Computational Physics Project / Hydrodynamics Code

This project is largely copied from Andrew MacFadyen's "How To Write A Hydrodynamics Code." It walks through the implementation of a 1D and 2D hydrodynamics code.

The equations of one-dimensional hydrodynamics can be written in the conservation form as:

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} = 0 \tag{1}$$

where $\vec{U} = \{\rho, \rho v, E\}$ are the conserved variables for mass, momentum, and energy, and $\vec{F} = \{\rho v, \rho v^2 + P, (E + P)v\}$ are the fluxes of those quantities. ρ is the density, v is the velocity, P is the pressure, and $E = \rho e + \frac{1}{2}\rho v^2$ is the total energy density. The quantity e is the specific internal energy (i.e. internal energy per unit mass).

To close the equations requires an equation of state $P = P(\rho, e)$. For an ideal gas:

$$P = (\gamma - 1)\rho e, \tag{2}$$

where γ is the adiabitic index.

1. Low-order 1D Method

If we establish a 1D grid of cells along the axis x, whose centers are indexed by i and whose sizes are uniform with size Δx, and we know the fluxes at the interfaces of the cells at some time, the time derivative for $\vec{U}$ can be calculated discretely:

$$\frac{\partial \vec{U}}{\partial t} = L\left(\vec{U}\right) = -\frac{\vec{F}_{i+1/2} - \vec{F}_{i-1/2}}{\Delta x}, \tag{3}$$

where $i + 1/2$ refers to the interface between i and $i + 1$ and similarly for $i - 1/2$.

Then with first-order forward Euler we can evolve the conserved quantities:

$$\vec{U}(t_{n+1}) = \vec{U}(t_n) + \Delta t L\left(\vec{U}(t_n)\right), \tag{4}$$

where Δt is the time step.

We need to calculate the fluxes given the conditions in neighboring cells. We could interpolate the densities, velocities, and energies to calculate the fluxes, but this turns out not to be the best path forward. Instead, most work uses a variant of the Godunov method, which treats each interface as a little "Riemann" problem describing what happens at the interface of a 1D discontinuity. That is, imagine that the fluid literally had piecewise constant values of its properties; at the interfaces, the changes of density, pressure, etc, would drive waves. These little "shock tube" problems can

be solved and the flow of density, momentum density, and energy density in these waves can be calculated, and then used. This method turns out to be more accurate.

The exact solution to the Riemann problem is very expensive computationally; that is, too expensive to calculate for every cell in a simulation. But there are approximate solutions in common use. The Harten-Lax-van Leer (HLL) approximation is a classic efficient one:

$$\vec{F}^{\rm HLL} = \frac{\alpha^+ \vec{F}^L + \alpha^- \vec{F}^R - \alpha^+ \alpha^- \left(\vec{U}^R - \vec{U}^L \right)}{\alpha^+ + \alpha^-}. \tag{5}$$

where L and R refer to the "left" and "right" states (i and $i+1$, for example). The $\alpha^\pm$ values are given by:

$$\alpha^\pm = \texttt{max} \left\{ 0, \pm\lambda^\pm \left(\vec{U}^L \right), \pm\lambda^\pm \left(\vec{U}^R \right) \right\} \tag{6}$$

where

$$\begin{aligned} \lambda^\pm &= v \pm c_s \\ c_s &= \sqrt{\frac{\gamma P}{\rho}} \end{aligned} \tag{7}$$

It isn't trivial to say why this works; it is tuned approximation known to have nice properties. You might want to think of the velocities $\alpha^\pm$ as expressing something about the velocities of the waves emanating from the initial discontinuity between the two cells.

In implementing this method, you do not want one the waves to travel further than one cell size over the course of a time step. This is the Courant-Friedrich-Levy condition. Therefore the time step is constrained by:

$$\Delta t < \Delta x / \texttt{max} \left(\alpha^\pm \right), \tag{8}$$

which you should note needs to hold everywhere on the grid.

2. Prep Work

- Rescale the equations 14 and 2 to a set of unitless variables and to the minimum number of dimensionful parameters that you can. You should perform your numerical analysis in these variables, and rescale the solutions to physical variables at the end.
- Consider a set of unit tests that you will implement to test this code. An example might be a test of your `minmod` function, but consider other components of the code that will benefit from unit tests.

3. Sod Shock Tube Problem

An initial test of the code is to run a Sod shock tube problem. Note that this is basically solving on a grid the same Riemann problem that is approximated at the cell level. Perform the following with the low order code.

Set up your Python code so that you separate the initialization from the evolution. You should have a script which imports three modules:

- Initialization (with functions setting up an initial grid given parameters).
- Evolution (integrates the equations and returns a history of the evolution of the grid).
- Output (writes the output to a file or files).

Then *in a separate script* you should be able to input the final files and make plots (again, a very good idea would be to separate the functionality into two different modules).

- Initialize two halves of the tube (left and right) to different states, where:

$$\begin{aligned} \frac{P_L}{P_R} &= 8 \\ \frac{\rho_L}{\rho_R} &= 10 \end{aligned} \tag{9}$$

  and the velocity is zero. Set $\gamma = 1.4$. Plot and examine the density, pressure, and velocity. Exchange left and right and make sure the answer is exactly the same both ways.

