A guide to writing your first CFD solver

Mark Owkes
mark.owkes@montana.edu

April 11, 2024

Abstract

CFD is an exciting field today! Computers are getting larger and faster and are able to bigger problems and problems at a finer level. This document provides a guide for the beginners in the field of CFD. It describes the steps necessary to write a two-dimensional flow solver which can be used to solve the Navier-Stokes equations. The document begins by reviewing the governing equations and then discusses the various components needed to form a simple CFD solver.

Contents

1 Governing equations

The Navier-Stokes equations describe almost all the flows around us and are the starting point for a CFD code. Additionally since the majority of flows can be approximated as incompressible, we will solve the incompressible form of the equations. The incompressible Navier-Stokes equations can be written as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\boldsymbol{u} \tag{1}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{2}$$

where $\boldsymbol{u} = [u, v]$ is the velocity vector, t is time, ρ is the density, p is pressure, and ν is the kinematic viscosity. The first equation is the momentum equation and the second equation is the continuity equation

which ensures incompressibility. These equations can not be solved analytically for most flows and must be solved using numerical methods.

2 Computational mesh

The governing equations are solved on a computational mesh. The mesh used in this document is uniform with mesh cells of width Δx and height, Δy. The grid divides the domain in to $n_x \times n_y$ cells where n_x and n_y are the number of cells in the x and y directions, respectively.

The grid cells are referred to using their index. The i index referes to the cells x direction and the j index referes to cells in the y direction.

A staggered grid is used to store the variables where the pressure is stored at the cell center and the velocities are stored at the cell faces. This, possibly odd, choice is made since it allows for the solution to have a tight coupling between pressure and the velocity and has been found to be the preferred methodology.

Arrays are created to refer to the locations important for each cell. $x(i)$ stores the location of the ith cells left face. $y(j)$ stores the location of the jth cells bottom face. The location of the middle of the cell is stored in the $x_m(i)$ and the $y_m(j)$ arrays.

MATLAB code to create this mesh is

```
% Index extents
imin=2; imax=imin+nx-1;
jmin=2; jmax=jmin+ny-1;

% Create mesh
x(imin:imax+1)=linspace(0,Lx,nx+1);
y(jmin:jmax+1)=linspace(0,Ly,ny+1);
xm(imin:imax)=0.5*(x(imin:imax)+x(imin+1:imax+1));
ym(jmin:jmax)=0.5*(y(jmin:jmax)+y(jmin+1:jmax+1));

% Create mesh sizes
dx=x(imin+1)-x(imin);
dy=y(jmin+1)-y(jmin);
dxi=1/dx;
dyi=1/dy;
```

A few notes on this code:

- nx=n_x and ny=n_y
- Lx and Ly are the lengths of the domain in the x and y directions, respectively.
- The index extents, imin, imax, jmin, and jmax, provide a quick way to access the first and last computational cells. The index extents do not start at 1 because we need to add cells outside the domain to enforce boundary conditions (more on this later).
- The mesh sizes are precomputed to save computational cost. Additionally $dxi = 1/dx$ and $dyi = 1/dy$ are also precomputed since divisions are significantly more computationally expensive than multiplications.

