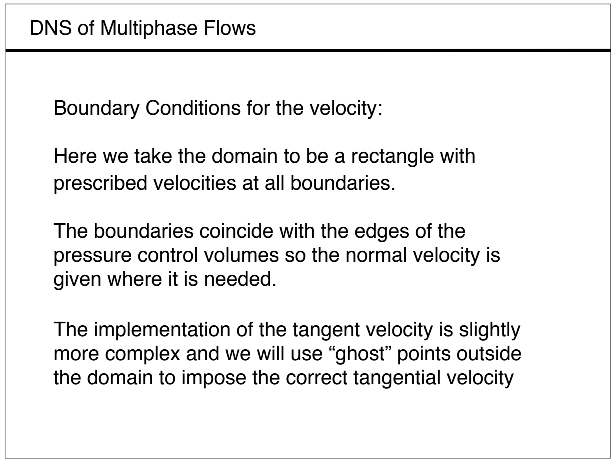
Gretar Tryggvason

1. We have now derived numerical approximations that allow us to update the flow field in time, in the interior of the domain. Before we can do a specific problem we need to include boundary conditions.



The boundary conditions

2. The boundary conditions can vary from problem to problem but here we will focus on walls with given tangential and normal velocities. We often take these velocities to be zero over most of the boundary, representing no-slip rigid walls.



Boundary Conditions for the velocity:

Here we take the domain to be a rectangle with prescribed velocities at all boundaries.

The boundaries coincide with the edges of the pressure control volumes so the normal velocity is given where it is needed.

The implementation of the tangent velocity is slightly more complex and we will use "ghost" points outside the domain to impose the correct tangential velocity

3. In our case the boundaries are straight lines, connected at the corners to enclose a rectangular domain with no in- or out-flow. Thus, the normal velocity is zero for all the boundaries, meaning that the u-velocities are zero on the left and right boundary and the v-velocities are zero at the top and bottom. The tangent boundaries are, however, allowed to move and induce a flow in the domain. Thus, we specify the tangent velocity at each boundary, setting one or more to a non-zero value. The tangent boundary velocity is specified using the ghost points.

DNS of Multiphase Flows

Tangent velocity Boundary Conditions

Velocity of wall is given, u_{wall} (no-slip)

Interpolate linearly

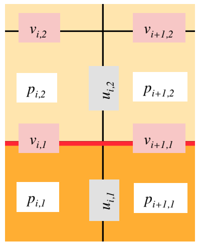
$$u_{wall} = \frac{u_{i,1} + u_{i,2}}{2}$$

Solve for the “ghost” velocity

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

If the wall velocity is zero: $u_{wall} = 0$ then the ghost velocity is a reflection of the interior velocity

$$u_{i,1} = -u_{i,2}$$

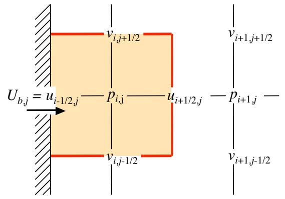


```
% tangential velocity at boundaries
  u(1:nx+1,1)=2*usouth-u(1:nx+1,2);
  u(1:nx+1,ny+2)=2*unorth-u(1:nx+1,ny+1);
  v(1,1:ny+1)=2*vwest-v(2,1:ny+1);
  v(nx+2,1:ny+1)=2*veast-v(nx+1,1:ny+1);
```

4. When we use a staggered grid, the normal velocities are specified directly on the boundary. The tangent velocities are, however, not. To enforce the correct tangent boundary conditions we need to use the ghost point and set the tangent velocity at the ghost points to the correct value. The velocity at the wall can be found by linear interpolation, assuming that the ghost velocity is known. Since the boundary is midway between the center of the ghost cell and the first cell inside the domain it is simply the average. The unknown velocity is, however, the ghost velocity, while the wall velocity is known, so we solve for it and find that the ghost velocity is equal to twice the wall velocity minus the velocity in the first interior cell. Obviously, if the wall velocity is zero, the ghost velocity is simply the negative of the first interior velocity. Thus, once the velocities in the interior have been found, we set the ghost velocity to allow us to take the next time step. Here we have assumed that we are working with the bottom boundary, obviously the derivation for the other boundaries is essentially the same.

