Computational Methods for the Euler Equations
In addition to the slides and code examples, my notes on PDEs with the finite-volume method are up online:

- https://github.com/Open-Astrophysics-Bookshelf/numerical_exercises
Gas Dynamics

- We’ll focus on 1-d hydrodynamics
- We want to solve:

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} &= 0 \\
\frac{\partial (\rho u)}{\partial t} + \frac{\partial}{\partial x} (\rho u^2 + p) &= 0 \\
\frac{\partial (\rho E)}{\partial t} + \frac{\partial}{\partial x} ((\rho E + p)u) &= 0
\end{align*}
\]
Linear Advection Equation

• The linear advection equation provides a simple problem to explore methods for hyperbolic problems

\[ a_t + u a_x = 0 \]

  – Here, \( u \) represents the speed at which information propagates

• First order, linear PDE

  – We'll see later that many hyperbolic systems can be written in a form that looks similar to advection, so what we learn here will apply later.
Linear Advection Equation

- We need initial conditions
  \[ a(x, 0) = a_0(x) \]

- and a single boundary condition, one of:
  \[ a(0, t) = a_l(t) \]
  \[ a(L, t) = a_r(t) \]
Linear Advection Equation

- Solution is trivial—any initial configuration simply shifts to the right (for $u > 0$)
  - e.g. $a(x - ut)$ is a solution
  - This demonstrates that the solution is constant on lines $x = ut$—these are called the characteristics
- This makes the advection problem an ideal test case
  - Evolve in a periodic domain
  - Compare original profile with evolved profile after 1 period
  - Differences are your numerical error
Gridded Data

- Discretized data is represented at a finite number of locations
  - Integer subscripts are used to denote the position (index) on the grid
  - Structured/regular: spacing is constant

- Data is known only at the grid points: $f_i = f(x_i)$
First Derivative / Order of Accuracy

- Taylor expansion:
  \[ f_{i+1} = f(x_i + \Delta x) = f_i + \frac{df}{dx} \bigg|_{x_i} \Delta x + \frac{1}{2} \frac{d^2 f}{dx^2} \bigg|_{x_i} \Delta x^2 + \ldots \]

- Solving for the first derivative:
  \[ \frac{df}{dx} \bigg|_{x_i} = \frac{f_{i+1} - f_i}{\Delta x} - \frac{1}{2} \frac{d^2 f}{dx^2} \bigg|_{x_i} \Delta x + \ldots \]

Discrete approximation to \( f' \)  
Leading term in the truncation error
First Derivative / Order of Accuracy

- This is a first-order accurate expression for the derivative at point $i$
  - Alternately, we can use the point to the left (blackboard)
  - These are called difference or finite-difference formulae
- Shorthand: $O(\Delta x)$
  - “big-O notation”
- How can we get higher order?
First Derivative / Order of Accuracy

- First derivative approximations:
  - First-order (one-sided):
    \[ f' = \frac{f_i - f_{i-1}}{\Delta x} \]
  - Second-order (centered):
    \[ f' = \frac{f_{i+1} - f_i}{\Delta x} \]
  - Fourth-order:
    \[ f' = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12\Delta x} \]

- Range of points involved is called the stencil
  - Some points may have a '0' coefficient
First Derivative Comparison

\[ f(x) = \sin(x) \]
\[ \text{at } x = \frac{\pi}{3} \]

- analytic: 0.5
- left-sided O(dx): 0.639529171481
- right-sided O(dx): 0.34028636457
- centered O(dx^2): 0.489907768026
- centered O(dx^4): 0.499756119208
Finite-Difference Approximation

- We store the function value at each point in our grid:

- We will use the notation: $$a_i^n$$
  - Superscripts = time discretization; subscripts = spatial discretization
  - We'll use 0-based indexing
- Simple discretization:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -u \frac{a_{i+1}^n - a_{i-1}^n}{2\Delta x}$$

- Explicit
- 2nd order in space, 1st order in time (FTCS method)
Boundary Conditions

- We want to be able to apply the same update equation to all the grid points:
  \[ a_{i}^{n+1} = a_{i}^{n} - \frac{C}{2} (a_{i+1}^{n} - a_{i-1}^{n}) \]

  Here, \( C = \frac{u \Delta t}{\Delta x} \) is the fraction of a zone we cross per timestep—this is called the Courant-Friedrichs-Lewy number (or CFL number).

- Notice that if we attempt to update zone \( i = 0 \) we “fall off” the grid.

- Solution: ghost points (for finite-volume, we'll call them ghost cells).

Grid for \( N \) points

PHY 688: Astrophysical Fluids and Plasmas
Boundary Conditions

- Before each timestep, we fill the ghost points with data that represents the boundary conditions.

- Note that with this discretization, we have a point exactly on each boundary (we only really need to update one of them).

- Periodic BCs would mean: $a_0^n = a_{N-1}^n$.
  - $a_N^n = a_1^n$; $a_{-1}^n = a_{N-2}^n$

- Other common BCs are outflow (zero derivative at boundary).
Implementation and Testing

- Recall that the solution is to just propagate any initial shape to the right:

  - We'll code this up with periodic BCs and compare after 1 period
  - On a domain [0,1], one period is simply: $1/u$

- We'll use a tophat initial profile
- Let's look at the code...

code: fdadvect.py
FTCS Linear Advection

- Evolution with $N=65$, $C=0.9$, after 1 period
  - Note the vertical scale!!!!
  - We see nothing that looks like a tophat
  - Any ideas?

code: fdadvect.py

PHY 688: Astrophysical Fluids and Plasmas
FTCS Linear Advection

- Reducing the timestep is equivalent to reducing the CFL number
  - CFL = 0.1, still 1 full period
  - The scale is much reduced, but it is still horrible
  - Let's look how these features develop

code: fdadvect.py
FTCS Linear Advection

- Here we evolve for only 1/10th of a period
  - CFL = 0.1
  - Notice that the oscillations are developing right near the discontinuities
  - So far we've looked at timestep, what about resolution?