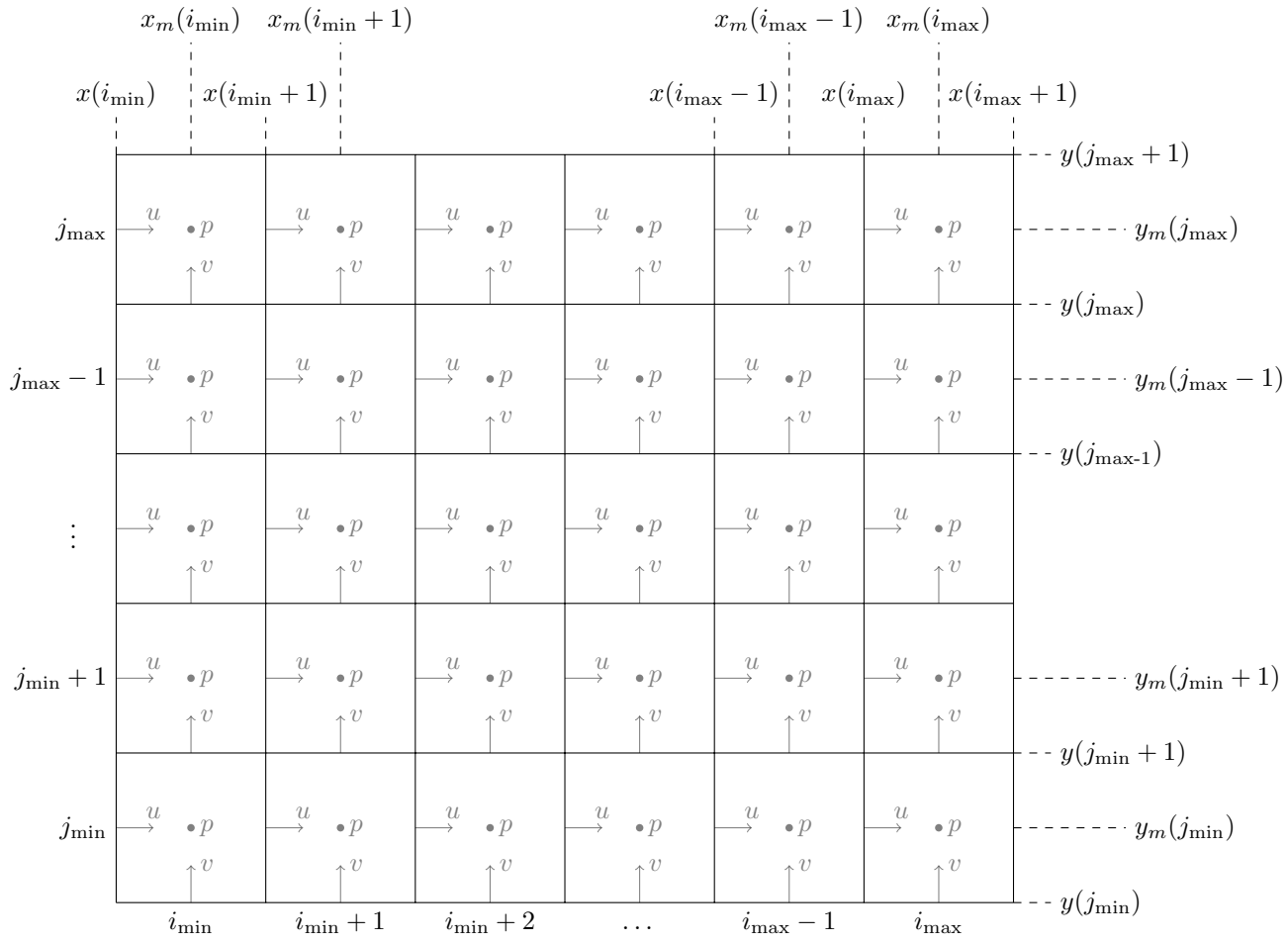

Figure 1: Computational mesh with location of the velocities and pressures. The x, x_m, y, and y_m array locations are also shown.

3 Temporal discretization

Temporal discretization is done using an explicit Euler scheme which can be written as,

$$\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^{n}}{\Delta t}=-\frac{1}{\rho}\nabla p^{n+1}-\boldsymbol{u}^{n}\cdot\nabla\boldsymbol{u}^{n}+\nu\nabla^{2}\boldsymbol{u}^{n}. \tag{3}$$

In the previous equation the superscript refers to the temporal iteration. Typically the simulation is started with $n=0$ and the initial condition is used to populate the initial velocity field $\boldsymbol{u}^{n=0}$. The equation is used to find subsequent solutions. The time-step Δt should be chosen so that $u\Delta t/\Delta x<1$. This condition is known as the Courant-Friedrichs-Lewy (CFL) condition.