DNS of Multiphase Flows

Boundary Conditions for the pressure. Use the normal velocity given at the boundary when we substitute the correction velocity into the mass conservation equation:



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

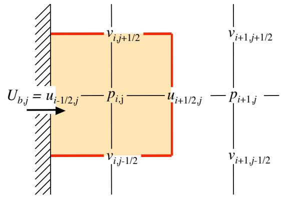
The pressure equation at the boundary is therefore:

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

5-1. In the problem that we will do at the end of this lecture we will be assuming zero flow through the boundary, or zero normal velocity. However, zero normal velocity is a special case of a given normal velocity and since we can allow for arbitrary normal velocity without any additional complexity, we will do so here. Consider the control volume in the figure, where there is a given normal inflow Ub, through the boundary on the left. Since the velocity through the boundary is known, we can modify the incompressibility conditions including Ub. The pressure equation is then derived in the usual way by replacing the unknown velocities in the incompressibility conditions with the expression for the corrected velocity, stating that the corrected, or final, velocities are the predicted velocities plus the discrete pressure gradient. However, since Ub is known, there is no need to replace it and the resulting pressure equation therefore does not include the pressure to the left of P_i,j, and in the divergence of the predicted velocities we replace the predicted velocity at i-1/2,j by U_b.

DNS of Multiphase Flows

Boundary Conditions for the pressure. Use the normal velocity given at the boundary when we substitute the correction velocity into the mass conservation equation:

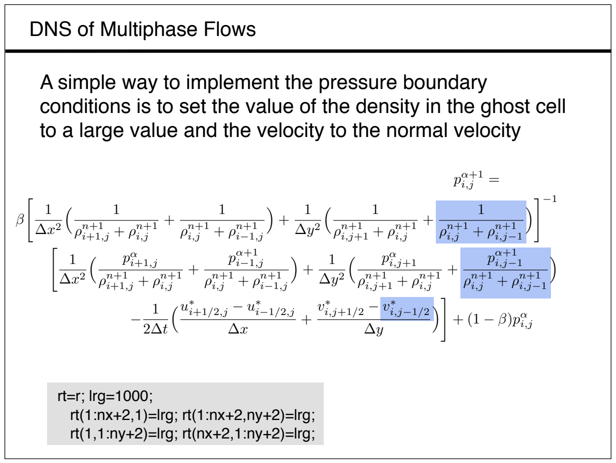


$$\frac{u^{n+1}_{i+1/2,j} - U_{b,j}}{\Delta x} + \frac{v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2}}{\Delta y} = 0$$

The pressure equation at the boundary is therefore:

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

5-2. This is, actually, a rather remarkable result since we do not have to worry about the boundary conditions for the pressure at all. While the pressure boundary condition deserve further discussion, we will not do so here, in the interest of moving on with the development of our code.



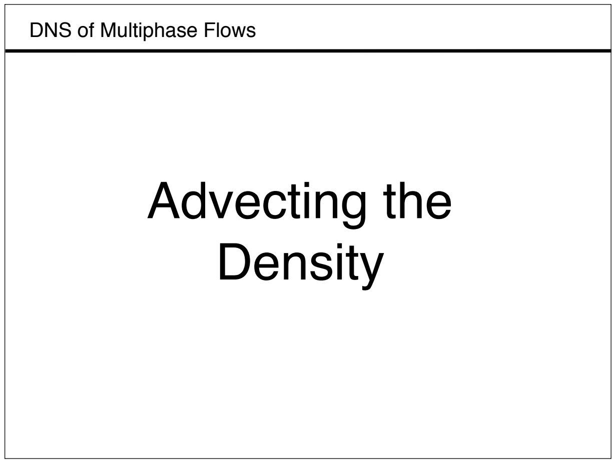
DNS of Multiphase Flows

A simple way to implement the pressure boundary conditions is to set the value of the density in the ghost cell to a large value and the velocity to the normal velocity