code: fdadvect.py
• This is with \( N = 257 \) points
  
  – There is something more fundamental than resolution or CFL number going on here...

```python
code: fdadvect.py
```
Stability

- The problem is that FTCS is not stable.
- There are a lot of ways we can investigate the stability:
  - It is instructive to just work through an update using pencil + paper to see how things grow.
  - Growth of a single Fourier mode.
  - Truncation analysis.
  - Graphically: we can trace back the characteristics to see what information the solution depends on.
Fourier Mode Analysis of FTCS

- Consider a solution consisting of a single Fourier mode:
  \[ a_i^n = A^n e^{j i \theta} \quad j = \sqrt{-1} \]
  
  - Since we are linear, we don't have to worry about different modes "talking" to one another, so we can just deal with a single mode

- Stability will require that:
  \[ \left| \frac{A^{n+1}}{A^n} \right| \leq 1 \]

- Putting this mode into our difference equation we find (blackboard...)
  \[ \left| \frac{A^{n+1}}{A^n} \right|^2 = 1 + C^2 \sin^2 \theta \geq 1 \]
  
  - Note there is no dependence on \( n \) – all modes grow the same
  
  - No value of \( C \) makes things work
Fourier Mode Analysis of FTCS

- FTCS is unconditionally unstable
- Note that although this method of stability analysis only works for linear equations, it can still guide our insight into nonlinear equations
- This methodology was developed by von Neumann during WWII at LANL and allegedly, it was originally classified...
Truncation Analysis

- Observe that:

  *Finite-difference methods solve linear advection equations approximately, but they solve modified linear advection equations exactly*

  —Laney (p. 265)

- Let's figure out what physical equation this difference approximation better represents

- Substitute in

\[
\begin{align*}
  a_{i}^{n+1} &= a_{i}^{n} + \ddot{a} \Delta t + \frac{1}{2} \dddot{a} \Delta t^2 + O(\Delta t^3) \\
  a_{i \pm 1}^{n} &= a_{i}^{n} \pm a_x \Delta x + \frac{1}{2} a_{xx} \Delta x^2 + O(\Delta x^3)
\end{align*}
\]

- And lot's of algebra...
Truncation Analysis

• We find:

\[ a_t + u a_x = -\frac{1}{2} \Delta t u^2 a_{xx} \]

This is our original equation
This looks like diffusion, but look at the sign!

– We are more accurately solving an advection/diffusion equation
– But the diffusion is negative!
  • This means that it acts to take smooth features and make them strongly peaked—this is unphysical!
– The presence of a numerical diffusion (or numerical viscosity) is quite common in difference schemes, but it should behave physically!
Upwinding

- Let's go back to the original equation and try a different discretization

- Instead of a second-order (centered-difference) spatial difference, let's try first order.
  - We have two choices:
    - Let's try the upwinded difference—that gives:
      \[
      a_x = \frac{a^n_i - a^n_{i-1}}{\Delta x}
      \]
      \[
      a_x = \frac{a^n_{i+1} - a^n_i}{\Delta x}
      \]
      upwind downwind

- Let's try the upwinded difference—that gives:
  \[
  a^{n+1}_i = a^n_i - C(a^n_i - a^n_{i-1})
  \]
Upwinding

- Upwinding solution with $N = 65$, $C = 0.9$, after 1 period
  - Much better
  - There still is some error, but it is not unstable
  - Numerical diffusion is evident

code: fdadvect.py

PHY 688: Astrophysical Fluids and Plasmas
Upwinding

• Notice that if we do $C = 1$, we get an exact translation of the data from one zone to the next:

$$a_i^{n+1} = a_i^n - C(a_i^n - a_i^{n-1})$$

– However, as we will see, for nonlinear eqs. and systems of linear advection eqs., we cannot in general do $C = 1$, because there is not a single speed over all the grid.

• Let's play with the code and explore resolution, number of periods, etc...
Stability Analysis of Upwinded Eq.

- Truncation analysis would show this method is equivalent to
  \[ a_t + u a_x = \frac{1}{2} u \Delta x (1 - C') a_{xx} \]
  - Represents physical diffusion so long as \( 1 - C > 0 \)
  - This also shows that we get the exact solution for \( C = 1 \)

- Note that if we used the downwind difference, our method would be unconditionally unstable
  - Direction of difference based on sign of velocity

- Physically, the choice of upwinding means that we make use of the information from the direction the wind is blowing
  - This term originated in the weather forecasting community
Upwind Results

- Finite-difference (node-centered) grid, with N=65, 1 period:
  - Tophat initial conditions, $C = 0.8$
Upwind Results

- Finite-difference (node-centered) grid, with N=65, 5 periods:
  - Tophat initial conditions, $C = 0.9$

Notice that the numerical diffusion is strongly apparent here.

What about smooth initial conditions?

code: fdupwind.py
Upwind Results

- Finite-difference (node-centered) grid, with $N=65$, 1 period:
  - sine wave, $C = 0.9$

code: `fdupwind.py`
Upwind Results

- Finite-difference (node-centered) grid, with N=65, 5 periods
  - sine wave, $C = 0.9$

Note that the sine wave stays in phase (that's a good thing)

Diffusion still apparent.

Just for fun, let's try a downwind method...

code: fdupwind.py
Grid Types

“Regular” finite-difference grid. Data is associated with nodes spaced $\Delta x$ apart. Note that here we can have a point exactly on the boundary.

Cell-centered finite-difference grid. Here we consider cells of size $\Delta x$ and associate the data with a point at the center of the cell. Note that data will be $\Delta x/2$ inside the boundary.