The previous equation does not include the role of continuity and u^{n+1} is not guaranteed to be divergence-free. To introduce continuity we solve this equation using the predictor-corrector or fractional step methodology. In this framework the Navier-Stokes equations are solved in two steps.

The first step, known as the *predictor step*, is to compute an intermediate velocity $\boldsymbol{u}^{*}$ by solving the momentum equation but omitting the effect of pressure, i.e.,

$$\frac{\boldsymbol{u}^{*}-\boldsymbol{u}^{n}}{\Delta t}=-\boldsymbol{u}^{n}\cdot\nabla\boldsymbol{u}^{n}+\nu\nabla^{2}\boldsymbol{u}^{n}. \tag{4}$$

The second step, known as the *corrector step*, is to solve for the new velocity u^{n+1} and include the influence of the pressure leading to

$$\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^{*}}{\Delta t}=-\frac{1}{\rho}\nabla p^{n+1}. \tag{5}$$

It is easy to show that Eq. 3 = Eq. 4 + Eq. 5.

The pressure is found such that u^{n+1} satisfies the continuity equation by solving

$$\nabla^{2}p^{n+1}=\frac{\rho}{\Delta t}\nabla\cdot\boldsymbol{u}^{*}, \tag{6}$$

which can be derived by taking the divergence of Eq. 5 and enforcing $\nabla\cdot\boldsymbol{u}^{n+1}=0$. This equation is referred to as the *pressure Poisson equation*.

4 u momentum discretization

The convective and viscous terms in Eq. 4 are discretized using finite differences which approximate the derivatives using neighboring values.

The predictor step for u velocity can be written as

$$u^{*}=u^{n}+\Delta t\left(\nu\left(\frac{\partial^{2}u^{n}}{\partial x^{2}}+\frac{\partial^{2}u^{n}}{\partial y^{2}}\right)-\left(u^{n}\frac{\partial u^{n}}{\partial x}+v^{n}\frac{\partial u^{n}}{\partial y}\right)\right) \tag{7}$$

The viscous and convective terms are discretized for the i,j cell using

$$\frac{\partial^{2}u}{\partial x^{2}}=\frac{u(i-1,j)-2u(i,j)+u(i+1,j)}{\Delta x^{2}} \tag{8}$$

$$\frac{\partial^{2}u}{\partial y^{2}}=\frac{u(i,j-1)-2u(i,j)+u(i,j+1)}{\Delta y^{2}} \tag{9}$$

$$u\frac{\partial u}{\partial x}=u(i,j)\frac{u(i+1,j)-u(i-1,j)}{2\Delta x} \tag{10}$$

$$v\frac{\partial u}{\partial y}=\frac{1}{4}(v(i-1,j)+v(i,j)+v(i-1,j+1)+v(i,j+1))\frac{u(i,j+1)-u(i,j-1)}{2\Delta y} \tag{11}$$

Figure 2 shows the velocity values used in the discretization.

