Meteorology 5344

Computational Fluid Dynamics

Computer Problem #5: Linear Advection Problem
Distributed: November 7, 2023
Due: December 5, 2023

1. Consider the 1-D linear convection equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \qquad (1)$$

where c is a positive and constant advection speed. This equation can be solved numerically using the two-step MacCormack method:

Predictor: $$(u_i^{n+1})^* = u_i^n - c\Delta t\frac{u_{i+1}^n - u_i^n}{\Delta x},$$

Corrector: $$u_i^{n+1} = \frac{1}{2}\left[u_i^n + (u_i^{n+1})^* - c\Delta t\frac{(u_i^{n+1})^* - (u_{i-1}^{n+1})^*}{\Delta x}\right].$$

a. Derive the modified equation for this two-step scheme and determine the anticipated error type (dispersive or dissipative). When trying to eliminate $(\quad)_t$, make sure you use FDE not PDE for the substitution.

b. Use the von Neumann technique to assess the stability of this scheme, and plot the phase and amplitude errors as a function of $k\Delta x$ for several Courant numbers, including a few for which linear stability is violated.

c. Write a computer code for this scheme, using as initial conditions the following function:

$$u(x,t=0) = 2 + u_0(x)\left[1+0.3\sin\left(\frac{2\pi x}{9\Delta x}\right)\right]\left[1+0.4\sin\left(\frac{2\pi x}{10\Delta x}\right)\right],$$

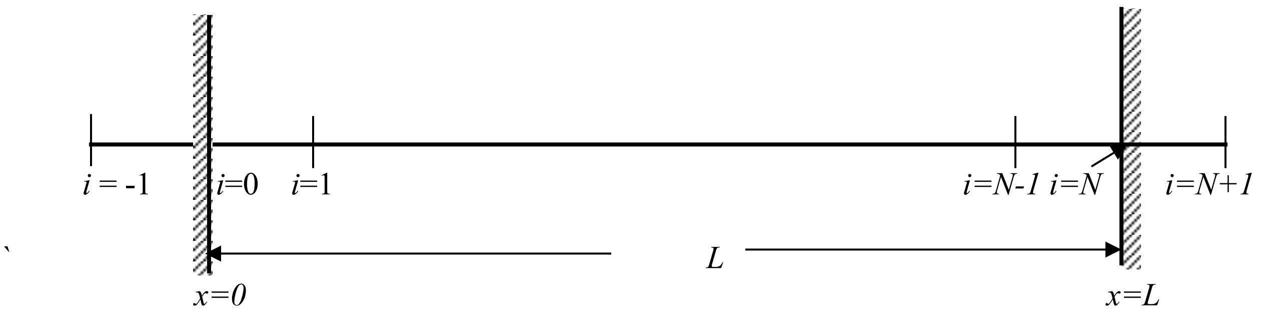
where

$$u_0(x) = \begin{cases} -1 & \text{if } 8 \le x \le 28 \\ 1 & \text{if } 28 < x \le 39, \\ 0 & \text{otherwise} \end{cases}$$

with $\Delta x = 1.0$ in a periodic domain of length 50. This somewhat unconventional initial condition provides a stringent test of advective schemes because it contains sharp gradients and other spatial irregularities.

The periodic boundary condition means that $u(x=0)=u(x=L)$, $u(x=-l)=u(x=(L-l))$ and $u(x=l)=u(x=(L+l))$, where L is the length of the physical domain.

To facilitate the implementation of periodic conditions at the lateral boundaries, we usually define an extra grid point outside each physical boundary, so for a physical domain of length L, $N = L/\Delta x$. Adding two boundary points outside the physical boundaries, the total number of grid points are N+3, as illustrated below:



In the discrete form, the boundary conditions are: $u(-1) = u(N-1)$ and $u(N+1) = u(1)$, where array u is declared as $u(-1{:}N+1)$. When these conditions are used, $u(N) = u(0)$) should be automatically satisfied. For actual implementation, you integrate the finite difference equation forward in time for $i = 0$ to N, and set boundary conditions at $i = -1$ and N+1. If you use Python arrays, you will have to shift your index to start at 0 instead of -1.

Assume $c = 1.0$, run your code with Courant numbers of 0. 10, 0.25, 0.50, and 1.00, show the plots for the numerical solutions at time t = 50.0, 100.0, 200.0, and 400.0, and discuss your results in light of what you know about this scheme based on the amplitude and phase error analyses. Is there anything special about solution with Courant number = 1?

d. For Courant numbers of 0. 10, 0.25, 0.50, and 1.00 and for t = 50.0, 100.0, 200.0, and 400.0, use Takacs' method to compute the amplitude and phase errors of numerical solutions relative to the exact solution (which is simply the initial condition shifted to the right by the number of time steps multiplied by the Courant number), and compare them with the theoretical predictions made in part b. Is the predominant error type similar to that anticipated from the modified equation? Comment on your findings.

