Math 511 Project 2:
ADI for Pressure-driven Laminar Flow

Shane McQuarrie

Introduction

A fluid that flows in parallel layers with no disruption between the layers is said to have a *laminar flow*. It is characterized by a smooth flow with high momentum diffusion and low momentum convection. We consider an initial boundary value problem for the start-up of pressure-driven laminar flow in an infinite channel of square cross section. That is, a fluid is flowing through a long pipe with a square cross section, which we will choose to be the unit square, and the function $u(x, y, t)$ describes the velocity of the fluid at the given space and time. The fluid adheres slightly to the pipe wall so that the velocity of the fluid at the boundary of the cross section is zero (this is called the *no-slip boundary condition*). In addition, the velocity is being internally driven by the pressure, resulting in a source term.

We model this situation with the following equations.

$$u_t = \Delta u + 1, \qquad t > 0,\ (x, y) \in [0, 1] \times [0, 1] \tag{1}$$

$$u(x, y, 0) = 0, \qquad (x, y) \in [0, 1] \times [0, 1] \tag{2}$$

$$u(0, y, t) = u(x, 0, t) = 0, \qquad t > 0 \tag{3}$$

$$u(1, y, t) = u(x, 1, t) = 0, \qquad t > 0 \tag{4}$$

Here $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, the 2-D Laplacian operator. Note that the governing equation (1) is just the 2-D heat equation with a source term.

Analytical Solution

The exact solution for this problem can be obtained by separation of variables and eigenfunction expansions. We do not provide the derivation, but the equation is given below.

$$u(x, y, t) = \frac{15}{\pi^2} \sum_{k,l=0}^{\infty} \frac{\sin(\pi(1 + 2k)x) \sin(\pi(1 + 2l)y)}{(1 + 2k)(1 + 2l)[(1 + 2k)^2 + (1 + 2l)^2]} \left(1 - e^{-\pi^2\left((1+2k)^2 + (1+2l)^2\right)}\right) \tag{5}$$

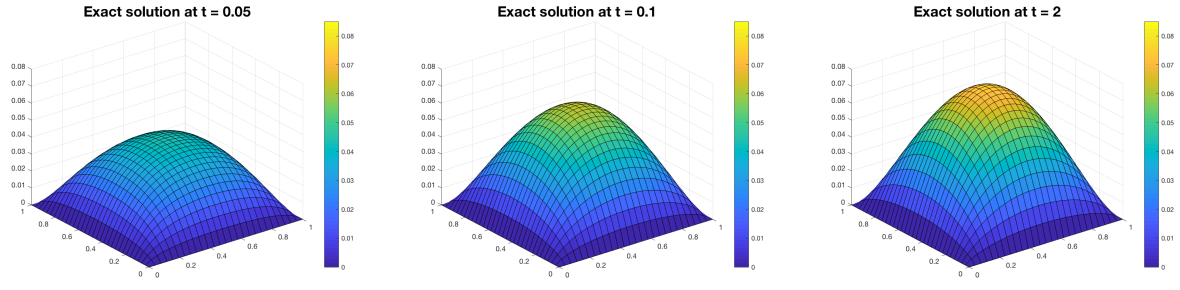
The following code computes the exact solution for a given (x, y, t), truncated to 30 terms for l and k.

```
function u_exact = ExactSoln(x, y, t, N)

total = 0;
for k=0:N
    kk = 1 + 2*k;                    % (1 + 2k)
    for l=0:N
        ll = 1 + 2*l;                % (1 + 2l)
        kl2 = kk^2 + ll^2;           % (1 + 2k)^2 + (1 + 2l)^2
        temp = sin(pi*kk*x) .* sin(pi*ll*y);
        total = total + (temp * (1 - exp(-pi^2 * kl2*t)) / (kk*ll*kl2));
    end
end

u_exact = 16 * total / pi^4;
end
```

Using this function, we compute exact solutions for $u(x, y, y)$ at $t = .05, .1, 2$.



So the solution grows with a symmetric bump in the middle until reaching a steady state (where the pressure and the diffusion effects cancel each other out).

ADI Equations

To approximate the solution to this problem numerically we will use an Alternating Direction Implicit (ADI) method. The idea is to use only the rows of the solution at time t_n step to compute the solution at time $t_{n+1/2}$, then use the columns of the solution at time $t_{n+1/2}$ to compute the solution at time t_{n+1}.

Discretization

First, we must discretize the system. We partition the x-axis of the domain into $J+2$ equally spaced points $0 = x_0 < x_1 < \cdots < x_J < x_{J+1} = 1$, and the y-axis of the domain into $K+2$ equally spaced points $0 = y_0 < y_1 < \cdots < y_K < y_{K+1} = 1$. Then there are J interior points in the x direction and K interior points in the y direction. For time, we also choose equally

spaced points $0 = t_0 < t_1 < \cdots < t_{N-1}$ so that there are N time steps. For the spacing, let

$$\Delta x = x_{i+1} - x_i = \frac{1}{J+1}, \qquad \Delta y = y_{i+1} - y_i = \frac{1}{K+1}, \qquad \Delta t = t_{i+1} - t_i = \frac{t_{N-1}}{N-1}$$

for each i.