MATLAB code that computes u^{*} is

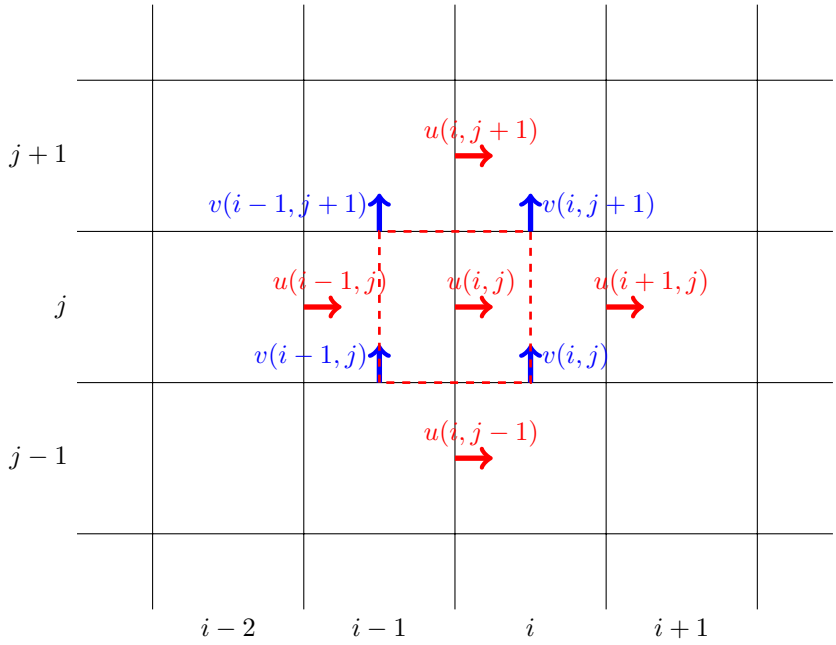

Figure 2: Velocities and their locations used to discretize the $u(i,j)$ cell in the u-momentum equation..

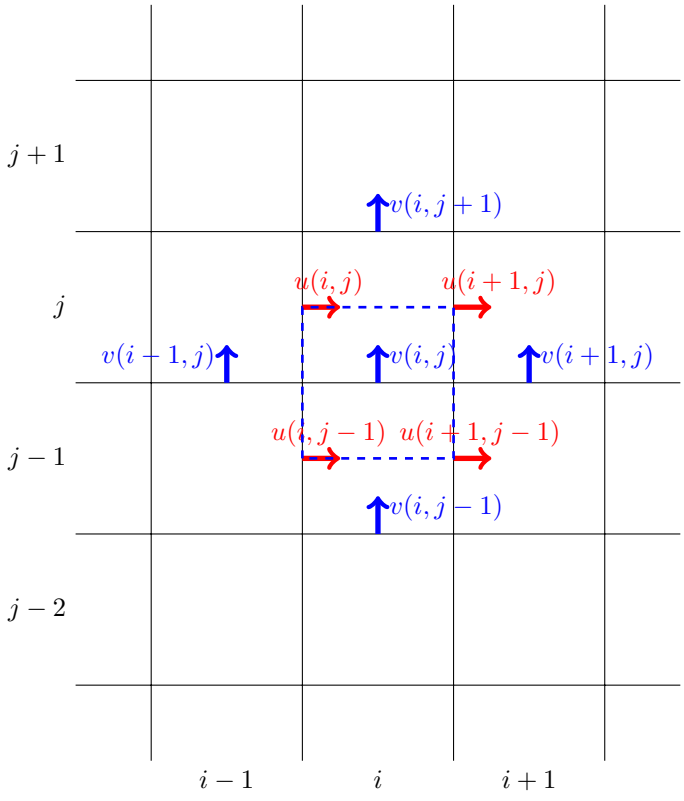

Figure 3: Velocities and their locations used to discretize the $v(i,j)$ cell in the v-momentum equation..

```
for j=jmin:jmax
    for i=imin+1:imax
        v_here=0.25*(v(i-1,j)+v(i-1,j+1)+v(i,j)+v(i,j+1));
        us(i,j)=u(i,j)+dt* ...
            (nu*(u(i-1,j)-2*u(i,j)+u(i+1,j))*dxi^2 ...
            +nu*(u(i,j-1)-2*u(i,j)+u(i,j+1))*dyi^2 ...
            -u(i,j)*(u(i+1,j)-u(i-1,j))*0.5*dxi ...
            -v_here*(u(i,j+1)-u(i,j-1))*0.5*dyi);
    end
end
```

5 v momentum discretization

Following the same approach as we did for u we can write the predictor step for the v velocity as

$$v^* = v^n + \Delta t\left(\nu\left(\frac{\partial^2 v^n}{\partial x^2} + \frac{\partial^2 v^n}{\partial y^2}\right) - \left(u\frac{\partial v^n}{\partial x} + v^n\frac{\partial v^n}{\partial y}\right)\right). \tag{12}$$