Finite-volume grid—similar to the cell-centered grid, we divide the domain into cells/zones. Now we store the average value of the function in each zone.
Grid Types

- **Finite-difference**
  - Some methods use staggered grids: some variables are on the nodes (cell-boundaries) some are at the cell centers
  - Boundary condition implementation will differ depending on the centering of the data
  - For cell-centered f-d grids: ghost cells implement the boundary conditions

- **Finite-volume**
  - This is what we'll focus on going forward
  - Very similar in structure to cell-centered f-d, but the interpretation of the data is different.
Finite-Volume Grid

- We store the average of a quantity in a zone

\[
\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) \, dx
\]
Conservation Laws

- Many systems appear as conservation laws:
  \[ U_t + [F(U)]_x = 0 \]

  - Linear advection: \[ a_t + [ua]_x = 0 \]

  - Burgers’ equation: \[ u_t + \left[ \frac{1}{2}u^2 \right]_x = 0 \]

- Hydrodynamics

  \[ U = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix} \quad F(U) = \begin{pmatrix} \rho u \\ \rho uu + p \\ \rho uE + up \end{pmatrix} \]

Understanding advection is key to each of these systems.
Finite-Volume Approximation

- Finite-volume methods are designed for conservation laws
  - Integrate our hyperbolic equation over a control volume
    \[
    \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} U_t \, dx = -\frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial}{\partial x} F(U) \, dx
    \]
    \[
    \frac{\partial}{\partial t} U_i = -\frac{1}{\Delta x} \left\{ [F(U)]_{i+1/2} - [F(U)]_{i-1/2} \right\}
    \]
  - Here we write the cell-averages without the <>
  - **Telescoping property**
    - Note that the flux of $U$ that leaves one volume enters the adjacent one—guaranteeing conservation
  - We still need to discretize in time
Finite-Volume Approximation

• Two different approaches for handling time discretization
  – **Method of lines**
    • Now that we’ve discretized in space, we are left with an ODE in time
      \[
      \frac{\partial}{\partial t} U_i = -\frac{1}{\Delta x} \{ [F(U)]_{i+1/2} - [F(U)]_{i-1/2} \}
      \]
    • Use our ODE methods (like Runge-Kutta) on this
    • We’ll focus on this
  – Explicitly discretize in time using a difference approximation
    • Second order accuracy would require that the RHS is evaluated at
      the midpoint in time
    • This is covered in my online notes
Finite-Volume Approximation

- Let’s look at the linear advection equation
- And let’s focus on a simple forward Euler time update
Finite-Volume Approximation

• Back to linear advection, written in conservative form:

\[
\frac{\partial a}{\partial t} + \frac{\partial (ua)}{\partial t} = 0
\]

• Discretization of the integral form

\[
\frac{\partial a_i}{\partial t} = -\frac{1}{\Delta x} \left\{ [f(a)]_{i+1/2} - [f(a)]_{i-1/2} \right\}
\]

• Spatial discretization:
  – We need a single interface state
  – We can then evaluate the flux as:

\[
[f(a)]_{i+1/2} = f(a_{i+1/2})
\]
First-Order Advection

- Our interface states

\[ a_i \rightarrow a_{i+1/2},_L \quad \times a_{i+1/2},_R \rightarrow a_{i+1} \]

- Simplest choice:
  \[ a_{i+1/2},_L = a_i^n; \quad a_{i+1/2},_R = a_{i+1} \]

- Now we need to resolve the degeneracy of the states. This requires knowledge of the equation
  - Riemann problem: two states separated by an interface
  - For advection, we do upwinding
First-Order Advection

• Riemann problem (upwinding):

\[ R(a_{i+1/2,L}, a_{i+1/2,R}) = \begin{cases} 
    a_{i+1/2,L} & u > 0 \\
    a_{i+1/2,R} & u < 0
\end{cases} \]

- This is simple enough that we can write out the resulting update (for \( u > 0 \)):

\[
\frac{a_{i}^{n+1} - a_{i}^{n}}{\Delta t} = \frac{u a_{i}^{n} - u a_{i-1}^{n}}{\Delta x}
\]

- This is identical to the first-order upwind finite-difference discretization we already studied
Second-Order Finite Volume

- To get second order accurate, we need to be more accurate in both space and time

- There are two broad ways of doing this:
  - Predict our interface states at the midpoint in time using a Taylor expansion, resulting in a time-centered update
  - Use a method of lines formalism and an ODE integrator

- We’ll look at the MOL version (the other scheme is detailed in my notes online)
  - For time, we will use a second-order Runge-Kutta ODE Integration scheme
  - For space, we will replace our interface states with piecewise linear interpolated values
Second-Order Finite Volume

- **Time integration:**
  \[
  \dot{y} = f(y)
  \]

- **Two steps:**
  \[
  y^* = y^n + \frac{\Delta t}{2} f(y)
  \]
  \[
  y^{n+1} = y^n + \Delta t f(y^*)
  \]
Second-Order Finite Volume

- Simple piecewise linear estimates:

\[ \Delta a_i = \frac{a_{i+1} - a_{i-1}}{2} \]

\[ a_{i-1/2,R} = a_i - \frac{1}{2} \Delta a_i \]

\[ a_{i+1/2,L} = a_i + \frac{1}{2} \Delta a_i \]
Design of a General Solver

- Simulation codes for solving conservation laws generally follow the following flow:
  - Set initial conditions
  - Main evolution loop (loop until final time reached)
    - Get the timestep
    - RK stages loop:
      - Fill boundary conditions
      - Compute the interface states
      - Solve the Riemann problem
      - Compute conservative stage update

- We'll see that this same procedure can be applied to nonlinear problems and systems (like the equations of hydrodynamics)
• We need 2 ghost cells on each end:

  The state here depends on the slope in zone $lo-1$

  This slope requires the data from the zones on either size of it

• Let's look at the code...
Second-Order Advection

- Tophat with 64 zones, $C = 0.8$

Notice the oscillations —they seem to be associated with the steep jumps

code: fv_mol.py
Second-Order Advection

- Gaussian with 64 zones, \( C = 0.8 \)

This looks nice and smooth

code: fv_advection.py
Always look at the convergence of your code

- If you do not get what you are supposed to get, you probably have a bug (or you don't understand the method well enough...)