$$p_{i,j}^{\alpha+1} = \beta\left[\frac{1}{\Delta x^2}\left(\frac{1}{\rho_{i+1,j}^{n+1}+\rho_{i,j}^{n+1}}+\frac{1}{\rho_{i,j}^{n+1}+\rho_{i-1,j}^{n+1}}\right)+\frac{1}{\Delta y^2}\left(\frac{1}{\rho_{i,j+1}^{n+1}+\rho_{i,j}^{n+1}}+\frac{1}{\rho_{i,j}^{n+1}+\rho_{i,j-1}^{n+1}}\right)\right]^{-1}$$
$$\left[\frac{1}{\Delta x^2}\left(\frac{p_{i+1,j}^{\alpha}}{\rho_{i+1,j}^{n+1}+\rho_{i,j}^{n+1}}+\frac{p_{i-1,j}^{\alpha+1}}{\rho_{i,j}^{n+1}+\rho_{i-1,j}^{n+1}}\right)+\frac{1}{\Delta y^2}\left(\frac{p_{i,j+1}^{\alpha}}{\rho_{i,j+1}^{n+1}+\rho_{i,j}^{n+1}}+\frac{p_{i,j-1}^{\alpha+1}}{\rho_{i,j}^{n+1}+\rho_{i,j-1}^{n+1}}\right)\right.$$
$$\left.-\frac{1}{2\Delta t}\left(\frac{u_{i+1/2,j}^{*}-u_{i-1/2,j}^{*}}{\Delta x}+\frac{v_{i,j+1/2}^{*}-v_{i,j-1/2}^{*}}{\Delta y}\right)\right]+(1-\beta)p_{i,j}^{\alpha}$$

```
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;
```

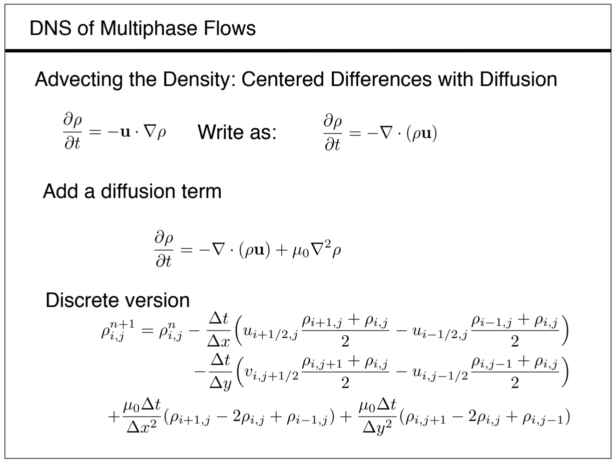
6. The pressure boundary conditions can be implemented in many ways and the most rigorous way is to rewrite the equations for the pressure points next to the boundary, setting the appropriate terms to zero. Doing this does not require any ghost points. Here we do, however, use a simpler approach and set the density in the ghost cells equal to a large value. Since we divide by the densities, this would results in these terms being zero if the density was truly infinitely high. While we do not use an infinitely high value for the density, we can nevertheless make the blue terms very small by a sufficiently high value. The predicted normal velocity at the wall, v at i,j-1/2 or the purple term, in our example, also needs to be set to the correct inflow velocity.



DNS of Multiphase Flows

Advecting the Density

7. For methods with sharp interfaces between fluids of different densities the advection of the sharp interface is the major challenge. We will introduce a method to track the interface in the next lecture, but here we simply assume that the density is updated using the equation stating that density of a material particle is constant.



DNS of Multiphase Flows

Advecting the Density: Centered Differences with Diffusion

$$\frac{\partial \rho}{\partial t} = -\mathbf{u}\cdot\nabla\rho \qquad \text{Write as:} \qquad \frac{\partial \rho}{\partial t} = -\nabla\cdot(\rho\mathbf{u})$$