Next, let $U^n_{j,k} = u(x_j, y_k, t_n)$ be the discretization of u over the 3-D grid. The initial condition (2) translates into

$$0 = u(x_j, y_k, 0) = U^0_{j,k} \quad \forall j, k. \tag{6}$$

The no-slip boundary conditions (3)–(4) become

$$U^n_{0,k} = u(x_0, y_k, t) = u(0, y_k, t) = 0 \qquad \forall k, n \tag{7}$$
$$U^n_{j,0} = u(x_j, y_0, t) = u(x_j, 0, t) = 0 \qquad \forall j, n \tag{8}$$
$$U^n_{J+1,k} = u(x_{J+1}, y_k, t) = u(1, y_k, t) = 0 \qquad \forall k, n \tag{9}$$
$$U^n_{j,K+1} = u(x_j, y_{K+1}, t) = u(x_j, 1, t) = 0 \qquad \forall j, n \tag{10}$$

Thus given $n > 0$, we must solve for only JK different values of $U^n_{k,j}$.

Since we will solve for the solution by rows and columns, we use MATLAB-like notation to denote the following vectors.

$$U^n_{j,:} = \begin{bmatrix} U^n_{j,1} \\ U^n_{j,2} \\ \vdots \\ U^n_{j,K-1} \\ U^n_{j,K} \end{bmatrix}, \qquad U^n_{:,k} = \begin{bmatrix} U^n_{1,k} \\ U^n_{2,k} \\ \vdots \\ U^n_{J-1,k} \\ U^n_{J,k} \end{bmatrix}$$

In other words, $U^n_{j,:}$ is the jth row of the solution at time t_n (x fixed) and $U{:}, k^n$ is the kth column of the solution at time t_n (y fixed).

Predictor Equations

The first step of ADI—solving for the $U^{n+1/2}_{j,k}$—is called the *predictor step*. Since these values are not at an actual time step, they will be stored in a temporary array and used to compute solution at the next time step in the *corrector step*.

We begin by splitting (1) into two parts.

$$\frac{1}{2}u_t = u_{xx} + \frac{1}{2} \qquad\qquad \frac{1}{2}u_t = u_{yy} + \frac{1}{2} \tag{11}$$

For u_{xx}, we use the shorthand notation

$$u_{xx}(x_j, y_k, t_n) = \frac{U^n_{j-1,k} - 2U^n_{j,k} + U^n_{j+1,k}}{(\Delta x)^2} = \frac{\delta^2_x U^n_{j,k}}{(\Delta x)^2},$$

so δ_x^2 represents the second-order centered finite difference approximation for u_{xx} without the denominator $(\Delta x)^2$. Similarly, δ_y^2 is the second-order centered finite difference approximation for u_{yy} without the denominator $(\Delta y)^2$.

For the left equation of (11), we use a backwards in time approximation for u_t (from $t_{n+1/2}$ back to t_n) and a centered approximation for u_{xx}.

$$\begin{aligned}
\frac{1}{2}u_t = u_{xx} + \frac{1}{2} \quad \Longrightarrow \quad \frac{1}{2}\left(\frac{U_{j,k}^{n+1/2} - U_{j,k}^{n}}{(\Delta t)/2}\right) &= \frac{\delta_x^2 U_{j,k}^{n+1/2}}{(\Delta x)^2} + \frac{1}{2} \\
\frac{1}{2}U_{j,k}^{n+1/2} - \frac{1}{2}U_{j,k}^{n} &= \frac{\Delta t}{2(\Delta x)^2}\delta_x^2 U_{j,k}^{n+1/2} + \frac{\Delta t}{4} \\
\left[\frac{1}{2} - \frac{\Delta t}{2(\Delta x)^2}\delta_x^2\right] U_{j,k}^{n+1/2} &= \frac{1}{2}U_{j,k}^{n} + \frac{\Delta t}{4} \qquad (12)
\end{aligned}$$

For the right equation of (11), we use a forwards in time approximation for u_t (from t_n to $t_{n+1/2}$) and a centered approximation for u_{yy}.

$$\begin{aligned}
\frac{1}{2}u_t = u_{yy} + \frac{1}{2} \quad \Longrightarrow \quad \frac{1}{2}\left(\frac{U_{j,k}^{n+1/2} - U_{j,k}^{n}}{(\Delta t)/2}\right) &= \frac{\delta_y^2 U_{j,k}^{n}}{(\Delta y)^2} + \frac{1}{2} \\
\frac{1}{2}U_{j,k}^{n+1/2} - \frac{1}{2}U_{j,k}^{n} &= \frac{\Delta t}{2(\Delta y)^2}\delta_y^2 U_{j,k}^{n} + \frac{\Delta t}{4} \\
\frac{1}{2}U_{j,k}^{n+1/2} &= \left[\frac{1}{2} + \frac{\Delta t}{2(\Delta y)^2}\delta_y^2\right] U_{j,k}^{n} + \frac{\Delta t}{4} \qquad (13)
\end{aligned}$$

Adding (12) and (13) gives our complete scalar predictor step equation,

$$\begin{aligned}
\left[1 - \frac{\Delta t}{2(\Delta x)^2}\delta_x^2\right] U_{j,k}^{n+1/2} &= \left[1 + \frac{\Delta t}{2(\Delta y)^2}\delta_y^2\right] U_{j,k}^{n} + \frac{\Delta t}{2} \\
U_{j,k}^{n+1/2} - \frac{r_x}{2}\delta_x^2 U_{j,k}^{n+1/2} &= U_{j,k}^{n} + \frac{r_y}{2}\delta_y^2 U_{j,k}^{n} + \frac{\Delta t}{2} \qquad (14)
\end{aligned}$$

$$-\frac{r_x}{2}U_{j-1,k}^{n+1/2} + (1+r_x)U_{j,k}^{n+1/2} - \frac{r_x}{2}U_{j+1,k}^{n+1/2} = \frac{r_y}{2}U_{j,k-1}^{n} + (1-r_y)U_{j,k}^{n} + \frac{r_y}{2}U_{j,k+1}^{n} + \frac{\Delta t}{2}, \quad (15)$$

where $r_x = \frac{\Delta t}{(\Delta x)^2}$ and $r_y = \frac{\Delta t}{(\Delta y)^2}$.