The viscous and convective terms are discretized for the i, j cell using

$$\frac{\partial^2 v}{\partial x^2} = \frac{v(i-1,j) - 2v(i,j) + v(i+1,j)}{\Delta x^2} \tag{13}$$

$$\frac{\partial^2 v}{\partial y^2} = \frac{v(i,j-1) - 2v(i,j) + v(i,j+1)}{\Delta y^2} \tag{14}$$

$$u\frac{\partial v}{\partial x} = \frac{1}{4}(u(i,j-1) + u(i,j) + u(i+1,j-1) + u(i+1,j))\frac{v(i+1,j) - v(i-1,j)}{2\Delta x} \tag{15}$$

$$v\frac{\partial v}{\partial y} = v(i,j)\frac{v(i,j+1) - v(i,j-1)}{2\Delta y} \tag{16}$$

Figure 3 shows the velocity values used in the discretization.
MATLAB code that computes v^* is

```
for j=jmin+1:jmax
    for i=imin:imax
        u_here=0.25*(u(i,j-1)+u(i,j)+u(i+1,j-1)+u(i+1,j));
        vs(i,j)=v(i,j)+dt* ...
            (nu*(v(i-1,j)-2*v(i,j)+v(i+1,j))*dxi^2 ...
            +nu*(v(i,j-1)-2*v(i,j)+v(i,j+1))*dyi^2 ...
            -u_here*(v(i+1,j)-v(i-1,j))*0.5*dxi ...
            -v(i,j)*(v(i,j+1)-v(i,j-1))*0.5*dyi);
    end
end
```

6 Poisson equation

The pressure Poisson equation, Eq. 6 is used to create a velocity field that satisfies the continuity equation and is incompressible. Solving the Poisson equation almost always uses the majority of the computational cost in the solution calculation. Many ways can be used to solve the Poisson equation and some are faster than others. The simplest way to solve the Poisson equation is to write it as

$$\boldsymbol{L}\boldsymbol{p}^{n+1} = \boldsymbol{R} \tag{17}$$

where $\boldsymbol{L} = \nabla^2$ is the Laplacian operator (a large $n_x \cdot n_y \times n_x \cdot n_y$ matrix), $\boldsymbol{p}^{n+1}$ is the pressure in each computational cell organized into one large vector, and $\boldsymbol{R} = -\frac{\rho}{\Delta t}\nabla \cdot \boldsymbol{u}^*$ is the right-hand-side of the pressure Poisson equation in each computational cell organized into one large vector. This equation can be solved for $\boldsymbol{p}^{n+1}$ using MATLAB's built in solver, i.e., $\boldsymbol{p}^{n+1} = \boldsymbol{L}\backslash\boldsymbol{R}$. It is advisable to use this method in your first CFD code. Note that there are other methods that do not require forming $\boldsymbol{L}$ which requires too much memory to store for large problems.

Boundary conditions for the Pressure poisson equation are Neuman or zero derivative. Additionally, the pressure poisson equation is only defined up to a constant and the pressure in one computational cell needs to be set. All other pressures in the domain are computed with respect to this pressure.

To discretie the pressure equation in the i, j cell we use (see Fig. 4 for details on the location of pressures and velocities),

$$\begin{aligned}
\nabla^2 p^{n+1} &= \frac{\partial^2 p^{n+1}}{\partial x^2} + \frac{\partial^2 p^{n+1}}{\partial y^2} \\
&\approx \frac{p^{n+1}(i-1,j) - 2p^{n+1}(i,j) + p^{n+1}(i+1,j)}{\Delta x^2} + \frac{p^{n+1}(i,j-1) - 2p^{n+1}(i,j) + p^{n+1}(i,j+1)}{\Delta y^2}
\end{aligned} \tag{18}$$

$$\begin{aligned}
\nabla \cdot \boldsymbol{u}^* &= \frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} \\
&\approx \frac{u^*(i+1,j) - u^*(i,j)}{\Delta x} + \frac{v^*(i,j+1) - v^*(i,j)}{\Delta y}
\end{aligned} \tag{19}$$