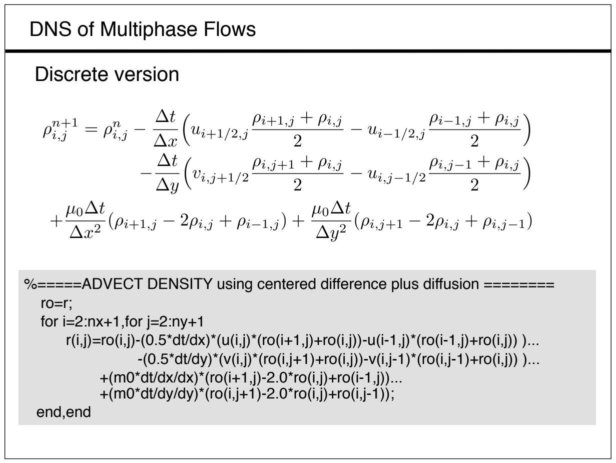
Add a diffusion term

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

Discrete version

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(u_{i+1/2,j}\frac{\rho_{i+1,j}+\rho_{i,j}}{2} - u_{i-1/2,j}\frac{\rho_{i-1,j}+\rho_{i,j}}{2}\right)$$
$$-\frac{\Delta t}{\Delta y}\left(v_{i,j+1/2}\frac{\rho_{i,j+1}+\rho_{i,j}}{2} - u_{i,j-1/2}\frac{\rho_{i,j-1}+\rho_{i,j}}{2}\right)$$
$$+\frac{\mu_0\Delta t}{\Delta x^2}(\rho_{i+1,j}-2\rho_{i,j}+\rho_{i-1,j}) + \frac{\mu_0\Delta t}{\Delta y^2}(\rho_{i,j+1}-2\rho_{i,j}+\rho_{i,j-1})$$

8. The equation stating that the density of a material point is constant says that the time derivative of the density plus the velocity times the density gradient is equal to zero. This equation can be rewritten slightly using that the flow is incompressible to pull the velocity under the divergence. While there are a number of schemes that allow us to write down a stable discrete equation for the advection equation, the simple forwards in time, centered in space scheme used for the Navier-Stokes equations is unconditionally unstable. To be able to use it we add a diffusive term to the equation, taking the diffusion coefficient equal to the fluid viscosity, so that the allowable time step is the same. The new density at the pressure points is therefore the old density minus delta t times the outflow of mass through the right and the top boundaries minus the inflow through the left and the bottom boundary, plus the viscosity times delta t times a discrete approximation of the second derivative. We note again that this is a temporary way to update the density, so that we can code up the fluid solver.



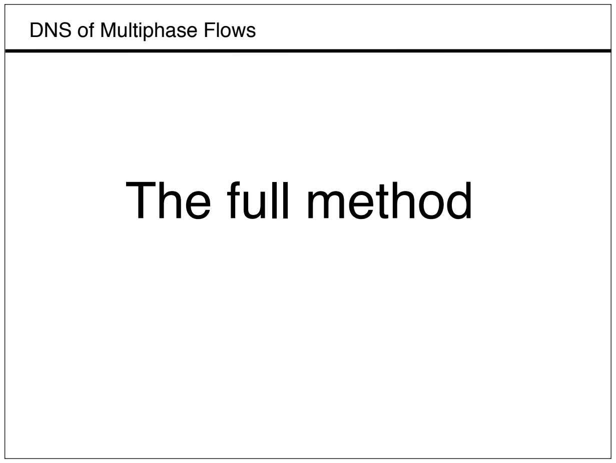
DNS of Multiphase Flows

Discrete version

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n} - \frac{\Delta t}{\Delta x}\Big(u_{i+1/2,j}\frac{\rho_{i+1,j}+\rho_{i,j}}{2} - u_{i-1/2,j}\frac{\rho_{i-1,j}+\rho_{i,j}}{2}\Big) - \frac{\Delta t}{\Delta y}\Big(v_{i,j+1/2}\frac{\rho_{i,j+1}+\rho_{i,j}}{2} - u_{i,j-1/2}\frac{\rho_{i,j-1}+\rho_{i,j}}{2}\Big) + \frac{\mu_0\Delta t}{\Delta x^2}(\rho_{i+1,j}-2\rho_{i,j}+\rho_{i-1,j}) + \frac{\mu_0\Delta t}{\Delta y^2}(\rho_{i,j+1}-2\rho_{i,j}+\rho_{i,j-1})$$

```
%=====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
```

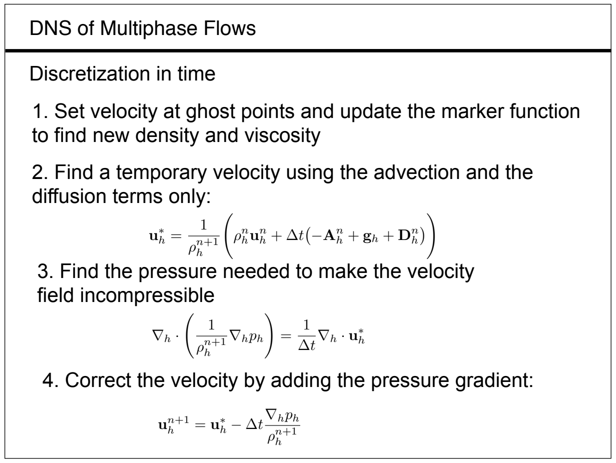
9. The code snippet for updating the density is a straightforward coding up of the discretization on the last slide. We loop over all the interior point, with i going from 2 to Nx+1 and j from 2 to Ny+1, computing the right hand side using the old values of the density.