Recall that goal of this step is to solve for the columns $U_{:,k}^{n+1/2}$, for which we must set up linear systems. For interior terms ($j = 2, 3, \ldots, J-1$, $k = 2, 3, \ldots, K-1$), we can use (15). For $j = 1$, (7) and (15) combine for a slight simplification.

$$\begin{aligned}
-\frac{r_x}{2}U_{0,k}^{n+1/2} + (1+r_x)U_{1,k}^{n+1/2} - \frac{r_x}{2}U_{2,k}^{n+1/2} &= \frac{r_y}{2}U_{1,k-1}^{n} + (1-r_y)U_{1,k}^{n} + \frac{r_y}{2}U_{1,k+1}^{n} + \frac{\Delta t}{2} \\
(1+r_x)U_{1,k}^{n+1/2} - \frac{r_x}{2}U_{2,k}^{n+1/2} &= \frac{r_y}{2}U_{1,k-1}^{n} + (1-r_y)U_{1,k}^{n} + \frac{r_y}{2}U_{1,k+1}^{n} + \frac{\Delta t}{2}.
\end{aligned}$$

Similarly, for $j = J$, (9) gives

$$\begin{aligned}
-\frac{r_x}{2}U_{J-1,k}^{n+1/2} + (1+r_x)U_{J,k}^{n+1/2} - \frac{r_x}{2}U_{J+1,k}^{n+1/2} &= \frac{r_y}{2}U_{J,k-1}^{n} + (1-r_y)U_{J,k}^{n} + \frac{r_y}{2}U_{J,k+1}^{n} + \frac{\Delta t}{2} \\
-\frac{r_x}{2}U_{J-1,k}^{n+1/2} + (1+r_x)U_{J,k}^{n+1/2} &= \frac{r_y}{2}U_{J,k-1}^{n} + (1-r_y)U_{J,k}^{n} + \frac{r_y}{2}U_{J,k+1}^{n} + \frac{\Delta t}{2}.
\end{aligned}$$

Thus for each $k = 2, 3, \ldots, K-1$, we have the following tridiagonal linear system.

$$A_p U_{:,k}^{n+1/2} = \begin{bmatrix} 1+r_x & -\frac{r_x}{2} & & & \\ -\frac{r_x}{2} & 1+r_x & -\frac{r_x}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{r_x}{2} & 1+r_x & -\frac{r_x}{2} \\ & & & -\frac{r_x}{2} & 1+r_x \end{bmatrix} \begin{bmatrix} U_{1,k}^{n+1/2} \\ U_{2,k}^{n+1/2} \\ \vdots \\ U_{J-1,k}^{n+1/2} \\ U_{J,k}^{n+1/2} \end{bmatrix}$$

$$= \frac{r_y}{2} U_{:,k-1}^n + (1-r_y) U_{:,k}^n + \frac{r_y}{2} U_{:,k+1}^n + \frac{\Delta t}{2} \mathbf{1}_J = \mathbf{F}_p(k), \tag{16}$$

where $\mathbf{1}_J$ is a J-vector of all 1's.

Conveniently, the $J \times J$ matrix A_p is the same for each value of k, so that will only need to be constructed once. However, the forcing term $\mathbf{F}_p(k)$ must be calculated for each k. For $k = 1, K$, the boundary conditions (8) and (10) give a slight modification to (16).

$$A_p U_{:,1}^{n+1/2} = (1-r_y) U_{:,1}^n + \frac{r_y}{2} U_{:,2}^n + \frac{\Delta t}{2} \mathbf{1}_J = \mathbf{F}_p(1) \tag{17}$$

$$A_p U_{:,K}^{n+1/2} = \frac{r_y}{2} U_{:,K-1}^n + (1-r_y) U_{:,K}^n + \frac{\Delta t}{2} \mathbf{1}_J = \mathbf{F}_p(K) \tag{18}$$

Finally, for $n = 0$, the initial conditions (6) further simplifies the forcing term to $\mathbf{F}_p(k) = \frac{\Delta t}{2} \mathbf{1}_J$ for each k. However, by initializing the first layer to a grid of all zeros, the algorithm still applied.

Corrector Equations

As mentioned, the *corrector step* uses $U_{j,k}^{n+1/2}$ to solve for $U_{j,k}^n$, completing one entire iteration of the ADI algorithm. This time we use a backwards in time approximation for u_t (from t_{n+1} to $t_{n+1/2}$), a centered approximation for u_{yy} at t_{n+1}, and a centered approximation for u_{xx} at $t_{n+1/2}$. The derivation of the difference scheme is the same, with x and y switched, replacing $t_{n+1/2}$ with t_{n+1}, and replacing t_n with $t_{n+1/2}$. That is, (14) becomes the following.

$$U_{j,k}^{n+1} - \frac{r_y}{2} \delta_y^2 U_{j,k}^{n+1} = U_{j,k}^{n+1/2} + \frac{r_x}{2} \delta_x^2 U_{j,k}^{n+1/2} + \frac{\Delta t}{2}$$

$$-\frac{r_y}{2} U_{j,k-1}^{n+1} + (1+r_y) U_{j,k}^{n+1} - \frac{r_y}{2} U_{j,k+1}^{n+1} = \frac{r_x}{2} U_{j-1,k}^{n+1/2} + (1-r_x) U_{j,k}^{n+1/2} + \frac{r_x}{2} U_{j+1,k}^{n+1/2} + \frac{\Delta t}{2} \tag{19}$$