2. For the same 1-D linear advection equation (1), consider a periodic domain of width 1.0 that is divided into 32 grid intervals with Δx=1/32 and c = 1.0. The initial condition is given by

$$u_0(x) = \begin{cases} \{64[(x-1/2)^2 - 1/64]\}^2 & \text{if } 3/8 \le x \le 5/8 \\ 0 & \text{otherwise} \end{cases}.$$

For the above, the feature being advected completes one pass through the domain in a time of 1.0.

a. Using the leapfrog time-differencing scheme and a second-order centered-in-space discretization for the advection term, run the solution to t = 3 using Courant numbers of 0.7, 0.5, and 0.1. Compute the dispersion, dissipation, and diffusion errors using Tackacs' method.

b. Repeat part a, this time adding the Asselin-Robert time filter with a filter coefficient of 0.1. Discuss the results.

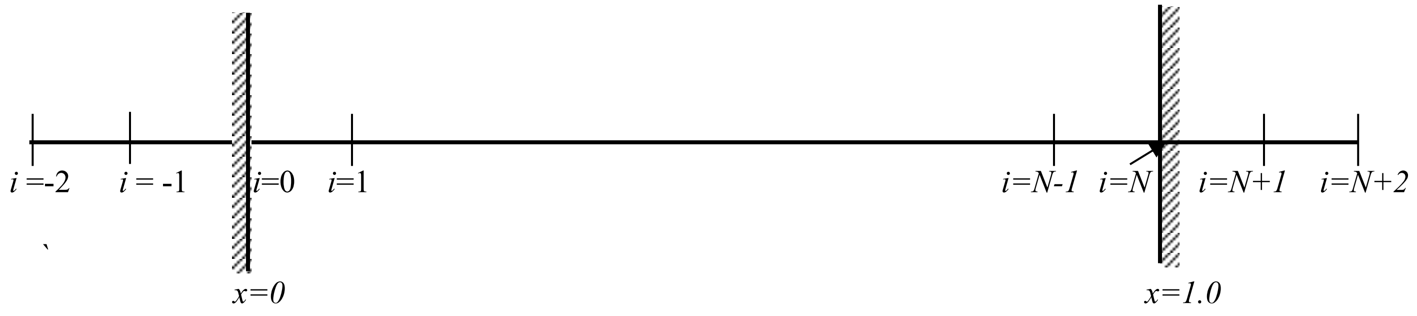
c. Perform a linear stability analysis on the leapfrog fourth-order centered-in-space scheme and derive its phase and amplitude changes per time step, and determine their errors as ratios to the analytical counterparts. Plots the error curves and compare them with those of leapfrog second-order-in-space scheme discussed in class. The leapfrog fourth-order centered-in-space scheme is given below:

$$\frac{u_i^{n+1}-u_i^{n-1}}{2\Delta t} = -c\left[\frac{4}{3}\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} - \frac{1}{3}\frac{u_{i+2}^n - u_{i-2}^n}{4\Delta x}\right].$$

d. Repeat part a, this time using a fourth-order centered-in-space difference for the advection term. Discuss the results.

The periodic boundary condition means that $u(x=0)=u(x=1)$, $u(-m\Delta x)=u(1.0-m\Delta x)$ and $u(m\Delta x)=u(1.0+m\Delta x)$, where *m* is any integer number, and 1.0 is the length of the physical domain.

To facilitate the implementation of periodic conditions at the lateral boundaries, we often define extra grid points outside the physical boundary. For the 4th-order difference scheme, two extra points are needed outside each boundary, as illustrated below:



In your Fortran code, it is convenient to define arrays as *u*(-2:*N*+2) where *N*=32 in this case. Then you boundary conditions will be like

u(-1) = *u*(*N*-1)
u(-2) = *u*(*N*-2)
u(N+1) = *u*(1)
u(N+2) = *u*(2)

u at *i* = 0 and *N* should be solved from the FDE. When the above conditions are used, *u*(*N*) = *u*(*0*)) should be automatically satisfied. For actual implementation, you integrate the finite difference equation forward in time for *i* = 0 to *N*, and set boundary conditions at *i* = -1, -2 and *N+1, N+2.* If you use Python arrays, you will have to shift your index to start at 0 instead of -2.