PHYS 643 Week 5: Introduction to Numerical Methods

These notes give an introduction to numerical methods for solving the fluid equations. They draw upon material in the book *Numerical Recipes* by Press & Teukolsky (see in particular section 19.1 of that book), and the course on hydrodynamics by P. Dullemond at the University of Heidelberg [1]. I focus here on finite differencing because this is the technique mostly used in astrophysics, where the geometry of the flow is usually quite simple, e.g. spherical (star or outflow), cylindrical (accretion disk) or plane-parallel (local box). An alternative technique is finite elements, used a lot in engineering applications that have complex geometries (e.g. flow around an aeroplane).

Finite difference approximation for derivatives

We solve for fluid properties on a numerical grid, at locations $x_j = j\Delta x$ where j labels the grid point. For simplicity here, we assume constant grid spacing Δx, although the results can be generalized to non-constant spacing. Quantities on neighbouring grid points are related by a Taylor expansion

$$f_{j+1} = f_j + \Delta x f'_j + \frac{(\Delta x)^2}{2} f''_j + \mathcal{O}(\Delta x^3)$$

$$f_{j-1} = f_j - \Delta x f'_j + \frac{(\Delta x)^2}{2} f''_j + \mathcal{O}(\Delta x^3).$$

Considering either of these gives a first order expression for the first derivative,

$$f'_j = \frac{f_j - f_{j-1}}{\Delta x} + \mathcal{O}(\Delta x) \qquad f'_j = \frac{f_{j+1} - f_j}{\Delta x} + \mathcal{O}(\Delta x).$$

Adding and subtracting instead gives a second order expression for the derivative and second derivative,

$$f'_j = \frac{f_{j+1} - f_{j-1}}{2\Delta x} + \mathcal{O}(\Delta x^2)$$

$$f''_j = \frac{f_{j+1} - 2f_j + f_{j-1}}{(\Delta x)^2} + \mathcal{O}(\Delta x^2).$$

The advection equation; numerical stability and numerical diffusion

First consider advection,

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} = 0.$$

Using our expressions for the derivatives, we might write

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = -v\frac{f_{j+1}^n - f_{j-1}^n}{2\Delta x},$$

[1] You can find the notes at
http://www.ita.uni-heidelberg.de/~dullemond/lectures/num_fluid_2011/index.shtml

where n labels the timestep. This gives an expression for the quantity f at the next timestep $n+1$ in terms of the value at the current timestep n:

$$f_j^{n+1} = f_j^n - \frac{v\Delta t}{2\Delta x}\left(f_{j+1}^n - f_{j-1}^n\right).$$

This is known as the forward-time centered-space (FTCS) scheme. This kind of scheme is referred to as *explicit* because the new values are written explicitly in terms of the old ones.

In fact, it turns out that this scheme is always numerically unstable. You can see this by looking for a solution

$$f_j^n = (\xi)^n e^{ikx_j},$$

where k is the wavevector and ξ is a complex amplitude. If $|\xi| > 1$ for any value of k, that mode will grow exponentially with increasing timestep n, and the numerical scheme is unstable. Trying a solution like this for the FTCS scheme gives

$$|\xi|^2 = 1 + \left(\frac{v\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x),$$

which is indeed greater than unity for any value of k.

Fortunately, there is a simple way to write a stable method, the Lax method:

$$f_j^{n+1} = \frac{1}{2}\left(f_{j+1}^n + f_{j-1}^n\right) - \frac{v\Delta t}{2\Delta x}\left(f_{j+1}^n - f_{j-1}^n\right).$$

This has

$$|\xi|^2 = 1 + \left[\left(\frac{v\Delta t}{\Delta x}\right)^2 - 1\right]\sin^2(k\Delta x),$$

and so we see that the scheme is stable as long as

$$\frac{v\Delta t}{\Delta x} \leq 1.$$

This condition on the timestep is the *Courant-Friedrichs-Levy criterion* or "Courant condition". The criterion states that our timestep must not exceed the fluid travel time between two grid points, which makes sense physically because the information about fluid quantities is advected at that speed. Larger timesteps require information from grid points further away than Δx, not included in our update.