From (7)–(10) and (19), we have the following tridiagonal system for $j = 2, 3, \ldots, J-1$.

$$A_c U_{j,:}^{n+1} = \begin{bmatrix} 1+r_y & -\frac{r_y}{2} & & & \\ -\frac{r_y}{2} & 1+r_y & -\frac{r_y}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{r_y}{2} & 1+r_y & -\frac{r_y}{2} \\ & & & -\frac{r_y}{2} & 1+r_y \end{bmatrix} \begin{bmatrix} U_{j,1}^{n+1} \\ U_{j,2}^{n+1} \\ \vdots \\ U_{j,K-1}^{n+1} \\ U_{j,K}^{n+1} \end{bmatrix}$$

$$= \frac{r_x}{2} U_{j-1,:}^{n+1/2} + (1-r_x) U_{j,:}^{n+1/2} + \frac{r_x}{2} U_{j+1,:}^{n+1/2} + \frac{\Delta t}{2} \mathbf{1}_K = \mathbf{F}_c(j), \tag{20}$$

where $\mathbf{1}_K$ is a K-vector of all 1's.

Similar to the previous case, the $K \times K$ matrix A_c is independent of j, but the forcing term $\mathbf{F}_c(j)$ must be calculated for each j. For $j = 1, J$, the boundary conditions (7) and (9) yield a simplification to (20).

$$A_c U_{1,:}^{n+1} = (1 - r_x) U_{1,:}^{n+1/2} + \frac{r_x}{2} U_{2,:}^{n+1/2} + \frac{\Delta t}{2} \mathbf{1}_K = \mathbf{F}_c(1)$$
$$A_c U_{J,:}^{n+1} = \frac{r_x}{2} U_{J-1,:}^{n+1/2} + (1 - r_x) U_{J,:}^{n+1/2} + \frac{\Delta t}{2} \mathbf{1}_K = \mathbf{F}_c(J)$$

The Algorithm

We can now detail the entire ADI procedure. The implementation will be given shortly.

procedure ADI(J, K, N)
    Initialize x, y, and t. ▷ Use `linspace()`.
    Calculate Δx, Δy, and Δt.
    Initialize the $J \times J$ matrix A_p
    Initialize the $K \times K$ matrix A_c
    $U \leftarrow \text{zeros}(J, K, N)$
    for $t = 0, t_1, \ldots, t_{N-1}$ **do** ▷ Main loop
        $U^{n+1/2} \leftarrow \text{zeros}(J, K)$ ▷ Temporary grid for $t_{n+1/2}$
        $\mathbf{F} \leftarrow (1 - r_y) U_{:,1}^n + \frac{r_y}{2} U_{:,2}^n + \frac{\Delta t}{2}$ ▷ PREDICTOR STEP; first column
        $U_{:,1}^{n+1/2} \leftarrow \text{TRIDIAG}(A_p, \mathbf{F})$
        for $k = 2, \ldots, K-1$ **do**
            $\mathbf{F} \leftarrow \frac{r_y}{2} U_{:,k-1}^n + (1 - r_y) U_{:,k}^n + \frac{r_y}{2} U_{:,k+1}^n + \frac{\Delta t}{2}$ ▷ Interior columns
            $U_{:,k}^{n+1/2} \leftarrow \text{TRIDIAG}(A_p, \mathbf{F})$
        $\mathbf{F} \leftarrow \frac{r_y}{2} U_{:,K-1}^n + (1 - r_y) U_{:,K}^n + \frac{\Delta t}{2}$ ▷ Last column
        $U_{:,K}^{n+1/2} \leftarrow \text{TRIDIAG}(A_p, \mathbf{F})$
        $\mathbf{F} \leftarrow (1 - r_x) U_{1,:}^{n+1/2} + \frac{r_x}{2} U_{2,:}^{n+1/2} + \frac{\Delta t}{2}$ ▷ CORRECTOR STEP; first row
        $U_{1,:}^{n+1} \leftarrow \text{TRIDIAG}(A_c, \mathbf{F})$
        for $j = 2, \ldots, J-1$ **do**
            $\mathbf{F} \leftarrow \frac{r_x}{2} U_{j-1,:}^{n+1/2} + (1 - r_x) U_{j,:}^{n+1/2} + \frac{r_x}{2} U_{j+1,:}^{n+1/2} + \frac{\Delta t}{2}$ ▷ Interior rows
            $U_{j,:}^{n+1} \leftarrow \text{TRIDIAG}(A_c, \mathbf{F})$
        $\mathbf{F} \leftarrow \frac{r_x}{2} U_{J-1,:}^{n+1/2} + (1 - r_x) U_{J,:}^{n+1/2} + \frac{\Delta t}{2}$ ▷ last row
        $U_{J,:}^{n+1} \leftarrow \text{TRIDIAG}(A_c, \mathbf{F})$
    return U

Algorithm 1

Tridiagonal System Solver

Since A_p and A_c are tridiagonal, we can use a special strategy to solve the systems $A\mathbf{U} = \mathbf{F}$ quickly, providing an implementation for the TRIDIAG operations of Algorithm 1.