Convergence for the Gaussian

code: fv_advection.py
Second-Order Advection

- General notes:
  - We derived both the left and right state at each interface
    - For advection, we know that we are upwinding, so we only really need to do the upwinded state
  - General conservation law:
    - The upwind direction can change from zone to zone and timestep to timestep
    - There can be multiple waves (e.g. systems), so for the general problem, we need to compute both states
  - If you set the slopes to 0, you reduce to the first-order method
Summary So Far...

- Finite-volume grid stores average in a zone:

\[
\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) \, dx
\]

- Integration over space gives:

\[
\frac{\partial a_i}{\partial t} = -\frac{1}{\Delta x} \left\{ [f(a)]_{i+1/2} - [f(a)]_{i-1/2} \right\}
\]

- Where \( f(a) = ua \) is the flux
Alternate Approach

- Taylor expand the zone values in space and time to get time-centered interface values

- Second-order update:

\[
\frac{a_i^{n+1} - a_i^n}{\Delta t} = - \frac{[f(a)]_{i+1/2}^{n+1/2} - [f(a)]_{i-1/2}^{n+1/2}}{\Delta x}
\]

\[
[f(a)]_{i+1/2}^{n+1/2} = f(a_i^{n+1/2})
\]

- See my online notes for a discussion of this
No-limiting Example

Here we see an initial discontinuity advected using a 2\textsuperscript{nd}-order finite-volume method, where the slopes were taken as unlimited centered differences. Note the overshoot and undershoots.
Limiting

- Limiting: modify slopes near extrema to prevent overshoots.
  - Can drop method down to 1st-order accurate near discontinuities

- Godunov's theorem:
  - Any monotonic linear method for advection is first order accurate
  - To be monotonic and 2nd-order, we need to make our method nonlinear

- There are many limiters, derived by requiring that the update not introduce any new minima or maxima.
  - Mathematically enforced by a requirement of total variation diminishing
  - Ex of a simple limiter (minmod):
    \[
    \left. \frac{\partial a}{\partial x} \right|_i = \text{minmod} \left( \frac{a_i - a_{i-1}}{\Delta x}, \frac{a_{i+1} - a_i}{\Delta x} \right)
    \]
    \[
    \text{minmod}(a, b) = \begin{cases} 
    a & \text{if } |a| < |b| \text{ and } a \cdot b > 0 \\
    b & \text{if } |b| < |a| \text{ and } a \cdot b > 0 \\
    0 & \text{otherwise}
    \end{cases}
    \]
A finite-volume grid showing the cell averages (gray, dotted, horizontal lines), unlimited center-difference slopes (gray, solid) and MC limited slopes (red). Note that in zones $i$ and $i+1$, the slopes are limited slightly, so as not to overshoot or undershoot the neighboring cell value. Cell $i-1$ is not limited at all, whereas cells $i-2$, and $i+2$ are fully limited—the slope is set to 0—these are extrema.
Limiting Example

This is the same initial profile advected with limited slopes. Note that the profile remains steeper and that there are no oscillations.
There are many different limiters (examples below with 128 zones, 5 periods, $C = 0.8$)

code: fv_advection.py

These are from the tracing method shown in the notes.
Computational Fluid Dynamics

- We model our system of interest as a fluid
- Evolution dictated by conservation laws:

\[
\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0
\]
Conservation of mass

\[
\frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho uu + p)}{\partial x} = 0
\]
Conservation of momentum

\[
\frac{\partial (\rho E)}{\partial t} + \frac{\partial (\rho u E + up)}{\partial x} = 0
\]
Conservation of energy
Eigensystem

- A system is **hyperbolic** if the eigenvalues are real and finite.
- For our system, the eigenvalues are:
  \[ \lambda^{(-)} = u - c, \lambda^{(0)} = u, \lambda^{(+)} = u + c \]
  - These are the speeds at which information propagates in our system
  - Three distinct wave speeds for 3 equations
  - We'd get the same eigenvalues from the Jacobian of the conserved system
- There is a rich mathematical description of the theory of hyperbolic systems of conservation laws.
  - The book by LeVeque is an excellent introduction.
Eigensystem

- We can also find the eigenvectors:

\[ A r^{(\nu)} = \lambda^{(\nu)} r^{(\nu)} ; \quad l^{(\nu)} A = \lambda^{(\nu)} l^{(\nu)} \]

\[
\begin{align*}
    r^{(-)} &= \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \\
    r^{(\circ)} &= \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\
    r^{(+)} &= \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
    l^{(-)} &= \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \\
    l^{(\circ)} &= \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \\
    l^{(+)} &= \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix}
\end{align*}
\]

- These are normalized such that \( l^{(i)} \cdot r^{(j)} = \delta_{ij} \)
Characteristic Variables

- A final form of the system is in terms of the characteristic variables
  - Construct matrices of the left and right eigenvectors
    \[
    R = (r^{(-)} | r^{(\circ)} | r^{(+)})
    \]
    \[
    L = \begin{pmatrix}
    l^{(-)} \\
    l^{(\circ)} \\
    l^{(+)}
    \end{pmatrix}
    \]
  - Satisfy: \( LR = RL = I \)
  - Define \( \mathbf{d}w = L \mathbf{d}q \)
  - Our system can be written as:
    \[
    \mathbf{w}_t + \Lambda \mathbf{w}_x = 0
    \]

Blackboard derivation...
Characteristic Variables

- Here, \( w \) are the characteristic variables
- The three equations are decoupled:

\[
\Lambda = LAR = \begin{pmatrix}
\lambda^(-) \\
\lambda^o \\
\lambda^+(+) \\
\end{pmatrix}
\]

- If our system were linear, we'd be done:
  - Transform into characteristic variables
  - Each characteristic variable advects at a given wave speed, without interacting with one-another
  - Solve and then transform back to primitive form (or conserved form)
- We're non-linear: the wave-speeds and eigenvectors change with the solution
Jumps Across Waves