DNS of Multiphase Flows

The full method

10. We are now ready to pull all the pieces together and write a complete flow solver capable of simulating flow driven by differences in density.



DNS of Multiphase Flows

Discretization in time

1. Set velocity at ghost points and update the marker function to find new density and viscosity

2. Find a temporary velocity using the advection and the diffusion terms only:

$$\mathbf{u}_h^* = \frac{1}{\rho_h^{n+1}}\left(\rho_h^n\mathbf{u}_h^n + \Delta t\big(-\mathbf{A}_h^n + \mathbf{g}_h + \mathbf{D}_h^n\big)\right)$$

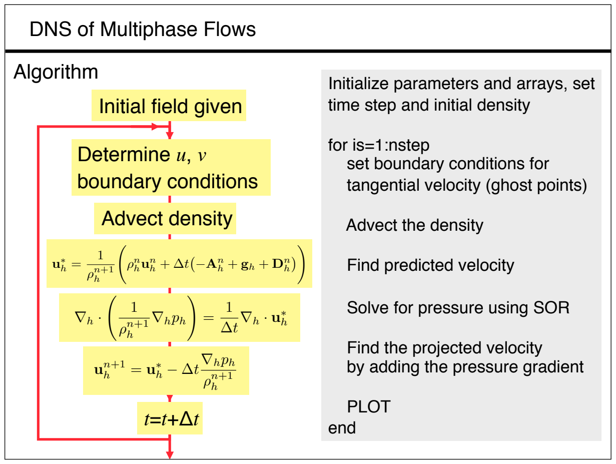
3. Find the pressure needed to make the velocity field incompressible

$$\nabla_h\cdot\left(\frac{1}{\rho_h^{n+1}}\nabla_h p_h\right) = \frac{1}{\Delta t}\nabla_h\cdot\mathbf{u}_h^*$$

4. Correct the velocity by adding the pressure gradient:

$$\mathbf{u}_h^{n+1} = \mathbf{u}_h^* - \Delta t\frac{\nabla_h p_h}{\rho_h^{n+1}}$$

11. First we review the steps, or the solution strategy. We start setting the velocity at the ghost points and finding the density at the new step, using the simple and temporary advection-diffusion equation approach just described. Then we find the predicted velocity. We then solve the pressure equation, and once we have the pressure, we can correct the new velocity field. Once we have the new velocity field we are ready to take the next time step.



DNS of Multiphase Flows

Algorithm

Initial field given

Determine u, v boundary conditions

Advect density

$$\mathbf{u}_h^* = \frac{1}{\rho_h^{n+1}}\left(\rho_h^n \mathbf{u}_h^n + \Delta t(-\mathbf{A}_h^n + \mathbf{g}_h + \mathbf{D}_h^n)\right)$$

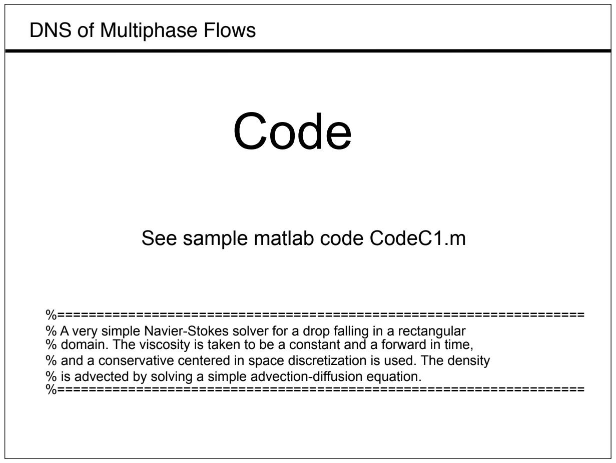
$$\nabla_h \cdot \left(\frac{1}{\rho_h^{n+1}}\nabla_h p_h\right) = \frac{1}{\Delta t}\nabla_h \cdot \mathbf{u}_h^*$$

$$\mathbf{u}_h^{n+1} = \mathbf{u}_h^* - \Delta t \frac{\nabla_h p_h}{\rho_h^{n+1}}$$

t=t+Δt

```
Initialize parameters and arrays, set
time step and initial density

for is=1:nstep
    set boundary conditions for
    tangential velocity (ghost points)

    Advect the density

    Find predicted velocity

    Solve for pressure using SOR

    Find the projected velocity
    by adding the pressure gradient

    PLOT
end
```