Crout Factorization

We begin with a modified LU-decomposition, proposed by Crout, in which the L is lower triangular with zeros below the first subdiagonal, U is upper triangular with zeros above the first superdiagonal, and U has all 1s on the diagonal. This is possible since the original matrix A is tridiagonal.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & \\ & \ddots & \ddots & \ddots & & \\ & & a_{N-1,N-2} & a_{N-1,N-1} & a_{N-1,N} & \\ & & & a_{N,N-1} & a_{N,N} \end{bmatrix}$$

$$= \begin{bmatrix} l_{1,1} & 0 & & & \\ l_{2,1} & l_{2,2} & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & l_{N-1,N-2} & l_{N-1,N-1} & 0 \\ & & & l_{N,N-1} & l_{N,N} \end{bmatrix} \begin{bmatrix} 1 & u_{1,2} & & & \\ 0 & 1 & u_{2,3} & & \\ & \ddots & \ddots & \ddots & \\ & & 0 & 1 & u_{N-1,N} \\ & & & 0 & 1 \end{bmatrix} = LU$$

Carrying out the matrix multiplication of L and U gives the following.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{N-1,N-2} & a_{N-1,N-1} & a_{N-1,N} \\ & & & a_{N,N-1} & a_{N,N} \end{bmatrix}$$

$$= \begin{bmatrix} l_{1,1} & l_{1,1}u_{1,2} & & & \\ l_{2,1} & l_{2,1}u_{1,2} + l_{2,2} & l_{2,2}u_{2,3} & & \\ & \ddots & \ddots & \ddots & \\ & & l_{N-1,N-2} & l_{N-1,N-2}u_{N-2,N-1} + l_{N-1,N-1} & l_{N-1,N-1}u_{N-1,N} \\ & & & l_{N,N-1} & l_{N,N-1}u_{N-1,N} + l_{N,N} \end{bmatrix}$$

Then $l_{i+1,i} = a_{i+1,i}$ for each i. That is, the subdiagonal of A is the subdiagonal of L. We can then use this subdiagonal to solve for the superdiagonal of U, and combine the information to get the diagonal of L. The process is detailed in Algorithm 2.

```
1: procedure CroutFactorization(A)
2:     N, N ← size(A)                          ▷ Store the dimensions of A.
3:     L ← zeros(N, N)                         ▷ Start L as all zeros.
4:     U ← Id(N)                               ▷ Start U as the identity.
5:     L_{1,1} ← A_{1,1}
6:     for i = 1 ... N − 1 do
7:         U_{i,i+1} ← A_{i,i+1}/L_{i,i}       ▷ Superdiagonal of U
8:         L_{i+1,1} ← A_{i+1,i}               ▷ Subdiagonal of L
9:         L_{i+1,i+1} ← A_{i+1,i+1} − L_{i+1,i}U_{i,i+1}   ▷ Diagonal of L
10:    return L, U
```

Algorithm 2

Forward and Backward Substitution

Now to solve the system $LU\mathbf{x} = A\mathbf{x} = \mathbf{b}$, we solve two problems in succession:

1. $L\mathbf{y} = \mathbf{b}$ by forward substitution, and
2. $U\mathbf{x} = \mathbf{y}$ by bacwards substitution.

The algorithm with these two steps is given below.

```
1: procedure TRIDIAG(A, b)
2:     N, N ← size(A)                          ▷ Store the dimensions of A.
3:     L, U ← CroutFactorization (A)
4:     y ← zeros(N)
5:     y_1 ← b_1/l_{1,1}                       ▷ FORWARD SUBSTITUTION
6:     for i = 2, ..., N do
7:         y_i ← (b_i − l_{i,i−1}y_{i−1})/l_{i,i}
8:     x ← zeros(N)
9:     x_N ← y_N                               ▷ BACKWARD SUBSTITUTION
10:    for i = N − 1, ..., 1 do
11:        x_i ← y_i − u_{i,i+1}x_{i+1}
12:    return x
```

Algorithm 3

Implementation

We begin with a function that combines and implements Algorithms 2 and 3.

```
function x = tridiag(A, b)

%% Factor A into L and U (Crout Factorization).
[N,~] = size(A);
L = zeros(N,N);
U = eye(N);

L(1,1) = A(1,1);
for i=1:N-1
    U(i,i+1) = A(i,i+1) / L(i,i);                    % Superdiagonal
    L(i+1,i) = A(i+1,i);                             % Subdiagonal
    L(i+1,i+1) = A(i+1,i+1) - L(i+1,i)*U(i,i+1);     % Main diagonal
end

%% Forward Substitution: Ly = b.
y = zeros(N,1);
y(1) = b(1) / L(1,1);
for i=2:N
    y(i) = (b(i) - L(i,i-1)*y(i-1)) / L(i,i);
end

%% Backward Substitution: Ux = y.
x = zeros(N,1);
x(N) = y(N);
for i=N-1:-1:1
    x(i) = y(i) - U(i,i+1)*x(i+1);
end

end
```