Using this discretization we can write Eq. 17 as

$$\underset{\boldsymbol{L}}{\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{-1}{\Delta x^2} & D_y & \frac{-1}{\Delta x^2} & 0 & \frac{-1}{\Delta y^2} & 0 & 0 & 0 & 0 \\
0 & \frac{-1}{\Delta x^2} & D_{xy} & 0 & 0 & \frac{-1}{\Delta y^2} & 0 & 0 & 0 \\
\frac{-1}{\Delta y^2} & 0 & 0 & D_x & \frac{-1}{\Delta x^2} & 0 & \frac{-1}{\Delta y^2} & 0 & 0 \\
0 & \frac{-1}{\Delta y^2} & 0 & \frac{-1}{\Delta x^2} & D & \frac{-1}{\Delta x^2} & 0 & \frac{-1}{\Delta y^2} & 0 \\
0 & 0 & \frac{-1}{\Delta y^2} & 0 & \frac{-1}{\Delta x^2} & D_x & 0 & 0 & \frac{-1}{\Delta y^2} \\
0 & 0 & 0 & \frac{-1}{\Delta y^2} & 0 & 0 & D_{xy} & \frac{-1}{\Delta x^2} & 0 \\
0 & 0 & 0 & 0 & \frac{-1}{\Delta y^2} & 0 & \frac{-1}{\Delta x^2} & D_y & \frac{-1}{\Delta x^2} \\
0 & 0 & 0 & 0 & 0 & \frac{-1}{\Delta y^2} & 0 & \frac{-1}{\Delta x^2} & D_{xy}
\end{bmatrix}}
\underset{\boldsymbol{p}^{n+1}}{\begin{bmatrix} p(1,1) \\ p(2,1) \\ p(3,1) \\ p(1,2) \\ p(2,2) \\ p(3,2) \\ p(1,3) \\ p(2,3) \\ p(3,3) \end{bmatrix}}
= \underset{\boldsymbol{R}}{\begin{bmatrix} R(1,1) \\ R(2,1) \\ R(3,1) \\ R(1,2) \\ R(2,2) \\ R(3,2) \\ R(1,3) \\ R(2,3) \\ R(3,3) \end{bmatrix}} \tag{20}$$

where $D = \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}$, $D_x = D - \frac{1}{\Delta x^2}$, $D_y = D - \frac{1}{\Delta y^2}$, and $D_{xy} = D - \frac{1}{\Delta x^2} - \frac{1}{\Delta y^2}$. The modify diagonal entries are due to the Neuman boundary conditions. For this $n_x \times n_y = 3 \times 3$ problem there is only 1 interior point and the unmodified D only shows up once.

Note that the Laplacian only depends on the computational mesh and can be computed once at the beginning of the simulation and stored. Additionally you can greatly speed-up your code by performing an LU decomposition of L which can easily be done in MATLAB using L = decomposition(L);. Here is MATLAB code that creates the Laplacian operator $\boldsymbol{L}$:

```
% Create Laplacian operator for solving pressure Poisson equation
L=zeros(nx*ny,nx*ny);
for j=1:ny
    for i=1:nx
        L(i+(j-1)*nx,i+(j-1)*nx)=2*dxi^2+2*dyi^2;
        for ii=i-1:2:i+1
            if (ii>0 && ii<=nx) % Interior point
                L(i+(j-1)*nx,ii+(j-1)*nx)=-dxi^2;
            else                % Neuman conditions on boundary
                L(i+(j-1)*nx,i  +(j-1)*nx)= ...
                    L(i+(j-1)*nx,i  +(j-1)*nx)-dxi^2;
            end
        end
        for jj=j-1:2:j+1
            if (jj>0 && jj<=ny) % Interior point
                L(i+(j-1)*nx,i+(jj-1)*nx)=-dyi^2;
            else                % Neuman conditions on boundary
                L(i+(j-1)*nx,i+(j  -1)*nx)= ...
                    L(i+(j-1)*nx,i+(j  -1)*nx)-dyi^2;
            end
        end
    end
end
% Set pressure in first cell (all other pressures w.r.t to this one)
L(1,:)=0; L(1,1)=1;

% Perform LU decomposition to speed-up code (uncomment once code is working)
% L = decomposition(L);
```