• The characteristic system is telling us something interesting already.
  – Consider an initial discontinuity in the primitive variables
  – Each wave will carry a jump in their associate characteristic quantity away from the discontinuity at their speed
  – The corresponding jump in the primitive variable is just \( dq = L^{-1}dw = R \, dw \)
PHY 688: Astrophysical Fluids and Plasmas

Jumps Across Waves

\[ \lambda_1 = u - c \quad \lambda_2 = u \quad \lambda_3 = u + c \]

\[ q_l \quad \lambda \quad q_r \]

\[ r^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \quad r^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad r^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix} \]

- All primitive quantities jump across the u-c wave
- Only density jumps across the middle (u) wave
- All primitive variables jump across the u+c wave
$\rho_L = 1.0$

$\rho_R = 1/8$

$u_L = 0$

$u_R = 0$

$p_L = 1.0$

$p_R = 1/10$
Solution Methodology

- This system has similarities to the advection equation we already studied.
  - We can take a similar approach

- We will use the finite-volume discretization:

\[
\frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} U_t = -\frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial}{\partial x} F(U) dx
\]

\[
\frac{\partial}{\partial t} U_i = -\frac{1}{\Delta x} \left\{ [F(U)]_{i+1/2} - [F(U)]_{i-1/2} \right\}
\]

  - Again, we’ll use a method of lines approach for this
Solution Methodology

- Note: in order to get the shock speed correct, we need to solve this in conservative form.

- Basic idea: evaluate fluxes at interfaces by predicting the fluid state there, then solve Riemann problem:

$$ F_{i+1/2} = F(U_{i+1/2}) $$

$$ U_{i+1/2} = \mathcal{R}(U_{i+1/2,L}, U_{i+1/2,R}) $$

\[\text{\includegraphics{solution_method.png}}\]
Interface State Prediction

- The simplest method to construct the interface states is to simply use the cell averages

\[ U_{i+1/2,L} = U_i ; \quad U_{i+1/2,R} = U_{i+1} \]

- This is first order accurate in space and time
- This method is called Godunov's method
- Doesn't consider how U is changing in space or over the timestep
Riemann Problem

- No matter the method used to predict the interface states, we now have left and right states at each interface.

\[ U_i \quad \begin{array}{c} F(U_{i+1/2}) \\ \hline \end{array} \quad U_{i+1/2,L} \times \quad \times U_{i+1/2,R} \quad U_{i+1} \quad \begin{array}{c} \quad i \\ \hline \quad i + \frac{1}{2} \\ \hline \quad i + 1 \end{array} \]

- Unlike the linear advection or Burger's equation, we rarely solve the Riemann problem for the Euler equations exactly.
  - We need to consider what is carried by each wave.
  - Different types of waves are present depending on the behavior of the characteristics.
Riemann Problem

- Wave type is determined by whether the characteristics converge, are parallel, or diverge

- We already saw shocks and rarefactions with Burger's equation
- A contact discontinuity is where the solution jumps, but there is no compression or expansion
Riemann Problem

- Across the middle wave ($\lambda = u$), only the density jumps

\[
    r^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \quad r^{(0)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad r^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix}
\]

- So the velocity is the same on either side—no convergence or divergence.
- The middle wave is always a contact discontinuity

- The left and right waves can be either a shock or rarefaction
• The Riemann problem for the Euler equations looks like:

\[ dx = (u - c)dt \quad \text{and} \quad dx = (u + c)dt \]

\[ x = ut \]

\[ \rho_{*, L}, u_{*}, p_{*} \]

\[ \rho_{L}, u_{L}, p_{L} \]

\[ \rho_{*, R}, u_{*}, p_{*} \]

\[ \rho_{R}, u_{R}, p_{R} \]

- Solving the Riemann problem means finding the 4 quantities:

\[ \rho_{*, L}, \rho_{*, R}, u_{*}, p_{*} \]
Wave Structure

- There are several different wave configurations

1-rarefaction and 3-shock

1-shock and 3-rarefaction

two rarefactions

two shocks

PHY 688: Astrophysical Fluids and Plasmas
Wave Structure

- And the can span the initial interface or all be on one side

supersonic flow to the left

supersonic flow to the right
Riemann Problem

- The basic solution idea is to link the left and right states (which we know) to the star state using our understanding of what happens across the left and right waves
Riemann Problem: Rarefaction

- **If we are not compressing, then we are a rarefaction**

- Let's look at the system of equations with entropy instead of pressure as a variable.
  - Entropy evolution: \( \frac{DS}{Dt} = 0 \)
  - We need to replace the pressure gradient in the momentum equation:

\[
\frac{\partial p(\rho, s)}{\partial x} = \frac{\partial p}{\partial s} \bigg|_{\rho} \frac{\partial s}{\partial x} + \frac{\partial p}{\partial \rho} \bigg|_{s} \frac{\partial \rho}{\partial x} = \frac{\partial p}{\partial s} \bigg|_{\rho} \frac{\partial s}{\partial x} + \frac{p \Gamma_1}{\rho} \frac{\partial \rho}{\partial x}
\]
Riemann Problem: Rarefaction

- System becomes:

\[
\frac{\partial \rho}{\partial t} + u \frac{\partial \rho}{\partial x} + \rho \frac{\partial u}{\partial x} = 0
\]

\[
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + \frac{1}{\rho} \left[ \frac{\partial p}{\partial s} \left|_{\rho} \right. \frac{\partial s}{\partial x} + \frac{p \Gamma_1}{\rho} \frac{\partial \rho}{\partial x} \right] = 0
\]

\[
\frac{\partial s}{\partial t} + u \frac{\partial s}{\partial x} = 0
\]

or

\[
\begin{pmatrix}
\rho \\
u \\
s
\end{pmatrix}_t + \begin{pmatrix}
0 & 0 & 0 \\
c^2 / \rho & u & \frac{1}{\rho} \frac{\partial p}{\partial s} \left|_{\rho} \right. \\
0 & 0 & u
\end{pmatrix}_x
\begin{pmatrix}
\rho \\
u \\
s
\end{pmatrix}
\]
Riemann Problem: Rarefaction