Now with the previous function, the following code implements Algorithm 1.

```
function U = laminar_flow_ibvp(J, K, T, dt)

% Calculate spatial step and related quantities.
dx = 1/(J+1);
dy = 1/(K+1);
dt2 = dt / 2;
N = ceil(T / dt) + 1;
rx = dt / dx^2;
ry = dt / dy^2;
rx2 = rx / 2;
ry2 = ry / 2;

% Construct predictor step matrix.
off_diags = ones(1,J-1) * -rx2;
Ap = eye(J)*(1 + rx) + diag(off_diags, -1) + diag(off_diags, 1);
```

```
% Construct corrector step matrix.
off_diags = ones(1,K-1) * -ry2;
Ac = eye(K)*(1 + ry) + diag(off_diags, -1) + diag(off_diags, 1);

% Iteratively compute the interior solution grid.
U_int = zeros(J,K,N);
for n=1:N-1
    U_temp = zeros(J,K);

    % PREDICTOR STEP
    F = (1-ry)*U_int(:,1,n) + ry2*U_int(:,2,n) + dt2;
    U_temp(:,1) = tridiag(Ap, F);           % First column.
    for k=2:K-1
        F = ry2*U_int(:,k-1,n) + (1-ry)*U_int(:,k,n) + ry2*U_int(:,k+1,n) + dt2;
        U_temp(:,k) = tridiag(Ap, F);       % Interior columns.
    end
    F = ry2*U_int(:,K-1,n) + (1-ry)*U_int(:,K,n) + dt2;
    U_temp(:,K) = tridiag(Ap, F);           % Last column.

    % CORRECTOR STEP
    F = (1-rx)*U_temp(1,:) + rx2*U_temp(2,:) + dt2;
    U_int(1,:,n+1) = tridiag(Ac, F');       % First row.
    for j=2:J-1
        F = rx2*U_temp(j-1,:) + (1-rx)*U_temp(j,:) + rx2*U_temp(j+1,:) + dt2;
        U_int(j,:,n+1) = tridiag(Ac, F)';   % Interior rows.
    end
    F = rx2*U_temp(J-1,:) + (1-rx)*U_temp(J,:) + dt2;
    U_int(J,:,n+1) = tridiag(Ac, F');       % Last row.
end

% Add boundary layers for final solution.
U = zeros(J+2,K+2,N);
U(2:J+1,2:K+1,:) = U_int;

end
```

Results

Using $T = 5$ and $\Delta t = .01$, we get very good results. In Figure 2 we compare the exact solution (left) to the numerical solution (right) at $t = .05, .1$, and 2. We also summarize numerical results in Tables 1 and 2 (where U_n is the complete grid of the numerical solution at $t = t_n$ and $\hat{U}_n$ is the grid of the exact solution at $t = t_n$) on the next few pages.

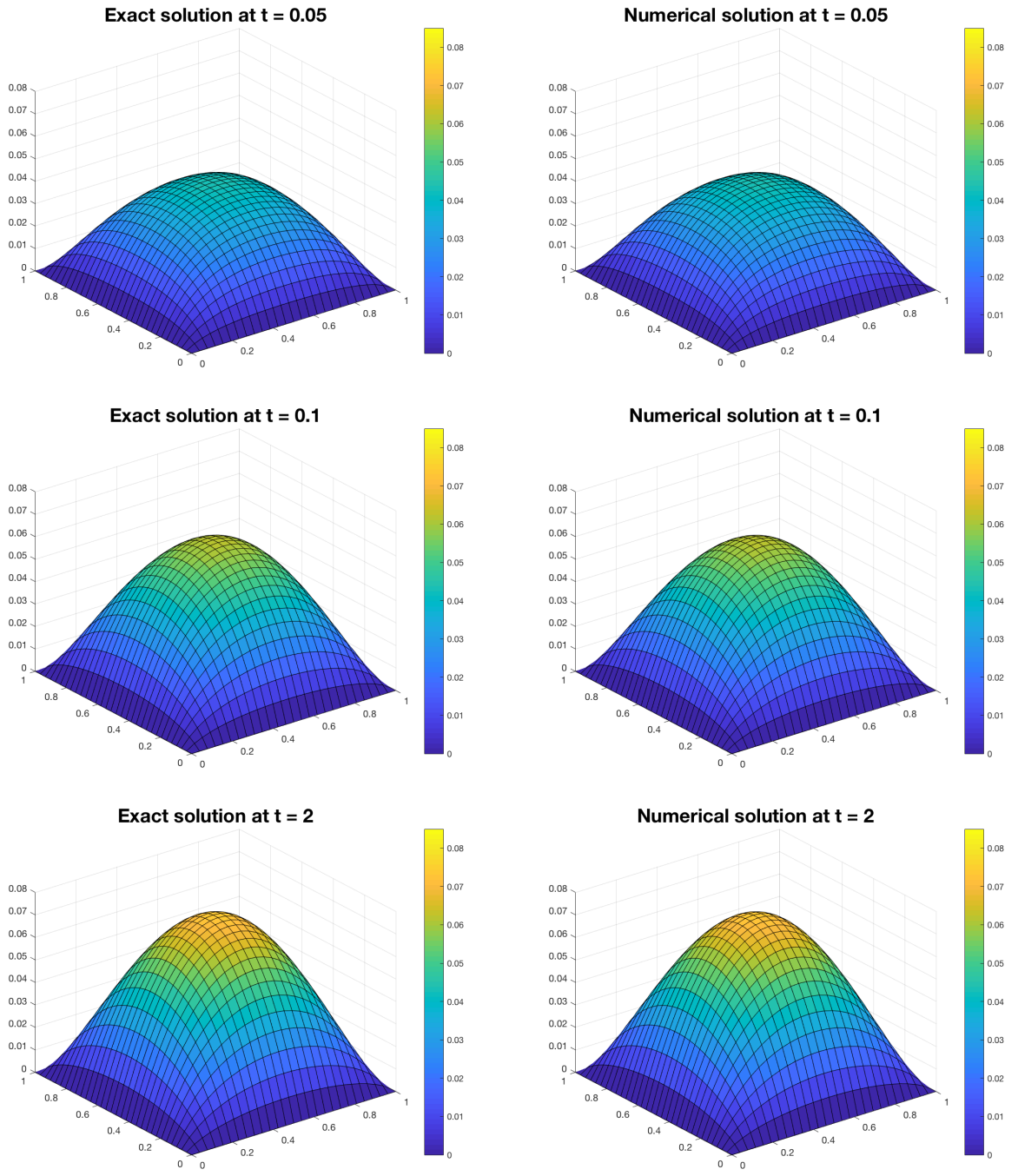

Exact solution at t = 0.05
Numerical solution at t = 0.05
Exact solution at t = 0.1
Numerical solution at t = 0.1
Exact solution at t = 2
Numerical solution at t = 2