MATLAB code to compute the right-hand-side $\boldsymbol{R}$ is:

```
n=0;
for j=jmin:jmax
    for i=imin:imax
        n=n+1;
        R(n)=-rho/dt* ...
            ((us(i+1,j)-us(i,j))*dxi ...
            +(vs(i,j+1)-vs(i,j))*dyi);
    end
end
```

The pressure is found by solving $\boldsymbol{L}\boldsymbol{p}^{n+1} = \boldsymbol{R}$ which in MATLAB can be done using

```
pv=L\R;
```

where pv is the vector representation of the pressure. Finally, pv is converted to the mesh representation p(i,j) which can be done using

```
n=0;
p=zeros(imax,jmax);
for j=jmin:jmax
    for i=imin:imax
        n=n+1;
        p(i,j)=pv(n);
    end
end
```

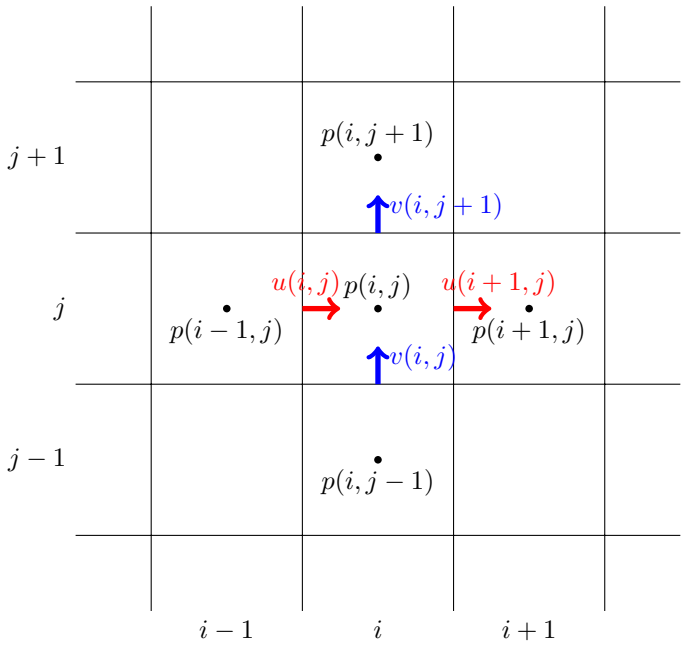

Figure 4: Velocities and their locations used to discretize the $p(i,j)$ cell in the pressure Poisson equation.

7 Corrector step

Once the pressure is computed using the Poisson equation it is used to update the velocity from $\boldsymbol{u}^*$ to $\boldsymbol{u}^{n+1}$ using Eq. 5. The pressure gradient can be computed using finite differences. To update u and v we use

$$\frac{\partial p}{\partial x} = \frac{p(i,j) - p(i-1,j)}{\Delta x}, \text{ and} \tag{21}$$

$$\frac{\partial p}{\partial y} = \frac{p(i,j) - p(i,j-1)}{\Delta y}, \tag{22}$$

respectively. MATLAB code to perform the corrector step is:

```
for j=jmin:jmax
    for i=imin+1:imax
        u(i,j)=us(i,j)-dt/rho*(p(i,j)-p(i-1,j))*dxi;
    end
```

```
    end
    for  j=jmin+1:jmax
        for  i=imin:imax
            v(i,j)=vs(i,j)-dt/rho*(p(i,j)-p(i,j-1))*dyi;
        end
    end
```