- This problem can be solved exactly. Compare your answer to the exact answer using the code here. This requires compiling a piece of Fortran, and a minor change in the parameters near its beginning. Test several initial parameter settings.

- Test the convergence of the method by doubling and quadrupling the number of spatial grid points (make sure you handle the time steps consistently so as not to violate the Courant-Friedrich-Levy condition).

4. Higher Order 1D Method

We can reduce the approximation error by working at higher order in space and time.

In time we can use a version of third-order Runge-Kutta due to Shu % Osher. This method starts with $\vec{U}(t_n)$ and advances to $\vec{U}(t_{n+1})$ as follows:

$$\vec{U}^{(1)} = \vec{U}(t_n) + \Delta t L\left(\vec{U}(t_n)\right)$$

$$\begin{aligned} \vec{U}^{(2)} &= \frac{3}{4}\vec{U}(t_n) + \frac{1}{4}\vec{U}^{(1)} + \frac{1}{4}\Delta t L\left(\vec{U}^{(1)})\right) \\ \vec{U}(t_{n+1}) &= \frac{1}{3}\vec{U}(t_n) + \frac{2}{3}\vec{U}^{(2)} + \frac{2}{3}\Delta t L\left(\vec{U}(t_n)\right) \end{aligned} \tag{10}$$

In space we go to higher order by using more spatial points to determine the left and right conditions on each cell interface from which to calculate fluxes, instead of just the piecewise constant model. We cannot use a fully self-consistent interpolation, since that does not lead to a discontinuity! So the left conditions are calculated using a left-biased set of points, and the right conditions from a right-biased set of points.

Specifically, for $c = \rho$, P, and v (not the components of $\vec{U}$!):

$$c^L_{i+1/2} = c_i + \frac{1}{2}\texttt{minmod}\left[\theta\left(c_i - c_{i-1}\right), \frac{1}{2}\left(c_{i+1} - c_{i-1}\right), \theta\left(c_{i+1} - c_i\right)\right] \tag{11}$$

where θ is a parameter we can choose between 1 and 2, and:

$$\texttt{minmod}(x, y, z) = \frac{1}{4}\left|\mathrm{sgn}(x) + \mathrm{sgn}(y)\right| \left(\mathrm{sgn}(x) + \mathrm{sgn}(z)\right) \min\left(|x|, |y|, |z|\right) \tag{12}$$

For the right interface:

$$c^R_{i+1/2} = c_{i+1} + \frac{1}{2}\texttt{minmod}\left[\theta\left(c_{i+1} - c_i\right), \frac{1}{2}\left(c_{i+2} - c_i\right), \theta\left(c_{i+2} - c_{i+1}\right)\right] \tag{13}$$

- Implement this higher order implementation. Start it with the same initial conditions as above.
- Repeat the test against a known exact solution and repeat the convergence test. Do you see the expected improvement in the approximation error and its convergence properties?
- For at least three choices of grid size, perform a timing test using both the low order and high order cases, and compare their speed. Is the higher order faster than lower order at fixed *computation time*?

5. 2D Higher-order Method

Extending to higher number of spatial dimensions is straightforward for a simple regular mesh. For example, in two dimensions:

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0 \tag{14}$$

where in this case the fluxes and conserved quantities are generalized:

$$\vec{U} = \{\rho, \rho v_x, \rho v_y, E\}$$

$$\begin{aligned} \vec{F} &= \left\{\rho v_x, \rho v_x^2 + P, \rho v_x v_y, (E+P)v_x\right\} \\ \vec{G} &= \left\{\rho v_y, \rho v_x v_y, \rho v_y^2 + P, (E+P)v_y\right\} \end{aligned} \tag{15}$$

and the total energy is $E = \rho e + \frac{1}{2}\rho(v_x^2 + v_y^2)$.

In this case, the time derivative of the consierved quantities are:

$$\frac{\partial \vec{U}}{\partial t} = L\left(\vec{U}\right) = -\frac{\vec{F}_{i+1/2} - \vec{F}_{i-1/2}}{\Delta x} - \frac{\vec{G}_{i+1/2} - \vec{G}_{i-1/2}}{\Delta x} \tag{16}$$

The time steps can again be done as a third-order Runge-Kutta, and the high order version of the flux determinations can be performed in an analagous way as above.

You shoud implement this 2D case. The boundary conditions will matter a little more in this case, so be careful of that.

- Implement periodic boundary conditions in the y-direction, and try to implement the 1D Sod shock tube problem in the x-direction. Evaluate whether the system is stable.
- Implement periodic boundary conditions in both directions, and set initial conditions such that the top part of the fluid (high y) is moving relative to the bottom part (low y), and otherwise the fluids are identical. Examine how the fluid behaves under different choices of the relative velocity.
- Near the center of the grid (say about 1/10 of the radius), put a region with a density 100 times the rest of the grid. This should drive a blast wave outwards.

6. Bonus: Other tests

Consult Jim Stone's web page for some 1D and 2D tests of potential interest. Consider the Rayleigh-Taylor or the Double Mach reflection tests. Note both require a special boundary condition.