Figure 2

t	Argmax of Numerical Velocity	Numerical Value	Exact Value
0.05	(0.5000, 0.4839)	0.043047398	0.043113997
0.10	(0.5000, 0.4839)	0.062146606	0.062213034
0.20	(0.5000, 0.4839)	0.071950852	0.072023573
0.50	(0.5000, 0.4839)	0.073529101	0.073602060
1.00	(0.5000, 0.4839)	0.073533348	0.073606303
2.00	(0.5000, 0.4839)	0.073533348	0.073606303

Table 1

t	$\|U_n - U_{n-1}\|_\infty$	$\|U_n - U_{n-1}\|_2$	$\|U_n - \hat{U}_n\|_\infty$	$\|U_n - \hat{U}_n\|_2$
0.05	1.310596903e-01	9.473115110e-02	1.302338009e-03	9.555336660e-04
0.10	4.901773861e-02	3.531044775e-02	1.406306688e-03	1.123784471e-03
0.15	1.829019031e-02	1.316330059e-02	1.493784707e-03	1.194585494e-03
0.20	6.815239969e-03	4.907144766e-03	1.522946958e-03	1.212994596e-03
0.25	2.541438623e-03	1.829336369e-03	1.527763426e-03	1.216900614e-03
0.30	9.472194126e-04	6.819593521e-04	1.528359971e-03	1.217188419e-03
0.35	3.531720105e-04	2.542280739e-04	1.527861983e-03	1.216873251e-03
0.40	1.316425518e-04	9.477386086e-05	1.527482990e-03	1.216594575e-03
0.45	4.908013900e-05	3.533081580e-05	1.527247994e-03	1.216431634e-03
0.50	1.829508804e-05	1.317100072e-05	1.527131909e-03	1.216348585e-03
0.55	6.820717553e-06	4.910027129e-06	1.527076054e-03	1.216309400e-03
0.60	2.542549709e-06	1.830412683e-06	1.527051153e-03	1.216291699e-03
0.65	9.478872256e-07	6.823609193e-07	1.527040158e-03	1.216283956e-03
0.70	3.533480543e-07	2.543778417e-07	1.527035482e-03	1.216280640e-03
0.75	1.317299534e-07	9.482970791e-08	1.527033504e-03	1.216279245e-03
0.80	4.910604963e-08	3.535163871e-08	1.527032686e-03	1.216278666e-03
0.85	1.830683015e-08	1.317876419e-08	1.527032348e-03	1.216278427e-03
0.90	6.824434236e-09	4.912921543e-09	1.527032211e-03	1.216278330e-03
0.95	2.544147799e-09	1.831491675e-09	1.527032155e-03	1.216278291e-03
1.00	9.484133427e-10	6.827632492e-10	1.527032133e-03	1.216278275e-03
1.05	3.535668563e-10	2.545277510e-10	1.527032124e-03	1.216278269e-03
1.10	1.318040937e-10	9.488564047e-11	1.527032121e-03	1.216278266e-03
1.15	4.913549081e-11	3.537250448e-11	1.527032119e-03	1.216278265e-03
1.20	1.831742744e-11	1.318649414e-11	1.527032119e-03	1.216278265e-03
1.25	6.829288871e-12	4.915732176e-12	1.527032118e-03	1.216278265e-03
1.30	2.545508943e-12	1.832554838e-12	1.527032118e-03	1.216278265e-03
1.35	9.488850677e-13	6.832269997e-13	1.527032118e-03	1.216278265e-03
1.40	3.545427840e-13	2.546074380e-13	1.527032118e-03	1.216278265e-03
1.45	1.322379706e-13	9.490238179e-14	1.527032118e-03	1.216278265e-03
1.50	4.894869232e-14	3.545451651e-14	1.527032118e-03	1.216278265e-03
1.55	1.862399124e-14	1.324934621e-14	1.527032118e-03	1.216278265e-03
1.60	7.388187284e-15	4.911297749e-15	1.527032118e-03	1.216278265e-03
1.65	3.576999807e-15	1.885586964e-15	1.527032118e-03	1.216278265e-03
1.70	1.863093013e-15	8.936745035e-16	1.527032118e-03	1.216278265e-03
1.75	9.332812301e-16	3.906719568e-16	1.527032118e-03	1.216278265e-03
1.80	1.693090113e-15	7.831296786e-16	1.527032118e-03	1.216278265e-03
1.85	8.638922910e-16	3.287965650e-16	1.527032118e-03	1.216278265e-03
1.90	8.326672685e-16	3.316074407e-16	1.527032118e-03	1.216278265e-03
1.95	1.210836986e-15	5.276288104e-16	1.527032118e-03	1.216278265e-03
2.00	1.882174971e-15	6.437272748e-16	1.527032118e-03	1.216278265e-03
4.00	8.586881206e-16	4.258631331e-16	1.527032118e-03	1.216278265e-03
5.00	8.552186737e-16	3.588093483e-16	1.527032118e-03	1.216278265e-03

Table 2

From the Figures, it is easy to see that the numerical solution matches the exact solution

qualitatively. There are two important observatons to make from Table 1 1:

- For each time step t_n, the maximum velocity given by $\max_{x,y} u(x, y, t_n)$ occurs at the center of the domain, roughly $(x, y) = (\frac{1}{2}, \frac{1}{2})$. It appears that the soution is symmetric and invariant to rotations in the x-y plane.