A way to understand why the scheme is stable is to separate out the FTCS part and see what additional terms have been added. The Lax method can be rewritten

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = -v\frac{f_{j+1}^n - f_{j-1}^n}{2\Delta x} + \left(\frac{\Delta x^2}{2\Delta t}\right)\frac{f_{j+1}^n - 2f_j^n + f_{j-1}^n}{(\Delta x)^2}.$$

The additional term on the right is a diffusion term with diffusivity $(\Delta x)^2/2\Delta t$. This is known as *numerical diffusion*, it provides numerical dissipation that stabilizes the method. The damping is largest for short wavelengths where $k\Delta x \sim 1$ which are most unstable.

The Lax scheme provides a good illustration of different types of error:

- When $v\Delta t < \Delta x$, $|\xi| < 1$, giving an *amplitude error*: the amplitude of any given mode k decreases over time (it should stay constant under advection)

- *Phase error*. The factor ξ in the Lax scheme can be rewritten as

$$\xi = e^{-ik\Delta x} + i\left(1 - \frac{v\Delta t}{\Delta x}\right)\sin k\Delta x.$$

  For a timestep $\Delta t = \Delta x/v$, the phase of each mode is shifted by $k\Delta x$, equivalent to advecting by one grid point. But for timesteps $\Delta t < \Delta x/v$ the phase shift depends on k, so that different modes are advected at different speeds. Again, this should not happen under advection. The numerical method introduces dispersion as the component waves of the profile we are trying to advect move with different speeds.

- *Transport errors*: in the Lax scheme, the information from cells $j-1$ and $j+1$ propagates to cell j in the next timestep. But physically, if the velocity is to the right for example, only information in cell $j-1$ should be used to update cell j. A way around this is *upwind differencing* which avoids this problem, but at the expense of being first order:

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = -v_j \frac{f_j^n - f_{j-1}^n}{\Delta x} \qquad v_j^n > 0$$

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = -v_j \frac{f_{j+1}^n - f_j^n}{\Delta x} \qquad v_j^n < 0.$$

Everything we've discussed here is first order in time, but there are higher order methods that you can read about in Numerical Recipes. A useful one is *staggered-leapfrog* which uses a second-order time-derivative

$$f_j^{n+1} = f_j^{n-1} - \frac{v\Delta t}{2\Delta x}\left(f_{j+1}^n - f_{j-1}^n\right).$$

Numerically this requires storing the previous two timesteps in order to do the update. This method has the advantage that $|\xi| = 1$ for all modes no matter what timestep is used: the stability analysis gives

$$\xi = -i\frac{v\Delta t}{\Delta x}\sin k\Delta x \pm \sqrt{1 - \left(\frac{v\Delta t}{\Delta x}\sin k\Delta x\right)^2},$$

so while there is dispersion (the phase evolution is different for different modes), the amplitude of each mode stays constant, much better than the very dispersive first order Lax method. Note that staggered leapfrog also has a limit $\Delta t \leq \Delta x/v$ for stability.

The diffusion equation: implicit methods

In the case of diffusion, the simplest differencing that you might write down is stable for small enough timesteps. The update is

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = D\frac{f_{j+1}^n - 2f_j^n + f_{j-1}^n}{(\Delta x)^2},$$

with

$$\frac{D\Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

for stability. The physical interpretation is that the timestep is constrained by the diffusion time between grid cells.

Solving diffusion problems with explicit schemes is particularly slow, because the distance diffused in time t grows slowly with time, as $L \propto t^{1/2}$. The number of timesteps needed to follow diffusion over a lengthscale L is $L^2/D\Delta t \geq 2(L/\Delta x)^2 \sim N^2$ where N is the number of grid points.

An alternative scheme that allows larger timesteps, at the expense of accuracy on small scales, is an *implicit* scheme

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = D\frac{f_{j+1}^{n+1} - 2f_j^{n+1} + f_{j-1}^{n+1}}{(\Delta x)^2},$$

in which we write the update in terms of the values at the next timestep rather than at the current timestep (hence the name implicit). Rearranging, we can write

$$-\alpha f_{j+1}^{n+1} + (1+2\alpha)f_j^{n+1} - \alpha f_{j-1}^{n+1} = f_j^n$$

where $\alpha = D\Delta t/(\Delta x)^2$. Written as a matrix equation this is

$$Af^{n+1} = f^n$$

for the vectors f^{n+1} and f^n, where the matrix A is tridiagonal, with entries $1+2\alpha$ on the diagonal and $-\alpha$ on the upper and lower diagonals. This system can be solved by finding the inverse of the matrix A, since then $f^{n+1} = A^{-1}f^n$.