8 Boundary conditions

Boundary conditions are needed for the velocity field and the pressure Poisson equation. The velocity boundary conditions are a bit tricky since some of the velocity components are not defined on the boundary. For example, to specify $u = u_{\text{top}}$ at the top of the domain is not straightforward because u is defined $\Delta x/2$ away from the top boundary. One solution to this problem is to create a fictitious velocity outside the domain such that the velocity on the domain boundary satisfies the boundary condition. In Fig. 5 shows how using a fictitious velocity at $u(i, j_{\max}+1) = -u(i, j_{\max})$ provides a $u_{\text{top}} = 0$ boundary condition. For a general boundary condition we can write

$$u_{\text{top}} = \frac{1}{2}(u(i, j_{\max}) + u(i, j_{\max}+1)), \tag{23}$$

which says the velocity at the top of the domain should be the average of the two neighboring velocities (linear interpolation). Rearranging this equations provides an expression to set the fictitious velocity.

$$u(i, j_{\max}+1) = 2u_{\text{top}} - u(i, j_{\max}) \tag{24}$$

Similar boundary conditions can be written for the other sides of the domain when the velocity is not coincident with the domain boundary. Here is MATLAB code to enforce the boundary conditions on the four sides of the domain:

```
u(:,jmin-1)=2*u_bot - u(:,jmin);
u(:,jmax+1)=2*u_top - u(:,jmax);
v(imin-1,:)=2*v_lef - v(imin,:);
v(imax+1,:)=2*v_rig - v(imax,:);
```

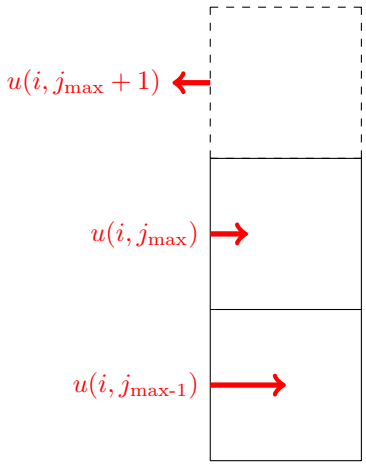

Figure 5: Example of how to apply a $u_{\text{top}} = 0$ boundary condition using the fictitious velocity $u(i, j_{\max}+1)$.

9 General overview of the code

You've almost made it to the end and are probably overwhelmed by all of the pieces. In this section, I will try to organize the pieces into a coherent structure that you can use as an outline when you write your code.

- Set input parameters: viscosity, density, number of grid points, time information, and boundary conditions
- Create the index extents and the computational grid (see Section 2)
- Initialize any arrays you use to allocate the memory
- Create the Laplacian operator (see Section 6)
- Apply boundary conditions to the initial velocity field (see Section 8)
- Loop over time (use a for or while loop)
  - Update time $t = t + \Delta t$
  - Perform the predictor step to find u^* and v^* (see Sections 4 and 5)
  - Apply boundary conditions to the predicted velocity field (see Section 8)
  - Form the right-hand-side of the Poisson equation (see Section 6)
  - Solve for the pressure using $\boldsymbol{pv} = \boldsymbol{L} \backslash \boldsymbol{R}$ and convert the pressure vector pv into a matrix $p(i, j)$ (see Section 6)
  - Perform the corrector step to find u^{n+1} and v^{n+1} (see Section 7) (copy the boundary conditions from the predicted velocity onto u^{n+1} and v^{n+1})
  - Plot the velocity field and the pressure field
- End Simulation

10 Test Case

Once you're code is written, it is useful to test it. A simple test case to try is the lid-driven cavity problem which is described here: `https://www.cfd-online.com/Wiki/Lid-driven_cavity_problem`.