- The solution very quickly stops growing in time, reaching a steady state where the effects of the pressure (forcing term) and the diffusion (heat equation) cancel each other out. The steady state reaches a maximum velocity of about 0.07361.

Our particular IBVP square spatial domain, but I wonder how different it would be to use a circular domain and do the problem in polar coordinates (like the previous project). While using polar coordinates would likely reflect a more physical problem (fluid flowing through a cylindrical pipe instead of a rectangular pipe), it would likely be difficult to maintain a completely tridiagonal matrix structure due to the periodicity of the angular direction θ.

Table 2 gives a better sense for how quickly the system converges to the steady state. For $t_n = 1.75$, $t_{n-1} = 1.74$, the difference $\|U_n - U_{n-1}\|$ is already on the order of $\epsilon_{\text{machine}} \approx 10^{-16}$ (in both the L^2 and L^∞ norms, for both the numerical and exact solutions), which signals that evolution has all but ceased at this point. On the other hand, the difference between the numerical solution U_n and the exact solution $\hat{U}_n$ stays consistently around $\|U_n - \hat{U}_n\| \approx 10^{-3}$. This indicates that the numerical method is a good approximation for the exact solution, though it does not necessarily show consistency or convergence because we aren't refining the grid. However, by setting $J = 50$ and $K = 60$, the error drops to $\|U_n - \hat{U}_n\| \approx 10^{-4}$, which does give some evidence for consistency and convergence.

The ADI method is extremely fast, which is one of the big advantages over Crank-Nicholson and related algorithms. To give some perspective, computing the exact solution with 30 series terms for about 500 time steps takes about 20 seconds, but calculating the numerical solution for the same domain only takes about 1.5 seconds (on my computer). The big difference comes from using a TRIDIAG to solve $A\mathbf{U} = \mathbf{F}$ at each step, as opposed to solving the pentadiagonal (or worse) system that accompanies traditional methods like Crank-Nicholson.

Another advantage of ADI is the simplicity of the algorithm. For instance, the strength of the forcing term can be quickly changed by replacing $\frac{\Delta t}{2}$ with $\frac{\Delta t}{2} f(x, y, t)$ in each computation of $\mathbf{F}$ (to see an example of this, run `laminar_fluct_ibvp()` instead of `laminar_flow_ibvp()` as part of `main.m`). A change to the existing Dirichlet boundary terms would also be fairly simple, and could be implemented without destroying the tridiagonal structure of A by modifying $\mathbf{F}$ carefully. I suspect, though, that changing to a Neumann condition (for example, the free-slip BCs) would be more difficult.

Appendix: Matrix and Vector Key

$$U^n_{j,:} = \begin{bmatrix} U^n_{j,1} \\ U^n_{j,2} \\ \vdots \\ U^n_{j,K-1} \\ U^n_{j,K} \end{bmatrix}, \qquad U^n_{:,k} = \begin{bmatrix} U^n_{1,k} \\ U^n_{2,k} \\ \vdots \\ U^n_{J-1,k} \\ U^n_{J,k} \end{bmatrix}$$

$$A_p = \begin{bmatrix} 1+r_x & -\frac{r_x}{2} & & & \\ -\frac{r_x}{2} & 1+r_x & -\frac{r_x}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{r_x}{2} & 1+r_x & -\frac{r_x}{2} \\ & & & -\frac{r_x}{2} & 1+r_x \end{bmatrix}$$

$$\mathbf{F}_p(1) = (1-r_y)U^n_{:,1} + \frac{r_y}{2}U^n_{:,2} + \frac{\Delta t}{2}\mathbf{1}_J$$

$$\mathbf{F}_p(k) = \frac{r_y}{2}U^n_{:,k-1} + (1-r_y)U^n_{:,k} + \frac{r_y}{2}U^n_{:,k+1} + \frac{\Delta t}{2}\mathbf{1}_J, \quad k = 2,3,\ldots,K-1$$

$$\mathbf{F}_p(K) = \frac{r_y}{2}U^n_{:,K-1} + (1-r_y)U^n_{:,K} + \frac{\Delta t}{2}\mathbf{1}_J$$

$$A_c = \begin{bmatrix} 1+r_y & -\frac{r_y}{2} & & & \\ -\frac{r_y}{2} & 1+r_y & -\frac{r_y}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{r_y}{2} & 1+r_y & -\frac{r_y}{2} \\ & & & -\frac{r_y}{2} & 1+r_y \end{bmatrix}$$

$$\mathbf{F}_c(1) = (1-r_x)U^{n+1/2}_{1,:} + \frac{r_x}{2}U^{n+1/2}_{2,:} + \frac{\Delta t}{2}\mathbf{1}_K$$

$$\mathbf{F}_c(j) = \frac{r_x}{2}U^{n+1/2}_{j-1,:} + (1-r_x)U^{n+1/2}_{j,:} + \frac{r_x}{2}U^{n+1/2}_{j+1,:} + \frac{\Delta t}{2}\mathbf{1}_K, \quad j = 2,3,\ldots,J-1$$

$$\mathbf{F}_c(J) = \frac{r_x}{2}U^{n+1/2}_{J-1,:} + (1-r_x)U^{n+1/2}_{J,:} + \frac{\Delta t}{2}\mathbf{1}_K$$