12. It is sometimes helpful to represents the steps in slightly different ways and on this slide we show a simple flowchart on the left and a pseudo code on the right. In both cases the steps are the same as on the previous slide: We set the ghost velocities, advect the density, find the predicted velocity, solve for the pressure and correct the velocity.



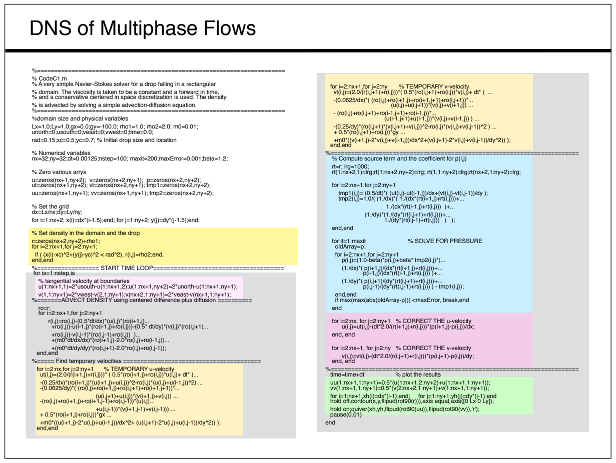
DNS of Multiphase Flows

Code

See sample matlab code CodeC1.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,
% and a conservative centered in space discretization is used. The density
% is advected by solving a simple advection-diffusion equation.
%=====================================================================
```

13. The full code is available as a download, but since it is only about hundred lines, we will go through the listing on the next slide. Remember though, that although this is a complete and working Navier-Stokes solver, it is a very elementary one since it is only first order in time, we advect the density in a very rudimentary way, and we take the viscosity to be the same everywhere.



DNS of Multiphase Flows

```
%=====================================================================
% CodeC1.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,
% and a conservative centered in space discretization is used. The density
% is advected by solving a simple advection-diffusion equation.
%=====================================================================
%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;
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
%================= 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);
%=======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
%===== Find temporary velocities ========================================
  for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity
    ut(i,j)=(2.0/(r(i+1,j)+r(i,j)))* ( 0.5*(ro(i+1,j)+ro(i,j))*u(i,j)+ dt* (...
    -(0.25/dx)*(ro(i+1,j)*(u(i+1,j)+u(i,j))^2-ro(i,j)*(u(i,j)+u(i-1,j))^2) ...
    -(0.0625/dy)*( (ro(i,j)+ro(i+1,j)+ro(i,j+1)+ro(i+1,j+1))*...
                          (u(i,j+1)+u(i,j))*(v(i+1,j)+v(i,j)) ...
    -(ro(i,j)+ro(i+1,j)+ro(i+1,j-1)+ro(i,j-1))*(u(i,j)...
                          +u(i,j-1))*(v(i+1,j-1)+v(i,j-1))) ...
    + 0.5*(ro(i+1,j)+ro(i,j))*gx ...
    +m0*((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)) );
  end,end

  for i=2:nx+1,for j=2:ny       % TEMPORARY v-velocity
    vt(i,j)=(2.0/(r(i,j+1)+r(i,j)))*( 0.5*(ro(i,j+1)+ro(i,j))*v(i,j)+ dt* (  ...
    -(0.0625/dx)*( (ro(i,j)+ro(i+1,j)+ro(i+1,j+1)+ro(i,j+1))*...
                          (u(i,j)+u(i,j+1))*(v(i,j)+v(i+1,j)) ...
    - (ro(i,j)+ro(i,j+1)+ro(i-1,j+1)+ro(i-1,j))*...
                        (u(i-1,j+1)+u(i-1,j))*(v(i,j)+v(i-1,j)) ) ...
    -(0.25/dy)*(ro(i,j+1)*(v(i,j+1)+v(i,j))^2-ro(i,j)*(v(i,j)+v(i,j-1))^2 ) ...
    + 0.5*(ro(i,j+1)+ro(i,j))*gy ...
    +m0*((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)) );
  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/(r(i+1,j)+r(i,j)))*(p(i+1,j)-p(i,j))/dx;
  end, end