This *fully-implicit* scheme has the feature that it goes to the steady-state solution for large time-steps $\Delta t \to \infty$. Although small scales are not followed accurately for large timesteps, they go the correct steady-state solution. An alternative *semi-implicit* scheme is Crank-Nicholson

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = \frac{D}{(\Delta x)^2}\left[\frac{1}{2}\left(f_{j+1}^n - 2f_j^n + f_{j-1}^n\right) + \frac{1}{2}\left(f_{j+1}^{n+1} - 2f_j^{n+1} + f_{j-1}^{n+1}\right)\right]$$

which is also stable for large timesteps. It has the advantage that it is second order in both space and time, whereas fully-implicit is second order in space, but first order in time.

Operator splitting

You will often have multiple operators in the equation you are solving. A simple example is the *advection-diffusion* equation

$$\frac{\partial f}{\partial t} = -v\frac{\partial f}{\partial x} + D\frac{\partial^2 f}{\partial x^2}.$$

One way to deal with this is to calculate the update for each operator separately. Starting with f^n, generate $f^{n+\frac{1}{2}}$ by updating with the diffusion operator with timestep Δt, then update $f^{n+\frac{1}{2}}$ with the advection operator with timestep Δt to obtain the final values f^n.

Flux-conservative schemes

We know that the fluid equations arise from the conservation laws for mass, momentum and energy. We can take advantage of that and work with the equations in flux-conservative form, so that the numerical method exactly conserves these quantities.

In *finite-volume methods*, we divide the volume into cells such that the grid points x_j are the locations of the cell centres, and the cell boundaries are at locations $x_{j\pm1/2} = (1/2)(x_j + x_{j\pm1})$. We then solve the equation

$$\frac{\partial f}{\partial t} = -\frac{\partial J}{\partial x},$$

or in discretized form

$$\frac{f_j^{n+1} - f_j^n}{\Delta t} = -\frac{J_{j+\frac{1}{2}}^{n+\frac{1}{2}} - J_{j-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta x},$$

where we write the flux of quantity f at the cell boundaries $(j \pm 1/2)$ averaged over the timestep:

$$J_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \frac{1}{\Delta t}\int_t^{t+\Delta t} dt\, J_{j+\frac{1}{2}}(t).$$

This formulation automatically conserves the quantity f, since the flux out of one cell equals the flux into the neighbouring cell.

The simplest choice for the flux J is to write

$$J_{j+\frac{1}{2}} = v_{j+\frac{1}{2}} f_j^n \qquad v_{j+\frac{1}{2}} > 0$$

$$J_{j+\frac{1}{2}} = v_{j+\frac{1}{2}} f_{j+1}^n \qquad v_{j+\frac{1}{2}} < 0$$

$$J_{j-\frac{1}{2}} = v_{j-\frac{1}{2}} f_{j-1}^n \qquad v_{j-\frac{1}{2}} > 0$$

$$J_{j-\frac{1}{2}} = v_{j-\frac{1}{2}} f_j^n \qquad v_{j+\frac{1}{2}} < 0$$

which is known as *donor cell advection* (equivalent to the upwind differencing discussed earlier). Depending on the sign of the velocity, the contents are either advected out of cell j or into cell j from the left or right neighbour. The assumption here is that

the profile of f within the cell is well-approximated by a constant (given by the value at the center f_j). More complex assumptions about the profile of f give rise to higher order methods. For example, assuming f is linear across the cell (with slope chosen to be consistent with the difference in f between cell j and its neighbours) gives a scheme that is 2nd order in time. These piecewise linear schemes are discussed in detail in Chapter 4 of the Heidelberg notes I linked to earlier (see footnote on page 1).

Papers

This week, the paper discussion will involve researching a (magneto)hydrodynamics code such as PLUTO, PENCIL, FLASH, ZEUS, Castro.

You should discuss:

- What equations is the code solving?
- What numerical methods are used?
- What microphysics is included?
- What geometry can the code simulate?
- What are some of the applications that the code has been used for?