- The eigenvalues of that matrix are the same, but the eigenvectors will tell us how entropy behaves across each wave:

\[
\begin{align*}
\mathbf{r}^{(-)} &= \begin{pmatrix} 1 & -c/\rho \\ -c/\rho & 0 \end{pmatrix} \\
\mathbf{r}^{(\circ)} &= \begin{pmatrix} 1 & 0 \\ 0 & -c^2/p_s \end{pmatrix} \\
\mathbf{r}^{(+)} &= \begin{pmatrix} 1 & c/\rho \\ c/\rho & 0 \end{pmatrix}
\end{align*}
\]

- Entropy only jumps across the contact, not the left and right way

- Considering entropy is only really useful for rarefactions
  - Shocks involve dissipative processes, so entropy needs to increase
  - This is why you need to solve a total energy instead of an entropy equation for shocks
Riemann Problem: Rarefaction

- Consider waves originating at a discontinuity (this is the Riemann problem)

- We come in with a jump in the primitive variables, $q$
  - The corresponding jump in characteristic variables is $\Delta w = L \Delta q$
  - Each wave carries a jump in its characteristic variable

- Consider the right (+) wave:
  - It moves at speed $u + c$
  - A piece of $\Delta q$ will jump across this wave (let's call the quantity that jumps $w^{(+)}$)
  - But $w^{(+)}$ will be constant across the other waves then
  - We can use this to link the states across the left (-) wave
Riemann Problem: Rarefaction

- \( w^{(+)} \) is constant across the (-) wave. This is defined as
  \[
  l^{(+)} \cdot dq = 0
  \]

- This is:
  \[
  \begin{pmatrix}
  0 & \frac{\rho}{2c} & \frac{1}{2c^2}
  \end{pmatrix}
  \begin{pmatrix}
  \frac{d\rho}{du} \\
  \frac{du}{dp}
  \end{pmatrix}
  = 0
  \]
  \[
  du + \frac{dp}{\rho c} = 0
  \]

- The general solution to this is:
  \[
  u = -\int \frac{dp}{\rho c}
  \]
  
  - We can't integrate this for a general EOS
Riemann Problem: Rarefaction

- If we assume a gamma-law ideal gas, then we know
  \[ p = K \rho^\gamma \]
  - for isentropic flows
  - \( \gamma \) is the constant ratio of specific heats

- Integration of this gives (blackboard):
  \[ u + \frac{2c}{\gamma - 1} = \text{constant} \]  
  across left rarefaction

- Similarly, we'd find
  \[ u - \frac{2c}{\gamma - 1} = \text{constant} \]  
  across right rarefaction
Riemann Problem: Rarefaction

- We can now connect the left and star state across a rarefaction:

\[
    u_L + \frac{2c_L}{\gamma - 1} = u_* + \frac{2c_*}{\gamma - 1}
\]

\[
    u_* = u_L + \frac{2c_L}{\gamma - 1} \left[ 1 - \left( \frac{c_*}{c_L} \right) \right]
\]

- We also know:

\[
    \frac{p_L}{\rho_L^\gamma} = \frac{p_*}{\rho_*^\gamma}
\]

- So:

\[
    \frac{c_*}{c_L} = \left( \frac{p_* \rho_l}{p_L \rho_*} \right)^{1/2} = \left( \frac{p_*}{p_L} \right)^{(\gamma - 1)/2\gamma}
\]

and

\[
    u_* = u_L + \frac{2c_L}{\gamma - 1} \left[ 1 - \left( \frac{p_*}{p_L} \right)^{(\gamma - 1)/2\gamma} \right]
\]
Riemann Problem: Shock

- If the flow is compressing, then we have a shock
  - Rankine-Hugoniot jump conditions provide the solution
  - We already worked this out
- Recall: in the frame of the shock
  - Jump conditions (for left wave) become:
    \[ \frac{\hat{F}_* - \hat{F}_L}{\hat{U}_* - \hat{U}_L} = 0 \]
    \[
    \hat{u}_L = u_L - S_L \\
    \hat{u}_* = u_* - S_L
    \]
    \[
    \begin{aligned}
    q_L & \quad q_{*,L} \\
    S_L &
    \end{aligned}
    \]
Riemann Problem: Shock

- Jump conditions are:
  \[
  \rho_L \hat{u}_L = \rho_* \hat{u}_*
  \]
  \[
  \rho_L \hat{u}_L^2 + p_L = \rho_* \hat{u}_*^2 + p_*
  \]
  \[
  \rho_L e_L \hat{u}_L + p_L \hat{u}_L + \frac{1}{2} \rho_L \hat{u}_L^3 = \rho_* e_* \hat{u}_* + p_* \hat{u}_* + \frac{1}{2} \rho_* \hat{u}_*^3
  \]

- With a bit of algebra, we have:
  \[
  [\tau] = -\frac{[p]}{W_L^2}
  \]
  \[
  [u] = -\frac{[p]}{W_L}
  \]
  \[
  [e] = -\bar{\rho}[\tau]
  \]
  \[
  [q] \equiv q_* - q_L
  \]
  \[
  \tau = 1/\rho
  \]
  \[
  \bar{\rho} = \frac{1}{2}(p_L + p_*)
  \]
  \[
  W_L = \rho_L \hat{u}_L = \rho_* \hat{u}_*
  \]

Note that: \( \hat{u}_* - \hat{u}_L = u_* - u_L \)

This is the mass flux
Riemann Problem: Shock

- How do we solve these?
  - We will come in with $p_*$ (more on this in a minute)
  - Root find on:
    - $e_* = e(p_*, \rho_*)$
    - $e_* - e_L = -\frac{1}{2}(p_* + p_L) \left( \frac{1}{\rho_*} - \frac{1}{\rho_L} \right)$

