Programs can be written in any language, and should be accompanied by a short description of how to compile and run them (you can put this in the comments at the head of the source code). Assignments will be submitted using git. All registered students should have received a document listing their username and password for bender.astro.sunysb.edu, along with some instructions on how to access their git repository.

1. (Implicit advection) Consider the linear advection equation:
\[ a_t + u a_x = 0 \]  
(1)

In class, we saw that an explicit first-order finite-difference upwind discretization of this resulted in a stable method. Here we consider an implicit discretization of this same upwind method—now the spatial derivative is evaluated at the new time.

a. Perform linear stability analysis (as we did in class) and show that this method is stable for any choice of Courant number.

b. Solve this implicit discretization numerically with periodic boundary conditions.

When you write this out, you will find that you have a coupled linear system of equations that can be written in a matrix form. The matrix is almost tridiagonal, except for a single element in a corner resulting from the periodicity. You can solve this system with any matrix solver routine you wish.

Make plots of the solution with 64 and 256 grid points with \( C = 0.5, 1, 10 \).

2. (Lax-Wendroff method) The Lax-Wendroff method can be derived by Taylor expanding in time, keeping terms to \( O(\Delta t^2) \), and then replacing the time-derivatives with spatial derivatives from the PDE. By using centered, second-order spatial derivatives, this results in a method that is second-order in space and time. For our advection equation, the update appears as:

\[ a_{i}^{n+1} = a_{i}^{n} - \frac{C}{2}(a_{i+1}^{n} - a_{i-1}^{n}) + \frac{C^2}{2}(a_{i+1}^{n} - 2a_{i}^{n} + a_{i-1}^{n}) \]  
(2)

This method can be shown to be stable—the explicitly modeled diffusion counteracts the numerical diffusion that made the method unstable (as shown in class).

Code this method up on a cell-centered finite-difference grid. For the initial conditions, choose a Gaussian:
\[ a(x, t = 0) = e^{-(x-0.5)^2/0.1^2} \]  
(3)

Evolve this with periodic boundary conditions on \( x = [0, 1] \).

We want to measure the convergence of this method. Convergence for PDEs requires that you change \( \Delta t \) in step with \( \Delta x \)—this is handled automatically if you use the same Courant number at each resolution. To get a single error number from the spatially discretized solution, we need to define a norm:

\[ \|\phi\|_2 = \left\{ \Delta x \sum_{i=0}^{N-1} \phi_i^2 \right\}^{1/2} \]  
(4)

This is called the L2 norm. We will define our error as:

\[ \epsilon_{\Delta x} = \|a(x, t = T) - a(x, t = 0)\|_2 \]  
(5)
where $T$ is one period. This is just the pointwise RMS error of the final solution compared to the initial solution, normalized by $\Delta x$.

Plot $\epsilon_{\Delta x}$ vs. $\Delta x$ for several grid resolutions and estimate the convergence rate.

Hint: depending on your choice of $\Delta t$, you may evolve past $t = T$—put a check in the evolution loop to make the final timestep smaller, if necessary, so you end exactly at $t = T$. (Note that if you reduce $\Delta t$ at the end, then $C$ reduces by a corresponding amount).

If you don’t see $O(\Delta t^2)$ convergence, then you likely have a bug somewhere.

3. (Nonlinear hyperbolic equations) In this problem we look at several first-order finite-difference methods for Burger’s equation:

$$u_t + uu_x = 0$$

This is a nonlinear equation. In conservative form, this appears as

$$u_t + \left[ \frac{1}{2} u^2 \right]_x = 0$$

Although these forms are analytically equivalent, the numerical solutions will differ.

a. Using first-order upwinding, we can difference Eq. 6 as

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x} u_i^n (u_i^n - u_{i-1}^n)$$

This is valid as long as $u_i^n > 0$. Write a program to solve this equation with initial conditions:

$$u(x, t = 0) = \begin{cases} 2 & \text{if } x < 0.5 \\ 1 & \text{if } x \geq 0.5 \end{cases}$$

and outflow boundary conditions. What type of solution do you see? Also run it with:

$$u(x, t = 0) = \begin{cases} 1 & \text{if } x < 0.5 \\ 2 & \text{if } x \geq 0.5 \end{cases}$$

Now what does the solution look like?

Important: This is a non-linear equation—make sure you use the proper CFL condition for this method. This means: pick a value for $C$, then evaluate $\Delta t = C \Delta x / \max_i \{u\}$

b. Now difference Eq. 7 as

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x} \left( \frac{1}{2} (u_i^n)^2 - \frac{1}{2} (u_{i-1}^n)^2 \right)$$

Write a program to solve this, and run it on the same initial data as in part a. Note that both of these discretizations assume that $u > 0$.

How do the solutions differ for the two discretizations? One of the initial conditions generates a shock—measure its speed for each method. How does the shock speed differ with resolution? Which method gets the shock speed correct? Why? Present your results at 3 different grid resolutions.