  for i=2:nx+1, for j=2:ny   % CORRECT THE v-velocity
      v(i,j)=vt(i,j)-(dt*2.0/(r(i,j+1)+r(i,j)))*(p(i,j+1)-p(i,j))/dy;
  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
```

14-1. The full code is shown here, in sufficiently small fount so that it fits into two columns. After the header we set the various parameters that define the problem and the numerical solution. Then we define the various arrays and set them to zero. The initial density distribution, defining the drop, is set in the light yellow box. Most of the code consists of the time loop, which is shown with a gray background. Inside the time loop we first enforce the tangential velocities at the boundaries by setting the ghost points in the pinkish box. Then we advect the density. The predicted velocities are computed in the light brown box at the bottom of the first column and the top of the second column. In the light blue box the pressure equation is solved and the velocities are corrected in the dark pink or lightly red box. The final thing is the plotting, done in the green box. Notice that since we are using a staggered grid, we need to find the velocity at a common point before doing a quiver plot.

DNS of Multiphase Flows

```
%=====================================================================
% CodeC1.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,
% and a conservative centered in space discretization is used. The density
% is advected by solving a simple advection-diffusion equation.
%=====================================================================
%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;
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
%================== 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);
%=======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
%===== Find temporary velocities ========================================
  for i=2:nx,for j=2:ny+1      % TEMPORARY u-velocity
    ut(i,j)=(2.0/(r(i+1,j)+r(i,j)))* ( 0.5*(ro(i+1,j)+ro(i,j))*u(i,j)+ dt* (...
    -(0.25/dx)*(ro(i+1,j)*(u(i+1,j)+u(i,j))^2-ro(i,j)*(u(i,j)+u(i-1,j))^2) ...
    -(0.0625/dy)*( (ro(i,j)+ro(i+1,j)+ro(i,j+1)+ro(i+1,j+1))*...
                                (u(i,j+1)+u(i,j))*(v(i+1,j)+v(i,j)) ...
    -(ro(i,j)+ro(i+1,j)+ro(i+1,j-1)+ro(i,j-1))*(u(i,j)...
                                +u(i,j-1))*(v(i+1,j-1)+v(i,j-1))) ...
    + 0.5*(ro(i+1,j)+ro(i,j))*gx ...
    +m0*((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)) );
  end,end

  for i=2:nx+1,for j=2:ny      % TEMPORARY v-velocity
    vt(i,j)=(2.0/(r(i,j+1)+r(i,j)))*( 0.5*(ro(i,j+1)+ro(i,j))*v(i,j)+ dt* (  ...
    -(0.0625/dx)*( (ro(i,j)+ro(i+1,j)+ro(i+1,j+1)+ro(i,j+1))*...
                                  (u(i,j)+u(i,j+1))*(v(i,j)+v(i+1,j)) ...
    - (ro(i,j)+ro(i,j+1)+ro(i-1,j+1)+ro(i-1,j))*...
                                (u(i-1,j+1)+u(i-1,j))*(v(i,j)+v(i-1,j)) ) ...
    -(0.25/dy)*(ro(i,j+1)*(v(i,j+1)+v(i,j))^2-ro(i,j)*(v(i,j)+v(i,j-1))^2 ) ...
    + 0.5*(ro(i,j+1)+ro(i,j))*gy ...
    +m0*((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)) );
  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/(r(i+1,j)+r(i,j)))*(p(i+1,j)-p(i,j))/dx;
   end, end