- Get the mass flux,
  \[ \frac{1}{W_L^2} = -\frac{[\tau]}{[p]} \]

- Get the star velocity:
  \[ u_* = u_L - \frac{p_* - p_L}{W_L} \]
Summary of Riemann Problem

- We need to consider the action of all 3 waves
  - Middle wave is always a contact
  - Left and right can be either shock or rarefaction
- Rarefaction:
  - Entropy is constant across the wave
  - Riemann invariants tell us how to connect the solution across the wave to the star region
- Shock:
  - Must be dissipative
  - Jump conditions tell us how to connect to the star state across the shock
Wave Structure

- The characteristic structure becomes:

\[ q_{\ast},L \quad u = \text{const} \quad p = \text{const} \]

1-rarefaction

\[ q_{L} \quad u + \frac{2c}{\gamma - 1} = \text{const} \]

Rankine-Hugoniot conditions

Using the correct relation across each wave allows us to link the 4-states and solve the Riemann problem.
For a gamma-law gas, the shock solution can be found analytically, and we get:

\[ u_{*,L}(p_*) = u_L + \begin{cases} \frac{2c_L}{\sqrt{2\gamma(\gamma-1)}} \frac{1-p_*/p_L}{\sqrt{1+\frac{\gamma+1}{\gamma-1} \frac{p_*}{p_L}}} & p_* > p_L \\ \frac{2c_L}{\gamma-1} \left[ 1 - \left( \frac{p_*/p_L}\right)^{(\gamma-1)/2\gamma} \right] & p_* \leq p_L \end{cases} \]

\[ u_{*,R}(p_*) = u_R - \begin{cases} \frac{2c_R}{\sqrt{2\gamma(\gamma-1)}} \frac{1-p_*/p_R}{\sqrt{1+\frac{\gamma+1}{\gamma-1} \frac{p_*}{p_R}}} & p_* > p_R \\ \frac{2c_R}{\gamma-1} \left[ 1 - \left( \frac{p_*/p_R}\right)^{(\gamma-1)/2\gamma} \right] & p_* \leq p_R \end{cases} \]

The constancy of the pressure and velocity across the contact gives:

\[ u_{*,L}(p_*) = u_{*,R}(p_*) \]
Riemann Solution

- This can be solved using standard root finding techniques
- Gets you $u_*, p_*$
- The density to the left and right of the contact can be found via the Riemann invariants (rarefaction) or jump conditions (shock)
  - This completes the entire solution
Riemann Solution

- We can look at the solution to the Riemann problem graphically

- Sod problem:
  \[ \rho_L = 1, u_L = 0, p_L = 1 \]
  \[ \rho_R = 1/8, u_R = 0, p_R = 1/10 \]

  - Plot shows the curves that each state can reach through a shock (dashed) or rarefaction (solid)
  - Solution is the intersection
  - Here we must have a left rarefaction and a right shock

\[
\begin{align*}
p^* &= 0.303130178051 \\
u^* &= 0.927452620049
\end{align*}
\]
\[ \rho_L = 1.0 \]
\[ \rho_R = 1/8 \]
\[ u_L = 0 \]
\[ u_R = 0 \]
\[ p_L = 1.0 \]
\[ p_R = 1/10 \]
Sampling the Solution

- We only need to know the Riemann solution on our interface

- Which region are we in?
  - We need to evaluate the various speeds
Sampling the Solution

- If the left or right wave is a shock, the speed comes from our jump conditions
  - Left: we had
    \[ W_L = \rho_L \hat{u}_L = \rho_L (u_L - S) \rightarrow S = u_L - \frac{W_L}{\rho} \]
  - An analytic expression for this can be written down for a gamma-law gas
  - A similar expression (but with a '+') is found for a shock at the right wave
- For a contact, we just use \( S_0 = u_\ast \)
- For a rarefaction, there are 2 speeds
  - Consider the left rarefaction
    - Head travels at: \( S_h = u_L - c_L \)
    - Tail travels at: \( S_t = u_\ast - c_\ast \)
Sampling the Solution

- To complete the rarefaction speed we need the density in the star region:
  - Gamma-law gas:
    - We know entropy is constant, so we can just do:
      \[
      \frac{p_L}{\rho_L^{\gamma}} = \frac{p_*}{\rho_*^{\gamma, L}}
      \]
  - General gas:
    - Use the contact's characteristic variable:
      \[
      l^{(\circ)} \cdot dq = \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \cdot \begin{pmatrix} d\rho \\ du \\ dp \end{pmatrix} = 0
      \]
      \[
      d\rho = \frac{dp}{c^2}
      \]
      - This can be integrated across the rarefaction
Sampling the Solution

- We evaluate the solution to the Riemann problem along $x/t = 0$

  - If $S_0 > 0$, then we are either the left or left-star state
    - The check the speed of the left shock / rarefaction to determine left or left-star
      - This gives you the state on the interface
  - If $S_0 < 0$, then we are either the right or right-star state
Sampling the Solution

- Caveat: the rarefaction could span the axis
  - \( S_{\text{head}} < 0 < S_{\text{tail}} \)
  - We can integrate the Riemann invariants to get the state inside the rarefaction
Once we get the state on the interface, we can construct the fluxes through the interface:

\[
F_{i+1/2} = \begin{pmatrix}
\rho_{i+1/2} u_{i+1/2} \\
\rho_{i+1/2}(u_{i+1/2})^2 + p_{i+1/2} \\
\frac{u_{i+1/2} p_{i+1/2}}{(\gamma - 1)} + \frac{1}{2} \rho_{i+1/2}(u_{i+1/2})^3 + u_{i+1/2} p_{i+1/2}
\end{pmatrix}
\]
General Solution

• We solve the Euler equations for a general problem (not a single jump) by constructing the interface states and then solving a Riemann problem at each interface
  – The Riemann problem captures the shock—we get the shock speed correct
  – The overall method is conservative
Structure of a Simulation Code

- The structure of your code is nearly identical to the codes we did for advection:
  - Initialize
    - We need to initialize all the conserved state, U
  - Evolve
    - Compute timestep
      - We restrict things so that no information can cross more than one zone in a timestep:
        \[ \Delta t = \min_i \frac{\Delta x}{|u_i| + c} \]
    - Fill ghost cells
      - The number of ghost cells will depend on the width of our stencil
      - Piecewise linear slopes need 2 ghost cells on each end of the domain
Evolve (continued)

- Construct interface states
  - We do this on the primitive variables
  - We wind up with a left and right primitive variable state at every interface

![](image)

- Solve Riemann problem
  - This takes the time-centered, edge-centered primitive variable states and returns the state on the interface
Structure of a Simulation Code

- Evolve (continued)
  - Conservative update:
    \[
    U_{i}^{n+1} = U_{i}^{n} + \frac{\Delta t}{\Delta x} (F_{i-1/2}^{n+1/2} - F_{i+1/2}^{n+1/2})
    \]
Boundary Conditions

• Boundary conditions should match the physics you are modeling

• Common ones:
  – **Reflecting / solid wall**: ghost cells reflect their partner across the physical boundary. Normal velocity component changes sign
  – **Outflow**: zero-gradient given to all ghost cells
    • Note: this is not perfect, because at the boundary, some characteristic waves may flow inward (for subsonic flows)
  – **Hydrostatic**: in the presence of gravity, provide pressure support to hold up an atmosphere
Structure of a Simulation Code

- Solving the Riemann problem (even for a gamma-law gas) can be expensive
  - Often we use approximations
    - **Approximate state methods:**
      - Approximate the primitive variable state in the star region
      - Some examples: two-shock and two-rarefaction solvers
    - **Approximate flux methods:**
      - Approximate the flux directly from the jump conditions without getting an approximate state
      - Some examples: HLL methods
  - **Linearized methods:**
    - Linearize the system and solve the Riemann problem as you would for a linear hyperboic system
    - Example: Roe solver
Bells and Whistles

- We described the basics, but there are some other features typically implemented in hydro codes:
  - **Flattening**: shocks self-steepen (characteristics converge), and can become too sharp. Flattening adds some dissipation near shocks to smear them out a bit.
  - **Species advection**: nuclear or chemical species can be advected with the flow:
    \[
    \frac{\partial X_k}{\partial t} + u \frac{\partial X_k}{\partial x} = 0
    \]
  - **Source terms**: source terms (e.g. gravity) do not change the characteristic structure, but do participate in the prediction to the interfaces.
  - **Other geometries**: extensions to cylindrical and spherical geometries involves volume and area factors throughout the code.
Test Problems

- There are several standard test problems with known analytic solutions
- Simplest is to just run a Riemann problem (shock tube) with your solver
- I’ll show some solutions using a method that does characteristic tracing, not method of lines
  - You’ll do the method of lines version for homework
Sod Problem

- Sod problem exhibits all 3 types of waves
  \[
  \rho_L = 1, u_L = 0, p_L = 1 \\
  \rho_R = 1/8, u_R = 0, p_R = 1/10
  \]

- Setup:
  - 128 zones
  - Piecewise constant reconstruction
  - 2-shock approximate Riemann solver

- Notice how smeared out the contact is
Sod Problem

- With piecewise parabolic profiles, all the waves look better.
- Contact is still the worst—no self-steepening.
Double Rarefaction

- Divergent flow:
  \[
  \rho_L = 1, u_L = -2, p_L = 0.4 \\
  \rho_R = 1, u_R = 2, p_R = 0.4
  \]

- This generates a vacuum state

- Setup:
  - 128 zones
  - Piecewise constant reconstruction
  - 2-shock approximate Riemann solver

code: hydro1d
Double Rarefaction

- Much better flow with ppm
Double Rarefaction

- But solvers have a hard time getting the internal energy right in the vacuum region
  - e from ppm solution
Strong Shock

- Strong shock followed closely by contact
  \[ \rho_L = 1, u_L = 0, p_L = 1000 \]
  \[ \rho_R = 1, u_R = 0, p_R = 0.01 \]

- Very difficult to get the shock structure correct

- Setup:
  - 128 zones
  - Piecewise constant reconstruction
  - 2-shock approximate Riemann solver
• We do better with ppm reconstruction

![Graphs showing piecewise parabolic plots of shock wave properties.](image-url)
Different Coordinate Systems

- Consider 1-d spherical coordinates

\[ \frac{\partial U}{\partial t} + \frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 F \right) = 0 \quad \Rightarrow \quad \frac{\partial U}{\partial t} + \frac{\partial F}{\partial r} = -\frac{2F}{r} \]

  - This looks like a Cartesian system with a geometric source term

- Strategy:

  - When doing the prediction to the interfaces, explicitly include the geometric source term

\[ \frac{\partial \rho}{\partial t} + u \frac{\partial \rho}{\partial r} + \rho \frac{\partial u}{\partial r} = -\frac{2\rho u}{r} \]
\[ \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial r} + \frac{1}{\rho} \frac{\partial p}{\partial r} = 0 \]
\[ \frac{\partial p}{\partial t} + u \frac{\partial p}{\partial r} + \gamma p \frac{\partial u}{\partial r} = -\frac{2\gamma p u}{r} \]

- Eigenvectors don't change here

PHY 688: Astrophysical Fluids and Plasmas
I have 2 codes online that implement these methods:

- **hydro1d**:
  - Implements the Godunov, PLM, and PPM hydro methods in 1-d
  - Gravity source terms
  - Written in Fortran 95/2003

- **pyro**:
  - Implements 2-d PLM compressible hydro (dimensionally unsplit)
  - Also does 2-d advection, 2-d incompressible, 2-d multigrid, 2-d diffusion
  - Written mostly in python, with some Fortran (through f2py)