   for i=2:nx+1, for j=2:ny   % CORRECT THE v-velocity
       v(i,j)=vt(i,j)-(dt*2.0/(r(i,j+1)+r(i,j)))*(p(i,j+1)-p(i,j))/dy;
   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
```

14-2. To get velocities at points that start on the boundary, we average u(i,j) and u(i,j+1) and v(i,j) and v(i+1,j) to get velocities at the corner of the pressure control volumes. The last pause statement ensures that the solution is plotted in every step. We note that we plot the solution at the end of the time loop so the first frame is after one time step. We could, obviously also plot at the beginning of the loop but then we would take one extra step at the end.

DNS of Multiphase Flows

The time step needs to be small enough so that the code is stable. For the explicit method used here we need to use:

Limitations on the time step

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

where

$$h = \max(\Delta x, \Delta y)$$

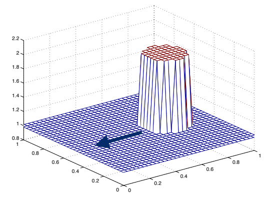
15. Since our method is explicit, the size of the time step must be small enough so that the method remains stable. We will not discuss the stability here, but simply quote the results. Those limits are derived for first order in time and second order centered in space discretization of the linear advection-diffusion equations but have been found to apply to the same approximations of the Navier-Stokes equations. As implemented the method is subject to two stability constrains. The first comes from the diffusion part and says that the viscosity times the size of the time step, divided by the density and the grid size squared must be less then one fourth. The second limitation comes from the fact that the current scheme is unstable for the inviscid case and the viscosity must be sufficiently large to stabilize the scheme. Thus, the maximum velocity squared times density and delta t, divided by the viscosity, must be less than 2.

DNS of Multiphase Flows

Example:

An initially stationary droplet in a square box that falls down due to gravity.

Here we ignore surface tension, take the viscosity to be the same as the ambient fluid and advect density by solving the advection-diffusion equation



Problem specification:
Lx=1.0; Ly=1.0;
gx=0.0; gy=-100.0;
rho1=1.0; rho2=2.0;
m0=0.01;

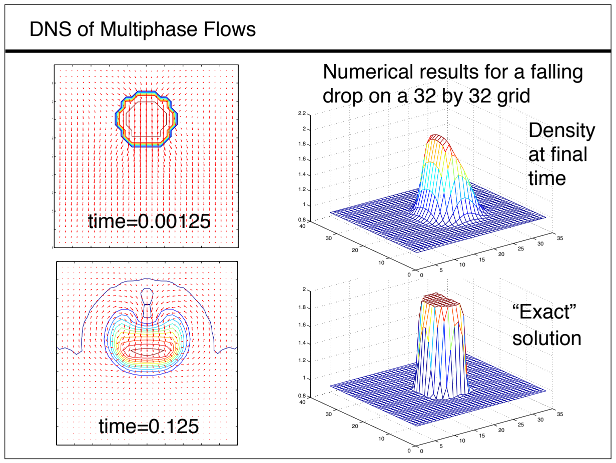
Tangential velocities:
unorth=0;usouth=0;
veast=0;vwest=0;

Initial drop size and location
rad=0.15; xc=0.5; yc=0.7;

Numerical variables
nx=32; ny=32;
dt=0.00125; nstep=100;

Pressure solver
maxit=200; beta=1.2;
maxError=0.001;

16. To test our code we use it to simulate the fall of a circular drop in a square domain. It is important to start with a simple problem that is easily solved, so we take the density of the drop to be only twice that of the ambient fluid. The tangent velocities at all walls are zero, explicitly set by specifying u north and so on and the normal velocities are also zero since the velocity field is initially set to zero and the boundary points are not updated. The drop has a radius of 0.15 and is initially located at x equal to 0.5 and y equal to 0.7. The grid is 32 times 32 pressure cells and we follow the evolution for 100 time steps, taking dt equal to 0.00125, determined by trial and error.



DNS of Multiphase Flows

Numerical results for a falling drop on a 32 by 32 grid

17. The code is easily run interactively so that the evolution of the solution can be seen. Here, however, we just show a contour plot of the density and the velocity field after one time step and after 100 steps, at time 0.125 on the left. Because of the coarse grid the initial shape of the drop is somewhat jagged. As the drop falls, it deforms, becoming flatter with the back “caving in,” and it also quickly becomes a smooth blob, instead of a drop with a sharp boundary. The density at the final time is shown in the top frame on the right hand side and in the bottom frame we show what it should look like. It is clear that even though it only fell a short distance, the present code does a terribly bad job at advecting the density and we need a better approach. It is true that we did add diffusion, but as discussed in the next lecture, direct advection of the density requires either a low order method with a similar level of diffusion or higher order methods that produce oscillatory or “wiggly” solutions. A different strategy for updating the density is therefore needed.